

Industria y Comercio
SUPERINTENDENCIA

Organizacion CSA Ltda

SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO
INVENTARIO

Nov
26

SOLICITUD

PATENTE DE INVENCION

21. EXPEDIENTE N°

03-46883

54. TÍTULO

SISTEMA Y METODO PARA ASOCIAR PROPIEDADES CON OBJETOS

51. CLASIFICACIÓN INTERNACIONAL

⁶ E06F 19/00

1. SOLICITANTE

MICROSOFT CORPORATION

DOMICILIO

REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA

4. APODERADO

DARIO CARDENAS NAVAS

2. BOGOTÁ, D. C.

2003-VI-04

03-468839

04

PETITORIO

Industria y Comercio
SUPERINTENDENCIA



03 046839 00

SUPERINDUSTRIA Y COMERCIO
Radicación: 03046839 00000000 Folios: 254
Fecha (HRB): 2003-06-04 13:52:59
Trámite: 002 PATENTES D 1 REGISTRO/ 411 PRESENTAR
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

FORMULARIO UNICO DE SOLICITUD DE PATENTE 2000-3

1

SOLICITUD DE:

☒ X

Patente de Invención

☐

Patente de Modelo de Utilidad

2

SOLICITANTE
(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 ☒ X

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. ☒ X NIT ☐

C.E. ☐ Otro ☐

Número 17.066.629 BTA
TP 1.250 C.S.J.

4

INVENTOR(ES)
(72)

Nombre:
1. JEFFREY L. BOGDAN
2. MARK J. FINOCCHIO
3. NICHOLAS M. KRAMER
Dirección:
1. 25936 NE 29th Place, Redmond, Washington 98053
2. 4631 168th Lt. NE, Redmond, Washington 98052
3. 3728 Wallingford Ave. Seattle, Washington 98103

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 ☒ X

Número _____

5

Título(54):

SISTEMA Y METODO PARA ASOCIAR PROPIEDADES CON OBJETOS

6

Clasificación Internacional (51):

7

Prioridad

SI ☒ X

NO ☐

(33)País de Origen

ESTADOS UNIDOS
DE AMERICA

(32)Fecha

28de Junio, 2002

(31)N° de Solicitud

10/187,012

8

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 - 37,517 / 03-37,520

Fecha Mayo 28, 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. (No
- ☒ 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.

11

NOMBRE:

DARIO CARDENAS NAVAS

FIRMA:



C.C. 17.066.629 BTA

TP. 1.250

C.SJ.

ar

SUPERINDUSTRIA Y COMERCIO

Radicacion : 03046839 00000000

Folios: 254

Fecha (AMD): 2003-06-04 13:52:59

Tramite : 002 PATENTES D 1 REGISTRO 411 PRESENTAC

Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 03 - 37,517

FECHA : MAYO 28 DE 2003

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGO	Vr. PAGO
CARDENAS Y CARDENAS P.I. CONSIGNACION		BANCO POPULAR	050-00110-5	13821544	28/05/2003	1,651,000.00

***** CONCEPTO *****

CANT. RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1 50005-01-01 SOLICITUDES	1 TRAMITES 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. _____

SISTEMA Y METODO PARA
ASOCIAR PROPIEDADES CON
OBJETOS.

CARDENAS & CARDENAS
ABOGADOS

4

SUPERINDUSTRIA Y COMERCIO

Radicacion : 03046839 00000000

Folios: 234

Fecha (AMD): 2003-06-04 14:52:53

Tramite : 002 PHENIES D 1 REGISTRO/ 411 PRESENTA

Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 03 - 37,520

FECHA : MAYO 28 DE 2003

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	Nº. PAGO	FECHA PGO	Gr. PAGO
CARDENAS Y CARDENAS P.I.	CONSIGNACION	BANCO POPULAR	050-0011076	13821544	28/05/2003	1,651,000.00

***** CONCEPTO *****

CANT.	RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1	50005-01-01	SOLICITUDES	82 INVOCACION DE UNA PRIORIDAD EN NU
			TOTAL : \$ 112,000.00

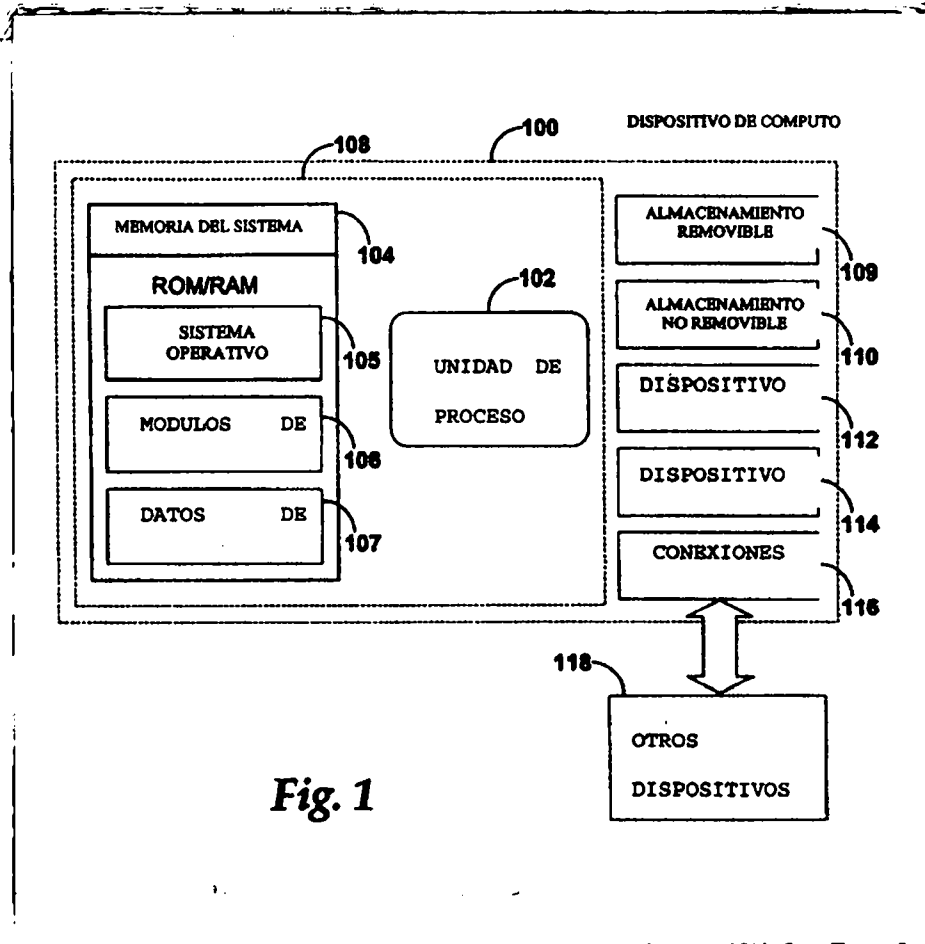
SON: CIENTO DOCE MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

SISTEMA Y METODO PARA
ASOCIAR PROPIEDADES CON
OBJETOS.

CARDENAS & CARDENAS
ABOGADOS



arte para
publicar

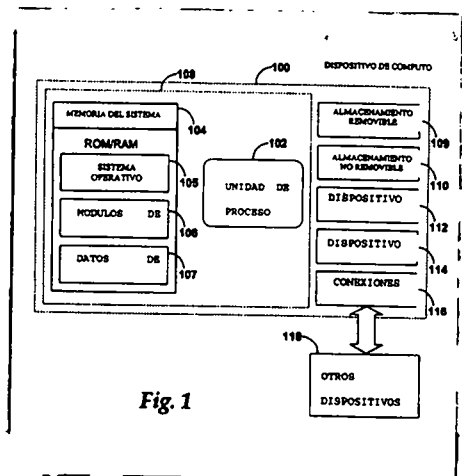


Fig. 1

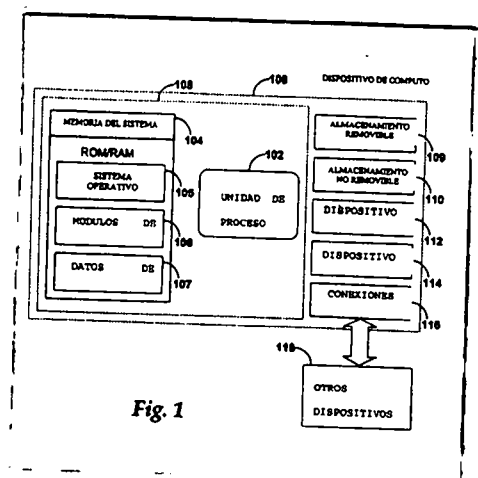


Fig. 1

7

SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS

Field of the Invention

The present invention relates generally to software applications, and,
5 more particularly, to mechanisms for managing properties of objects within a software application.

Background of the Invention

Most programming models today support the concept of classes. These classes are typically structured in a hierarchical tree having branches that
10 represent different classes in the class hierarchy. When two branches are at different levels, the lower branch represents a child class. The child class inherits information from the class associated with the upper branch (i.e., the parent class). When two branches are at the same level, the classes are referred to as sibling classes. For the purposes of the following discussion, when referring to a child class in relation to a
15 parent class, the terms, lower class and upper class, may be used to refer to the child class and the parent class, respectively. The uppermost branch in the hierarchy represents a base class in the hierarchical class tree. Typically, information in each class includes properties, methods, and events. The properties describe characteristics associated with the class. For example, a button class may have
20 properties such as width, background color, font type, visible, and pressed. When one of these classes is instantiated, an object of that class is created. Each property in the object has an associated value, which can be queried and set during run-time operation. The value that is queried or set may be expected to conform to a specific data type if the syntax is strongly typed. Having syntax that is strongly typed is
25 desirable because errors can be detected within a software application before run-time operation.

Once the class hierarchy is in place, adding new functionality to objects within the class hierarchy presents a problem. In one programming model, the new functionality is forced into the base class. When this is done, the base class
30 becomes very large (e.g., one hundred methods, fifty properties, and twenty events), which results in an almost unmanageable object hierarchy. One undesirable outcome of this programming model is that the number of properties, methods, and

events within the base class becomes overwhelming for developers to fully understand before implementing desired features into objects they create. Yet another undesirable outcome of this programming model is that the memory requirements become enormous due to the fact that values for the properties are stored locally. Because the values are stored locally, applications created with this programming model do not scale well.

This programming model also causes other problems for third party developers. Third party developers who wish to add new functionality must add a child class at the bottom of the hierarchy. Because the new child class is at the bottom of the hierarchy, the new functionality is not available to other classes within the hierarchy. Therefore, third party developers may need to add the new functionality to several child classes. As one can imagine, this results in code duplication, which impacts the maintainability of the object hierarchy. For the above reasons, this programming model is not very desirable.

Until the present invention, a programming model that allowed new functionality to be provided to a class within an existing class hierarchy without the above shortcomings has eluded those skilled in the art.

Summary of the Invention

The present invention provides a mechanism that allows new functionality to be provided to a class without having the new functionality become a permanent part of the class. In addition, the present invention allows the new functionality to be expressed as a property that is not built into the class. In general, the present invention provides a mechanism for associating properties in one class with another class. This association is easily modifiable so that other sets of properties may be associated with the class.

In one embodiment, a computer-readable medium that includes an object having a property in a first set of properties, further includes a data structure. The data structure includes definitions for each of a second set of properties and includes at least one static method. The static method is associated with one property out of the second set of properties and includes a first parameter. The first parameter uniquely identifies the one property. The static method is operative to

associate the one property with the object without specifying an explicit reference to the one property in the object.

In one aspect of the invention, the static method supports a strongly typed syntax.

5 In another aspect of the invention, the static method includes retrieving a value for the object without having the value stored locally on the object. The value may be retrieved from several stages, such as a parent object or a property sheet.

10 One advantage of the present invention is that the dividing of properties into one or more subsets allows each subset of properties to be more easily maintained. Another advantage of the present invention is that the object hierarchy is more extensible and allows developers to add functionality to the object hierarchy that impacts the base class and all lower classes.

15 Another advantage of the present invention is that managing storage for the properties in the objects becomes more efficient and convenient.

Yet another advantage of the present invention is that the programming model operates within a programmatic or a markup environment. In addition, the programming model operates in strongly typed programming languages, such as C++ and C#. In addition, the programming model supports
20 property sheets, change notifications and value inheritance.

Another advantage of the present invention is that independent libraries can coexist because name conflicts are less likely because the name of the attached class effectively becomes part of the property name. Therefore, if two different developers each create an attached property having a property named
25 "color", the two attached properties will not conflict.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention.

30 FIGURE 2 is an exemplary display that may be created with the computing device of FIGURE 1.

FIGURE 3 is a graphical representation of a programming model that allows properties from one class to be attached to another class in accordance with the present invention.

FIGURE 4 illustrates several exemplary syntaxes for implementing
5 the programming model shown in FIGURE 3.

FIGURE 5 is a logical flow diagram illustrating a process for setting a value in accordance with the present invention.

FIGURE 6 is a logical flow diagram illustrating a process for retrieving a value in accordance with the present invention.

10

Detailed Description of the Preferred Embodiment

Briefly stated, the present invention provides a programming model that allows new functionality to be provided to a class without having the new functionality become a permanent part of the class. In addition, the present invention allows the new functionality to be expressed as a property that is not built
15 into the class. In general, the present invention provides a mechanism for associating properties in one class with another class. As will become apparent after reading the detailed description below, the programming model of the present invention provides dynamic properties to objects without building the properties into the object.

20

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device 100. In a very basic configuration, computing device 100 typically includes at least one processing unit 102 and system memory 104. Depending on the exact configuration and type of computing device, system memory 104 may be volatile
25 (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory 104 typically includes an operating system 105, one or more program modules 106, and may include program data 107. Examples of program modules 106 include Visual Studio IntelliSense from Microsoft Corporation of Redmond, WA, and other software programming
30 environments, which utilize object libraries. In addition, program modules 106 include software applications created using a software-programming environment. When these software applications execute on processing unit 102, a property engine

processes the software application in accordance with the programming model of the present invention. The property engine may be part of operating system 105 or may be another program module 106. This basic configuration of computing device 100 is illustrated in FIGURE 1 by those components within dashed line 108.

5 Computing device 100 may have additional features or functionality. For example, computing device 100 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage 109 and non-removable storage 110. Computer storage media may include
10 volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 104, removable storage 109 and non-removable storage 110 are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM,
15 EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 100. Any such computer storage media may be part of device 100.
20 Computing device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 114 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here.

 Computing device 100 may also contain communication connections
25 116 that allow the device to communicate with other computing devices 118, such as over a network. Communication connections 116 is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any
30 information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and

wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

FIGURE 2 is an exemplary display that may be created by an application within the programming environment in FIGURE 1. While the programming model of the present invention may be used in various environments, by way of non-limiting example, FIGURE 2 illustrates the use of the programming model within a user-interface environment. Thus, in this example, one of the applications in FIGURE 1 creates display 200 shown in FIGURE 2. The display 200 includes a dialog box 202. The dialog box 202 may include any number of other controls (i.e., objects). In this example, the dialog box 202 includes a list box 204, an edit box 206, and a container object 208 having two button objects (e.g., an OK button 210 and a CANCEL button 212). As will be discussed in further detail later in conjunction with FIGURE 3, using conventional programming techniques, each of these objects (e.g., 202 - 212) is a child object derived from a base object (e.g., an element class). In addition, in accordance with one embodiment of the present invention, each child object may also be derived from a node class. As will be described later, this node class provides the "attaching" feature for the attached properties.

While the appearance of display 200 may appear similar to displays created using prior programming models, the mechanism for creating the display 200 is quite different. In prior programming models, the software code implementing display 200 would have had both the behavior of the object and the appearance of the object incorporated within each object itself or in one of the parent objects. However, as will be described in detail below, in accordance with the present invention, separate classes provide the actions and the appearance for objects 202 - 212 without resorting to an explicit reference between the two classes.

This allows developers to modify either the action portion or the appearance portion without modifying the entire object. Thus, in this user-interface example, each object is factored into two parts. One part relates to the action (i.e., behavior) associated with the object, and, the other part relates to the appearance (i.e., rendering) of the object. Because the rendering is separated from the behavior

of the object, the appearance of the object may be easily modified to change the way the object appears.

While the above user-interface example conveniently factors the properties for the objects into a behavior group and an appearance group, the inventors have determined that properties of many objects in various environments can be conveniently divided into two or more relevant groups. Then, in accordance with the present invention, these relevant groups can be "implicitly" associated with each other to fully implement the object. This "implicit" association provides several advantages over prior programming models. One of the advantages is that third party developers can conveniently modify the relevant group that pertains to their application while keeping the other groups untouched. This greatly simplifies the amount of information that the developer needs to know in order to change the functionality of the object. In addition, the object hierarchy becomes more manageable. Another advantage is that storage is reduced because each object does not necessarily maintain a local value with a current state. Rather, as will be described in detail in conjunction with FIGURE 6, the state may be retrieved from various sources when needed.

FIGURE 3 is a graphical representation of a programming model that provides a mechanism for "implicitly" associating relevant groups of properties from one class with another class in accordance with the present invention. Similar to other programming models, the present invention supports a base class 302. The base class 302 includes a first set of properties 304 associated with base class 302. In addition, base class 302 includes a set of methods 306 and a set of events 308. However, in accordance with the present invention, the base class is derived from a node class 301. The node class 301 provides a SetValue() method 307 and a GetValue() method 309.

A second set of properties 324 is included within an attached class 322. The attached class 322 also includes at least two static methods associated with each of the second set of properties 324 (e.g., SetFont() 326 and GetFont() 330). These static methods may be thought of as global methods that are accessible to objects that have registered the property associated with the static method. The developer supplying the attached class 322 is responsible for writing the code for each of these static methods 326 and 330. The code will include a call to one of the

10 methods provided by the node class 301. For example, an application calling
SetFont() 326 for a specific identifier PropertyID and a given font Value, will
ultimately execute the call to SetValue() 307 in the node class 301. The parameters
that SetFont() 326 passes to the call to SetValue() 307 will include the specific
5 identifier PropertyID and the given font Value that were provided in the SetFont()
326 call. An exemplary SetFont() function may appear as follows:

```
10      SetFont(PropertyID,white)
      { PropertyID->SetValue(PropertyID,white);
      }.
```

One skilled in the art will appreciate that the attached class, thus,
provides strong typing for any attached property. While the present invention
supports the application calling the SetValue() 307 directly, this circumvents the
15 advantage of providing strong typing during compile time. As mentioned earlier,
strong typing allows errors to be detected before run-time operation. In another
embodiment, the attached class 322 provides an additional static method (e.g.,
GetFontID() 328) for each of the attached properties 324. This additional static
method is operative to test whether the requested attached property has been
20 registered. Thus, this additional static method 328 is used to insure that the
application has properly registered the attached property before attempting to
perform the set and get static methods on the attached property. For example, if
GetFontID() 328 detects that the Color attached property has not been registered,
GetFontID() may give an error, may automatically register the Color attached
25 property for the requesting object, or the like.

The following is an exemplary embodiment for providing the delayed
registration:

```
30      class FontProvider
      {   public static Font GetFont(Node n)
          {   return n.GetValue(GetFontID());   }

          public static void SetFont(Node n, Font newFont)
```

```

        {    n.SetValue(GetFontID(), newFont);    }

        public static DynamicProperty GetFontID()
        {    // Delay register Font property if necessary
5           if (FontID == null)
            {    FontID = RegisterProperty("Font", typeof(Font), "Arial", ...);
            }
        }

10        private static DynamicProperty FontID = null;
    }

```

Similar to other programming models, the present invention supports child classes (not shown) of base class 302. For simplicity, FIGURE 3 does not graphically illustrate any child classes of base class 302 or any other attached classes, and, therefore, does not illustrate child classes being associated with another attached class. However, one skilled in the art, after reading the following description, will be able to easily design an application that has child classes referencing other attached classes. In general, any class may be a "provider" of attached properties, as long as the class provides the necessary static methods for the attached properties.

The attached class 322 allows third party developers the ability to dynamically add properties and functionality to their applications without modifying the base class 302. Similar to modifying properties within base class 302, the added properties (e.g., second set of properties 324) affect objects within an entire hierarchical object tree 390. The attached class 322 is not instantiated during the execution of the application. Therefore, for each object instantiated from base class 302, the same attached class (attached class 322) may provide the second set of properties 324 to each of these objects.

During run-time of the application, a base object 312 becomes instantiated. In addition, one or more child objects (e.g., child objects 340 - 350) become instantiated. The instantiation of the objects in object tree 390 may be through programmatic control (e.g., C#), through markup language (e.g., XML), or

through other means. When the object tree 390 is created through programmatic control, a compiler typically identifies whether the instructions implementing the attached class 322 has any errors, such as illegal name or data type. As discussed above, the attached class 322 provides strong typing. Therefore, errors may be caught before run-time. In addition, software development tools may detect errors during development of the application. When object tree 390 is created through markup language, a parser interprets markup statements. In general, a name of a tag in the markup language is the name for the corresponding class.

Each of these child objects 340 - 350 may include one or more child properties (e.g., child property 341) within child object (e.g., child object 340). Values for child properties may be stored in the associated child object. However, in accordance with the present invention, the child objects do not necessarily have local storage for storing a value. In addition, each of these child objects 340 - 350 may include one or more attached properties (e.g., attached property 352 represented within dashed box) associated with the child object. As will be described in detail in conjunction with FIGURE 6, attached property 352 may obtain its value from local storage associated with child object 350, may obtain its value through inheritance (e.g., base object 312), through a property sheet (not shown), through programmatic methods (e.g., static method 330 in conjunction with method 309), and through other means in accordance with the present invention. While these values for the attached properties are associated with the child object, the storage for these values is not typically part of the child object.

Again, FIGURE 3 illustrates the programming model in a user-interface environment. Thus, one will note that the properties within base object 312 and child objects 340 -350 relate to the behavior of the objects. Base object 312 represents a dialog object, child object 340 represents a button object, child object 350 represents an edit box, and child object 342 represents a selector object having two child objects 346 and 348, representing a list box and a tree, respectively. Each of these child objects 340 - 350 includes one or more child properties (e.g., child property 341), such as a pressed property for button object and expanded property for tree object.

In contrast, the properties within attached class 322 relate to the appearance for the object. For example, the second set of properties may include a

property and default value for font type, color, and the like. This programming model is configured to set a property at run-time on the child objects. To that extent, the attached property (e.g., attached property 352) in each of the child objects is configured to receive the value from the attached class.

5 FIGURE 4 illustrates several exemplary syntaxes for various operating environments in which the programming model shown in FIGURE 3 may be implemented. The use of the term "Provider" within each of the exemplary syntaxes denotes that the static methods described in the syntaxes are associated with an attached class having a name of "Provider". Thus, because each of these
10 syntaxes allows the attached class to be named, properties that are named identically within two different attached classes will not conflict with each other. This allows developers to develop attached property classes without having to be wary of potential naming conflicts.

 Now turning to the exemplary syntaxes, if the programming
15 environment is a programmatic language, such as Visual Basic or C#, pseudo code may appear as shown in programmatic syntax 401. Programmatic syntax 401 includes a class identifier 402 that identifies the class as containing an attached property. Programmatic syntax 401 further includes a method identifier 404, a
20 parameter list having a first identifier 406 for specifying a name for a property and a second identifier 408 for specifying a value associated with the property specified in the first identifier 406. The parameter list is enclosed within parenthesis and a period separates the class identifier 402 and method identifier 404. Referring to the attached class 322 shown in FIGURE 3, for attached property "FONT", the programmatic syntax 401 is as follows:

25

Attached Class.SetFont(PropertyID, value).

 Interestingly, this syntax appears to look similar to syntaxes in previous programming models. However, the syntax is not identical and
30 programmatic syntax 401 is different than prior syntaxes. For example, in programmatic syntax 401, method identifier 404 identifies a static method on a Provider class (i.e., attached class 322 in FIGURE 3). In prior programming models, method identifier would have identified an instance method on a Provider

instance. Because the present invention utilizes the Provider class (i.e., the Attached Class 322 in FIGURE 3), the programming model can support extender properties in markup and property sheets because the lifetime of the Provider instance does not need to be tracked. Another feature of programmatic syntax 401 is that the syntax is strongly typed. As mentioned earlier, a strongly typed syntax enables good tool support because potential errors, such as misspelled names, incorrect data types, and the like, can be detected at compile time rather than at run-time.

In addition, for programming environments such as Visual Basic, C# or the like, the programming model also allows a loosely typed syntax (i.e., programmatic syntax 411). Programmatic syntax 411 includes an object identifier 412 identifying a target object, a method identifier 414 identifying a static method within an attached property class, a parameter list including an attached property identifier and a value 419. The attached property identifier includes a class identifier 416 and a property identifier 418 that identifies the attached property within the associated attached class.

In another embodiment, the programming model may be implemented using markup language syntax, such as markup syntax 421. Markup syntax 421 includes a tag 422, an attached property having a provider designator 424 and a property identifier 426, and a value 428. In one embodiment, the tag 422 is the name of the target object (e.g., button object 340 in FIGURE 3). Typically, markup syntax 421 begins and ends with an open and close symbol, "<" and ">", respectively. Again, a parser performs processing to locate where the attached properties exist because the attached properties are not directly on the class.

In yet another embodiment, the programming model may be implemented using extensible Markup Language (XML) with style sheets, such as style sheet syntax 431 shown in FIGURE 4. For this embodiment, the style sheet syntax 431 includes a "tag" attribute 432. The tag attribute 432 corresponds to the class that is requesting processing. One illustrative example for attached properties in XML is as follows:

30

```
<Button xmlns:ap="expandoNamespace" Pressed="false"
ap:Expandos.String="string"/>
```

When an application implementing attached properties in accordance with the present invention executes, the application instantiates the objects in the object tree. Then, the application, in conjunction with the property engine, handles the run-time operations, which includes retrieving and setting values for the objects.

5 FIGURES 5 and 6 illustrate processing performed by the application, in conjunction with the property engine.

However, before either the SetValue() process or the GetValue() process, shown in FIGURES 5 and 6, may be performed, the attached property that is specified within either of these two processes must be registered. As mentioned

10 above, one embodiment may perform a check to determine whether the property has been registered. This check (e.g., GetFontID() 328) may be performed within any of the static methods that correspond to the attached property (i.e., SetFont() 326 and GetFont() 330). Because this check must be preformed each time any of the static methods are called, this embodiment incurs a performance penalty. Those skilled in

15 the art will appreciated that other embodiments that optimize the registration of the property may be implemented without departing from the present invention, such as requiring the application to perform the registration.

The registration of the attached property returns a unique PropertyID for the attached property. One illustrative call for registering a property may be in

20 the form as follows:

```
BarDP = RegisterProperty("Bar",0 ,typeof(string),typeof(Button),0x3);.
```

In this embodiment, the BarDP variable will then contain the unique identifier for the attached property. This unique identifier is then used in the SetValue() and the

25 GetValue() function calls. In one embodiment, the RegisterProperty is called, directly or indirectly, from the owner's static constructor. The owner refers to the class that attaches the attached property. Once the attached property is registered, any instance that matches the target type for the attached property may "implicitly" attach the property by calling the Get and Set static methods associated with the

30 attached property.

As shown above in the exemplary call for registering a property, "Bar" designates a name for the attached property. The property engine typically does not use this name. However, a parser uses this name to determine whether a

property with the same name as already been registered to the owner. The target type is Button. For attached properties, the owner and the target will not be the same. The default value is "0". As will be described in detail in conjunction with FIGURE 6, the property engine determines where to retrieve a value for the attached property. Therefore, the registerProperty call includes behavior bits that identify stages in which a value for the property is searched, such as inheritance, property sheets. In the above exemplary call, the behavior bits "0x3" indicates that a local value, a property sheet, and inheritance are searched in order to determine the value for the attached property. Thus, in one embodiment, the interpretation of these behavior bits may be coded within the GetValue() static method. Alternatively, the determination of these stages may be provided by an expression that models relationships between properties in a formal manner.

In one embodiment, the code for an attached class may take the form shown in Table 1.

```

class BarProvider: Object {
    public static DynamicProperty BarDP = RegisterProperty(
        "Bar", 0 , typeof(string), typeof(Button));

    public static string GetBar(Button button) {
        return (string) button.GetValue(BarDP); }
    public static void SetBar(Button button, string value) {
        button.SetValue(BarDP, value); }
}.

```

Table 1.

In the above code, the attached property named "Bar" is defined within attached class "BarProvider". The attached property name "Bar" is defined with a data type as string and can only be attached to Button objects. The static methods are strongly typed and allow access to the property and setting the property.

Once the attached property has been registered, the SetValue() and GetValue() functions may be performed. FIGURE 5 is a logical flow diagram

illustrating a process for setting a value in accordance with the present invention. Processing begins at block **501**, where the set function has been initiated to change a value associated with some property. In general, the SetValue process **500** will store the value locally for a property of interest (hereinafter referred to as the interested property). Processing continues at decision block **502**.

At decision block **502**, a determination is made whether local storage exists for the interested property. Because the present invention does not automatically have storage for each property for each instance of an object, if the interested property has not previously had a value stored for it, storage is allocated (block **504**). Once storage has been allocated, processing continues to block **506**, as it would have if it had detected local storage at decision block **502**.

At block **506**, the value is copied into the storage associated with the interested property. Once the interested property has changed states, notifications are sent of the change in state (block **508**). In one embodiment, this notification may involve setting a dirty bit to indicate to dependent properties that their state may not be valid anymore. Alternatively, the notification may involve reporting to each dependent that a source has changed as described in the above-mentioned patent application. Processing continues at decision block **510**.

At decision block **510**, a determination is made whether a value was stored in the cache for the interested property. If there is not a value in the cache for this interested property, the process ends. However, if there is a value in the cache, the cache is cleared (block **512**) because this value is no longer valid. The process then end.

FIGURE 6 is a logical flow diagram illustrating a process for retrieving a value in accordance with the present invention. Processing begins at block **601**, where the query has been initiated. Processing continues at block **602**.

At block **602**, the application indicates that one of the properties of interest (hereinafter, referred to as interested property) needs updating. Typically, this may occur when an object is querying properties for their current state. The process **600** determines a value for the property by checking various stages. These stages are defined when the property is registered. Decision blocks **606-610** represent exemplary stages, but other stages may be added without departing from the present invention. Processing continues at decision block **604**.

At decision block 604, the process checks the cache to determine whether the interested property has been previously cached. Typically, interested properties are cached in order to optimize retrieval. If the property has been previously cached, the process continues at block 612, where the value is retrieved from the cache. Alternatively, the process continues at decision block 606. At decision block 606, a determination is made whether the value for the interested property is local. The value will be local if a SetValue() has been performed to set a value locally for the property, as described in FIGURE 5. If the value is local, the process continues at block 612, where the value is retrieved from local storage. If the value is not local, the process continues at decision block 608.

At decision block 608, a determination is made whether the value for the interested property is available in a property sheet. When a base class has an associated property sheet, the property sheet specifies how each of the children of the base class will be rendered once instantiated. If the interesting property is available in a property sheet, processing continues at block 612, where the value is retrieved from the property sheet. However, if the value is not in the property sheet, processing continues at decision block 610.

At decision block 610, a determination is made whether the value for the interested property may be obtained through inheritance (i.e., an inherited value). Inherited values are derived from a parent object of the child object. If one of the parent objects has the interesting property, processing proceeds to block 612, where the value is retrieved from the parent object. However, if the value is not inherited, a default value from the attached class associated with the attached property is retrieved (block 614). Processing continues at block 616.

At block 616, the application calculates a weight metric based on the stage at which the value was received. The weight metric is stored and is used to make educated decisions on which interested properties to cache in order to optimize future retrievals. Processing then ends.

In addition, to the SetValue and GetValue processes described above, the programming model of the present invention provides enhanced functions, such as group queries or group notification. Thus, as described, the present invention provides a programming model that provides a dependency-based property system, which supports complex relationships between properties through the use of

attached classes. Applications built using this programming model scale very well because the property management technique does not require excessive local, per-instance based storage. Instead, the programming model promotes reuse of property values, such as through the use of property sheets and inheritance at the base class level. As described above, the present invention provides a property management mechanism whereby objects in the object hierarchy store property values through attaching.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

24

WE CLAIM:

1. A computer-readable medium including an object having a property in a first set of properties, the computer-readable medium further comprising:
a data structure including definitions for each of a second set of
5 properties and including at least one static method associated with one property out of the second set of properties, the at least one static method having a first parameter that uniquely identifies the one property, the static method being operative to associate the one property with the object without specifying an explicit reference to the one property in the object.
- 10 2. The computer-readable medium of Claim 1, wherein the static method supports a strongly typed syntax.
3. The computer-readable medium of Claim 1, wherein the static method being operative to associate the one property with the object includes
15 retrieving a value for the object without having the value stored locally on the object.
4. The computer-readable medium of Claim 3, wherein retrieving the value comprises determining which one of a plurality of stages holds the value associated with the one property.
5. The computer-readable medium of Claim 4, wherein one stage
20 includes a parent object of the object.
6. The computer-readable medium of Claim 4, wherein one stage includes a property sheet.
7. The computer-readable medium of Claim 1, wherein the static method further includes a second parameter, the second parameter corresponding to
25 a given value for the one property, and wherein the static method being operative to associate the one property with the object comprises setting the given value in a local storage for the object.

8. The computer-readable medium of Claim 1, wherein the first set of properties corresponds to behaviors of the object, and the second set of properties corresponds to an appearance of the object.

5 9. A computer-readable medium having computer-executable components, comprising:
a base class including a plurality of properties; and
an attached class including a plurality of attached properties, each of the attached properties capable of being associated with an instance of an object derived from the base class, the attached class further including a static method for
10 associating one of the plurality of attached properties with the instance of the object.

10. The computer-readable medium of Claim 9, wherein the object derived from the base class inherits at least one method from a node class, the one method being called by the static method when associating the one attached property with the instance of the object.

15 11. The computer-readable medium of Claim 10, wherein the static method and the one method each include a first parameter for passing a unique identifier for the one attached property that is being associated with the instance of the object.

20 12. The computer-readable medium of Claim 11, wherein the unique identifier is available after registering the one attached property at run-time.

13. The computer-readable medium of Claim 10, wherein the one method supports a loosely typed syntax.

14. The computer-readable medium of Claim 9, wherein the static method supports a strongly typed syntax.

25 15. The computer-readable medium of Claim 9, wherein associating the one attached property comprises retrieving a value for the object from a parent object of the object.

16. The computer-readable medium of Claim 9, wherein associating the one attached property comprises retrieving a value for the object from a property sheet.

17. A computer system that uses objects, comprising:
5 a first object derived from a first class, the first object including a first set of properties;
a second class providing a second set of properties to the first object, the second class including a static method operative to attach one of the second set of properties to the first object in response to an operation associated with the first
10 object.

18. The computer system of Claim 17, wherein the static method includes a first parameter for uniquely identifying which one property out of the second set of properties the operation is requesting.

19. The computer system of Claim 18, wherein the one property is
15 registered before the first parameter can uniquely identify the one property.

20. A computer-implemented method for setting a state for a property, comprising:

declaring a first object having a first set of properties ;
registering one or more dynamic properties that are associated with
20 the first object, the dynamic properties providing a second set of properties to the first object, wherein storage for the second set of properties is not within the first object.

21. The computer-implemented method of Claim 20, wherein registering the one or more dynamic properties includes assigning a unique identifier to each of
25 the one or more dynamic properties.

22. The computer-implemented method of Claim 21, further comprising calling a static method having a first parameter, the first parameter referring to the unique identifier.

23. The computer-implemented method of Claim 22, wherein the static method is strongly typed.

24. The computer-implemented method of Claim 20, wherein the first object has one or more associated child objects, each child object thereby being
5 associated with the second set of properties.

25. A computer-readable medium having at least one computer executable instruction, comprising:

an instruction means associating an attached property with an instance of an object derived from another class through a static method in an
10 attached property class.

26. The computer-readable medium of Claim 25, wherein the instruction means supports a strongly typed syntax for a programmatic language.

27. The computer-readable medium of Claim 25, wherein the instruction means supports loosely-type syntax for a programming language.

15 28. The computer-readable medium of Claim 25, wherein the instruction means comprises a statement in a markup document.

29. The computer-readable medium of Claim 25, wherein the instruction means comprises a statement in a property sheet.

20 30. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:

a class identifier identifying an attached property class;

a method identifier identifying a method in the attached property class for affecting an attached property associated with a target object;

a parameter list, comprising:

25 a property identifier identifying the attached property.

28

31. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:

an object identifier identifying a target object;

a method identifier identifying a method in an attached property class

5 for affecting an attached property associated with the target object;

a parameter list, comprising:

an attached property identifier including a class identifier

identifying the attached property class and a property identifier identifying a name

for the attached property within the attached property class.

10 32. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:

a tag representing a target object;

an attached property identifier having a provider designator

identifying an attached property class and a property identifier identifying a name

15 for the attached property within the attached property class.

33. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:

a XML schema for defining a target object, an attached property

class, and an attached property within the attached property class, in such a manner

20 that the attached property becomes associated with the target object without an explicit reference to the attached property in the target object.

Abstract

Described is a mechanism for allowing new functionality for an object to be expressed as a property that is not built into the class from which the object derives. More specifically, the mechanism associates properties in one class with another class. A computer-readable medium, that includes an object having a property in a first set of properties, further includes a data structure. The data structure includes definitions for each of a second set of properties and includes at least one static method. The static method is associated with one property out of the second set of properties and includes a first parameter. The first parameter uniquely identifies the one property. The static method is operative to associate the one property with the object without specifying an explicit reference to the one property in the object. The property is registered during run-time in order to receive the unique identifier.

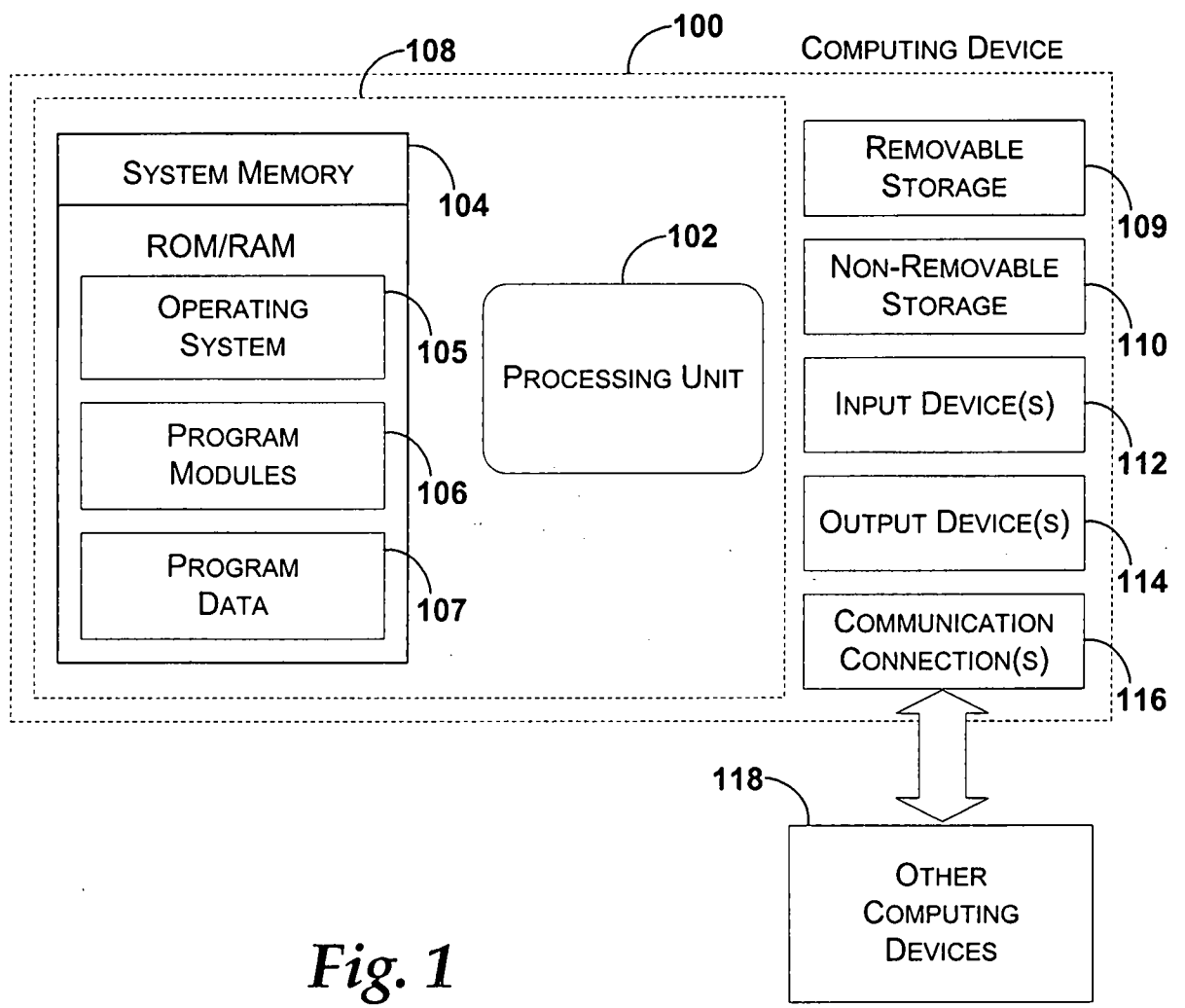


Fig. 1

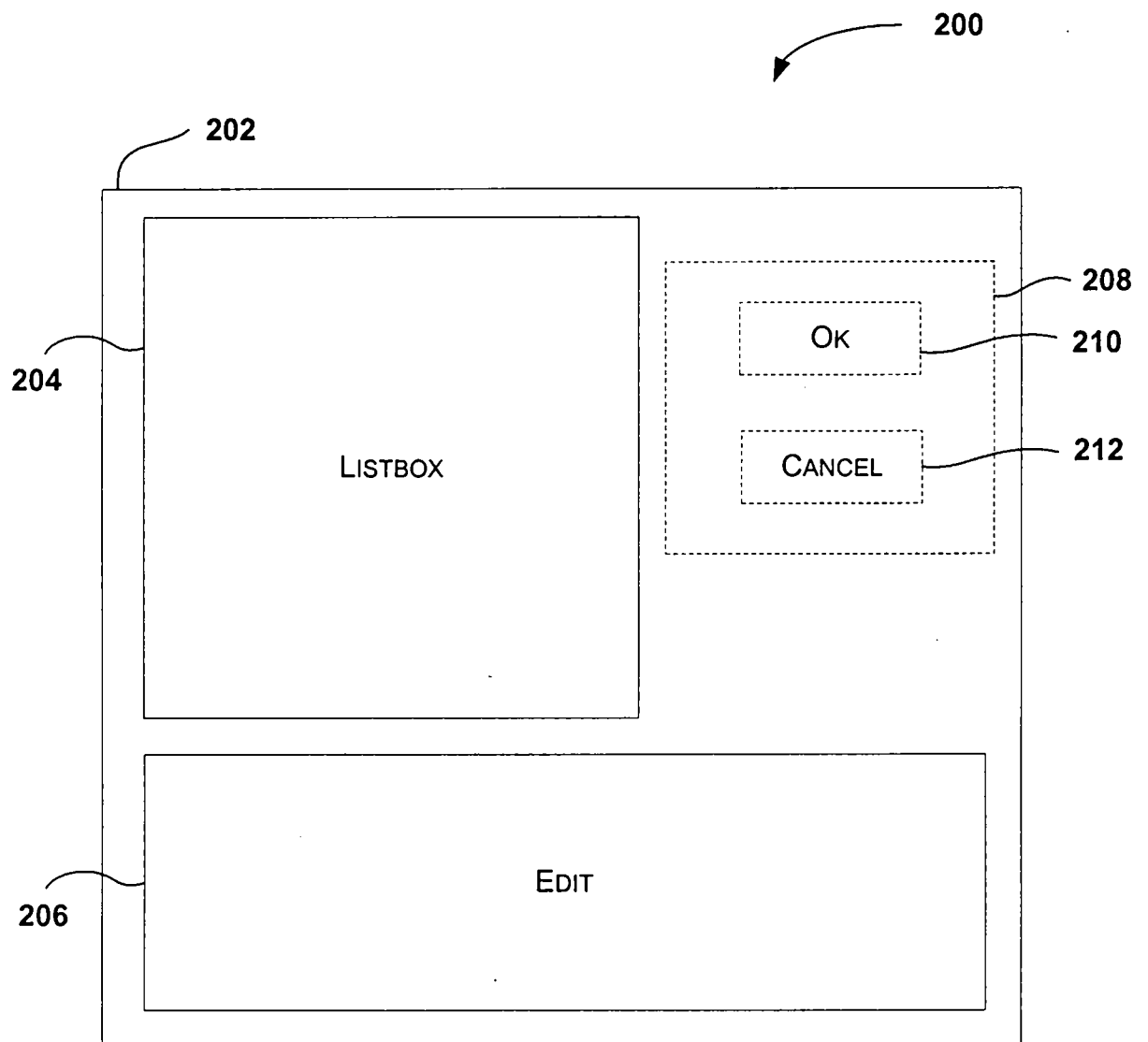


Fig. 2

3/6

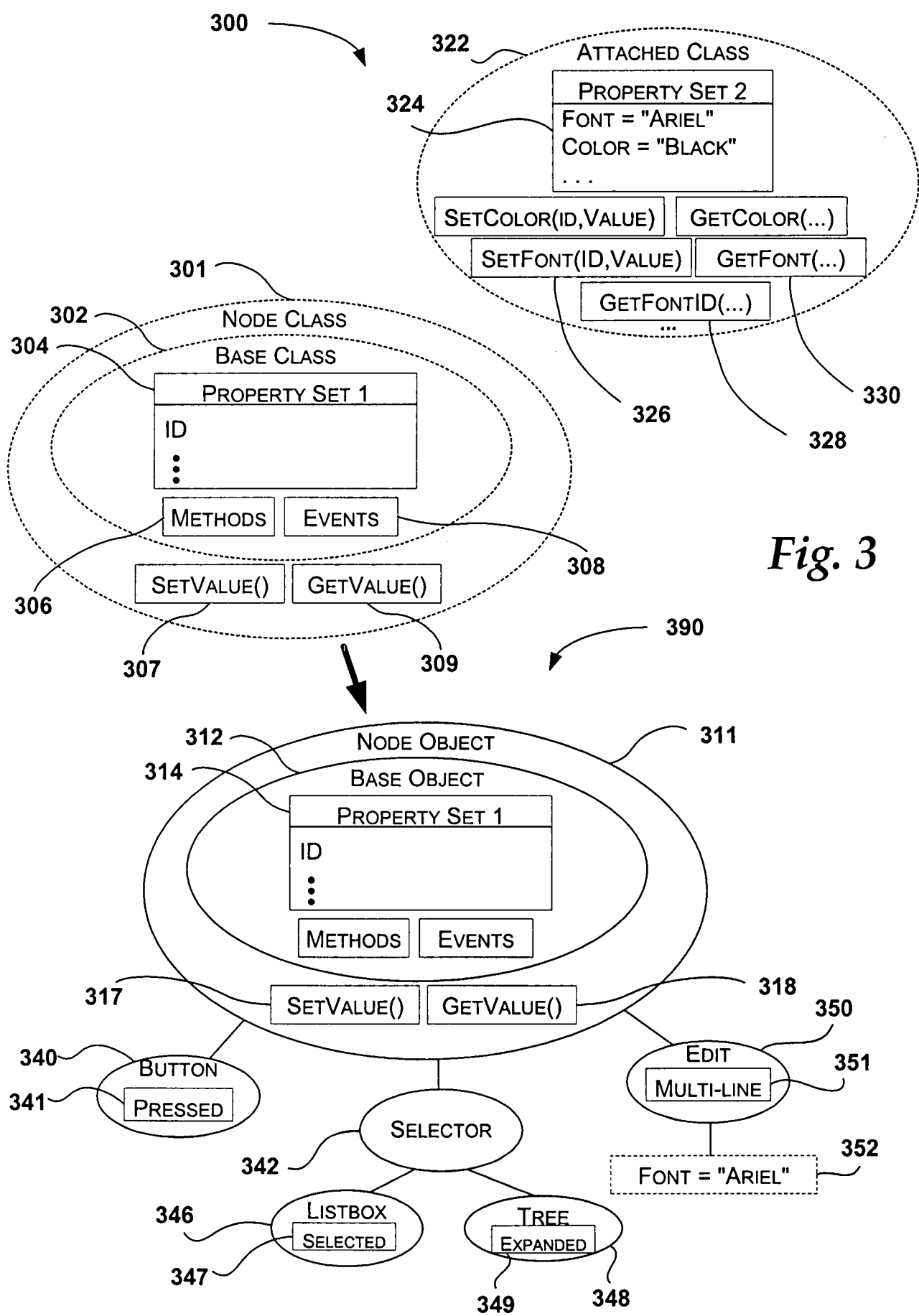


Fig. 3

4/6

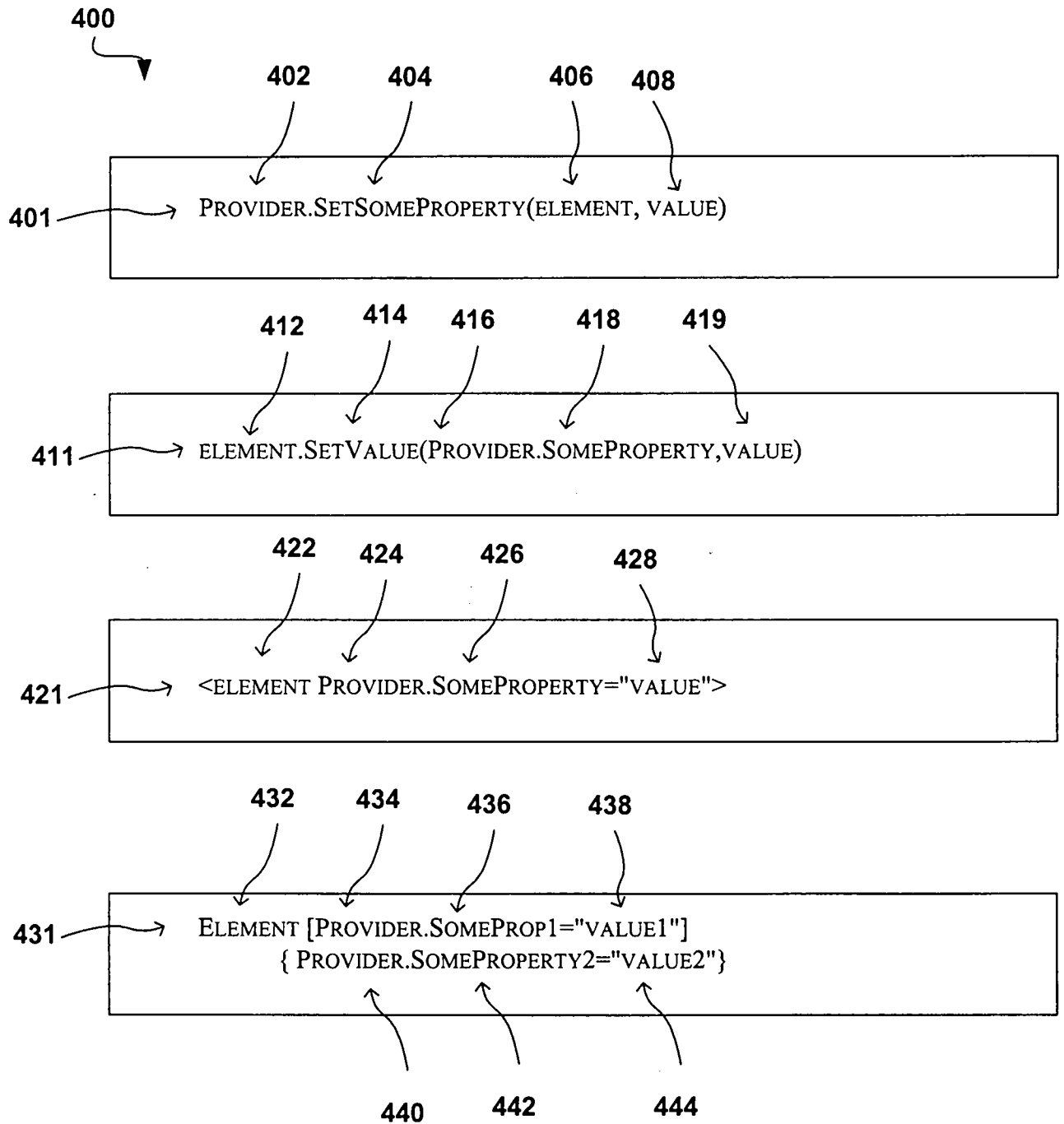


Fig. 4

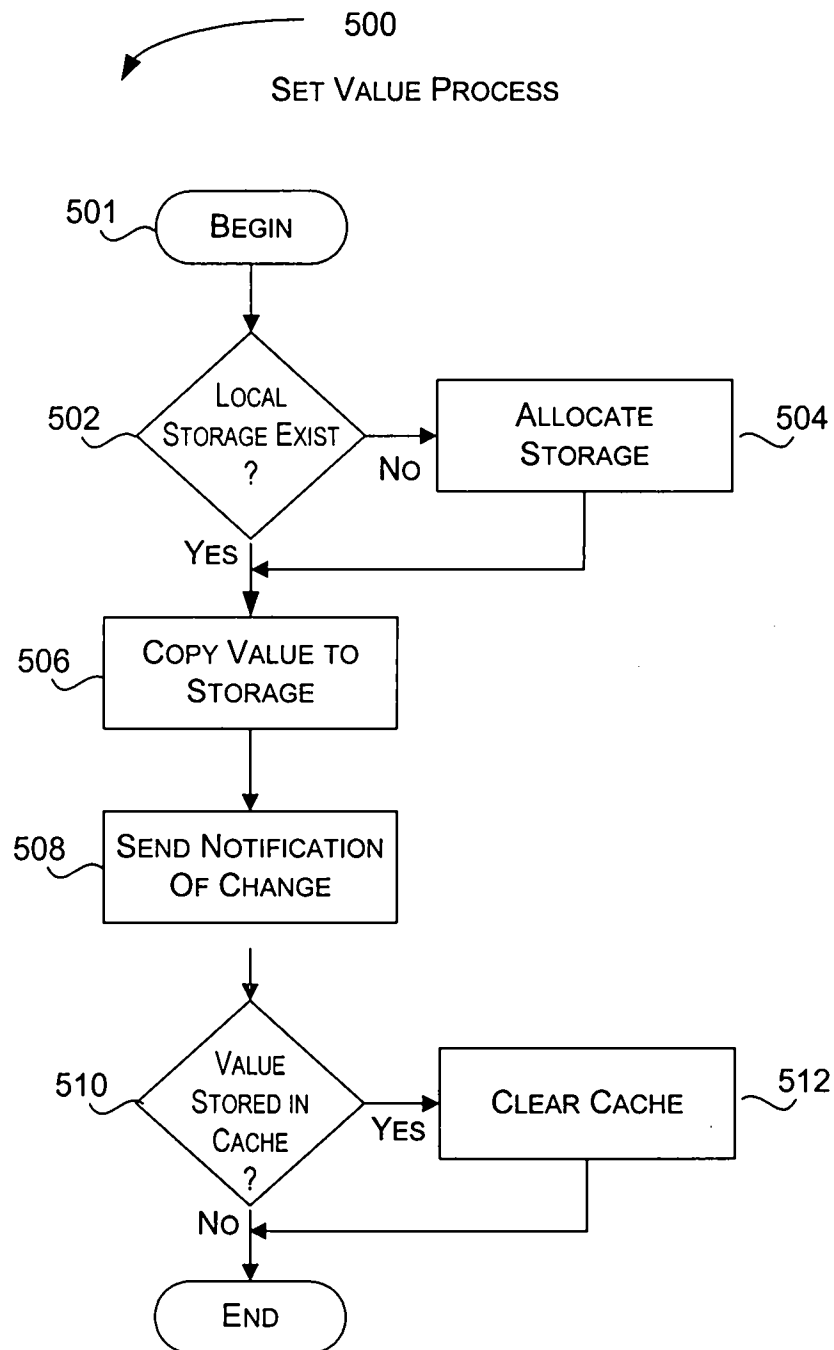


Fig. 5

6/6

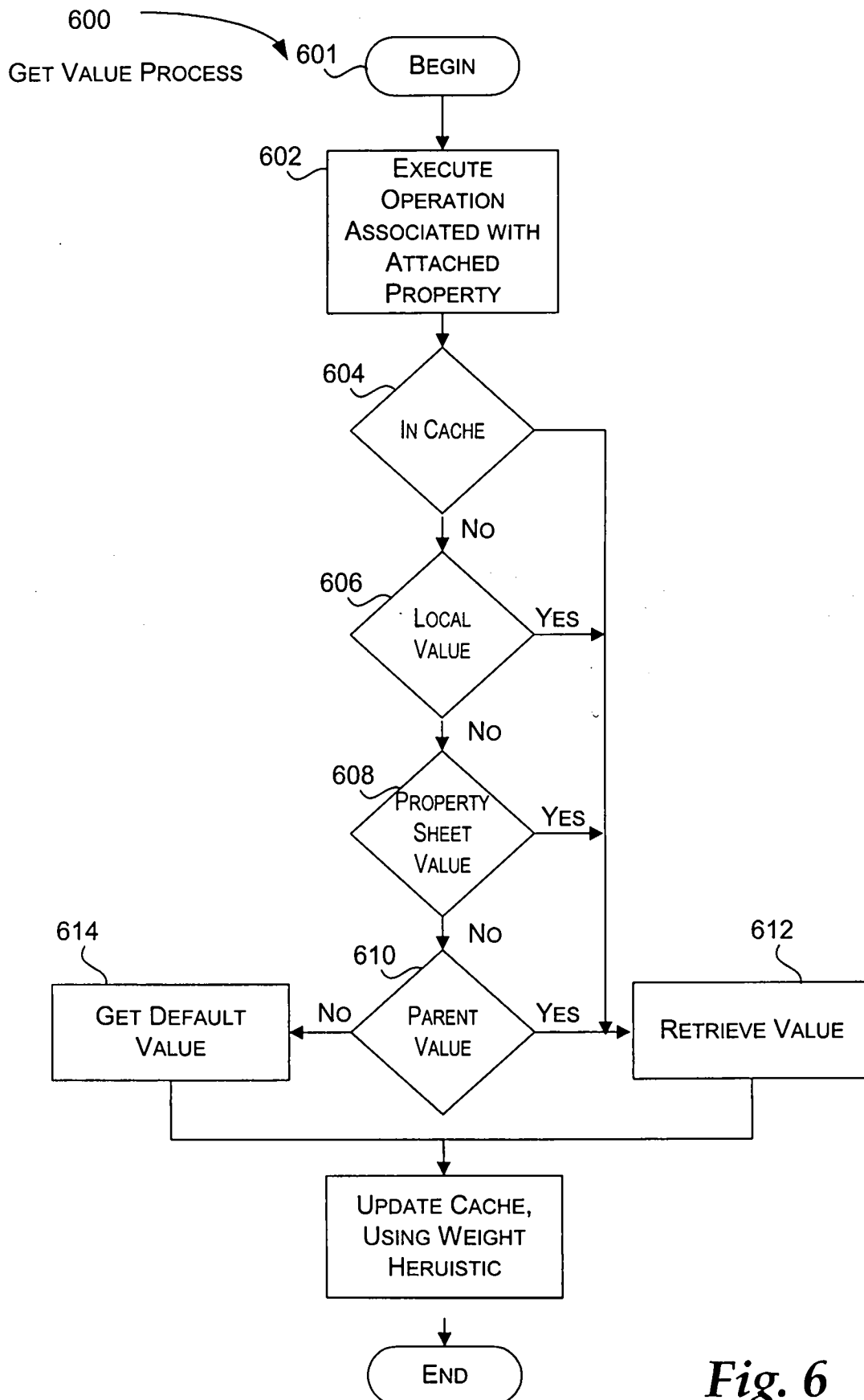


Fig. 6

SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

CAMPO DE LA INVENCIÓN

La presente invención se relaciona de manera general con aplicaciones de soporte lógico, y más particularmente, con mecanismos para manejar propiedades de objetos dentro de una aplicación de soporte lógico.

ANTECEDENTES DE LA INVENCIÓN

La mayoría de los modelos de programación actuales soportan el concepto de clases. Estas clases son típicamente estructuradas en un árbol jerárquico que tiene ramas que representan diferentes clases en la clase jerárquica. Cuando dos ramas están en diferentes niveles,

la rama inferior representa la clase hija. La clase hija hereda la información de las clases asociadas con la rama superior (es decir la clase padre). Cuando dos ramas están en el mismo nivel, las clases se denominan como clases hermanas. Para el propósito de la siguiente discusión, cuando se hace referencia a una clase hija en relación a una clase padre, los términos, clase inferior y clase superior, se pueden utilizar para referirse a la clase hija y a la clase padre, respectivamente, la rama más superior en la jerarquía representa una clase base en el árbol de la clase jerárquica. Típicamente, la información en cada clase incluye propiedades, métodos, y eventos. Las propiedades describen características asociadas con la clase. Por ejemplo, una clase botón puede tener propiedades tales como amplitud, color de trasfondo, tipo de letra, visible, y presionada. Cuando

una de las clases se inicia, se crea un objeto de esa clase. Cada propiedad en el objeto tiene un valor asociado, que puede ser requerido y establecido durante una operación del tiempo de ejecución. El valor que es requerido o establecido se puede esperar para que se adecue a un tipo de datos específico si la sintaxis es fuertemente digitada. Teniendo sintaxis que es fuertemente digitada es deseable entonces que los errores se puedan detectar dentro de la aplicación del software antes de la operación del tiempo de ejecución.

Una vez la clase jerárquica está en su lugar, agregar nueva funcionalidad a objetos dentro de la clase jerárquica representa un problema. En un modelo de programación, la nueva funcionalidad es forzada dentro de la clase base. Cuando se hace esto, la clase base se hace

muy grande (por ejemplo 100 métodos, 50 propiedades y 20 eventos), que dan como resultado una jerarquía de objeto casi inmanejable. Un resultado indeseable de este modelo de programación es que el número de propiedades, métodos, y eventos dentro de la clase base se hace opresiva para que los desarrolladores la entiendan completamente antes de implementar las características deseadas en los objetos que crean ellos. Aún otro resultado indeseable de este modelo de programación es que los requisitos de memoria se hacen enormes debido al hecho de que los valores para las propiedades son almacenados localmente. En razón a que los valores son almacenados localmente, las aplicaciones creadas dentro de este modelo de programación no se escalan bien.

Este modelo de programación también origina otros problemas a terceras partes desarrolladoras. Las terceras partes desarrolladoras que desean agregar nuevas funcionalidades deben agregar una clase hija al fondo de la jerarquía. En razón a que la nueva clase hija está en el fondo de la jerarquía, la nueva funcionalidad no está disponible a otras clases dentro de la jerarquía. Por lo tanto, las terceras partes desarrolladoras podrían necesitar agregar nueva funcionalidad a varias nuevas clases hija. Como uno puede imaginar, esto da como resultado una duplicación de código, que impacta la mantenibilidad de la jerarquía de objeto. Por las anteriores razones, el modelo de programación no es muy deseable.

Hasta la presente invención, un modelo de programación que permitía que se suministrara nueva funcionalidad a una clase dentro de una jerarquía de clase existente sin los inconvenientes anteriores ha eludido aquellos expertos en la técnica.

RESUMEN DE LA INVENCION

La presente invención suministra un mecanismo que permite que se suministre nueva funcionalidad a una clase sin tener que volverse la nueva funcionalidad parte permanente de la clase. Además, la presente invención permite que la nueva funcionalidad sea expresada como una propiedad que no se construye dentro de la clase. En general, la presente invención suministra un mecanismo para asociar propiedades en una clase con otra clase.

Este asocio es fácilmente modificable de tal forma que otros conjuntos de propiedades se pueden asociar con la clase.

¿de qué?

En una modalidad, un medio legible por computadora que incluye un objeto que tiene una propiedad en el primer conjunto de propiedades, incluye además una estructura de datos. La estructura de datos incluye definiciones para cada una de un segundo conjunto de propiedades e incluye al menos un método estadístico. El método estadístico se asocia con una propiedad además del segundo conjunto de propiedades e incluye un primer parámetro. El primer parámetro identifica únicamente una propiedad. El método estadístico es operativo para asociar una propiedad con el objeto sin especificar una referencia explícita a una propiedad en el objeto.

En un aspecto de la invención, el método estadístico soporta una sintaxis fuertemente digitada.

En otro aspecto de la invención, el método estadístico incluye retornar un valor para el objeto sin tener el valor almacenado localmente sobre el objeto. El valor se puede regresar desde varias etapas, tales como un objeto padre o una hoja de propiedad.

Una ventaja de la presente invención es que la división de las propiedades en uno o más subconjuntos le permite a cada subconjunto de propiedades ser más fácilmente mantenido. Otra ventaja de la presente invención es que la jerarquía del objeto es más extensible y le permite a los desarrolladores agregar

funcionalidad a la jerarquía de objeto que impacta la clase base y todas las clases inferiores.

Otra ventaja de la presente invención es que el manejo del almacenamiento para las propiedades en los objetos se hace más eficiente y conveniente.

Aún otra ventaja de la presente invención es que el modelo de programación opera dentro de un ambiente programático o de marcadores. Además, el modelo de programación opera en lenguajes de programación fuertemente digitados, tales como C++ y #. Adicionalmente, el modelo de programación mantiene hojas de propiedad, cambia notificaciones y herencia de valor.

Otra ventaja de la presente invención es que las bibliotecas independientes pueden coexistir en razón a que los conflictos de nombre son menos probables porque el nombre de la clase anexa se hace efectivamente parte del nombre propiedad. Por lo tanto, si dos diferentes desarrolladores quedan cada uno una propiedad anexa que tenga una propiedad denominada "color", las dos propiedades anexas no entrarán en conflicto.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

LA FIGURA 1 ilustra un ejemplo de un dispositivo de cómputo que se puede utilizar en una modalidad de ejemplo de la presente invención.

11

46

LA FIGURA 2 es una pantalla de ejemplo que se puede crear con el dispositivo de cómputo de la figura 1.

LA FIGURA 3 es una representación gráfica de un modelo de programación que le permite a las propiedades de una clase ser anexadas a otra clase de acuerdo con la presente invención.

LA FIGURA 4 ilustra varias sintaxis de ejemplo para implementar el modelo de programación mostrado en la figura 3.

LA FIGURA 5 es un diagrama de flujo lógico que ilustra un proceso para establecer un valor de acuerdo con la presente invención.

1

LA FIGURA 6 es un diagrama de flujo lógico que ilustra un proceso para recuperar un valor de acuerdo con la presente invención.

DESCRIPCIÓN DETALLADA DE LA MODALIDAD PREFERIDA

De manera resumida, la presente invención suministra un modelo de programación que permite que una nueva funcionalidad se suministre a una clase sin tener, la nueva funcionalidad, que hacer parte permanente de la clase. Además, la presente invención le permite a la nueva funcionalidad ser expresada como una propiedad no se construye dentro de la clase. En general, la presente invención suministra un mecanismo para asociar propiedades en una clase con otra clase. Como será evidente después de la lectura de la descripción

detallada que sigue, el modelo de programación de la presente invención suministra propiedades dinámicas a objetos sin construir las propiedades dentro del objeto.

Con referencia a la figura 1, un sistema de ejemplo para implementar la invención incluye un dispositivo de cómputo, tal como un dispositivo de cómputo 100. En una configuración muy básica, el dispositivo de cómputo 100 típicamente incluye al menos una unidad de procesamiento 102 y una memoria de sistema 104. dependiendo de la configuración exacta y del tipo de dispositivo de cómputo, la memoria del sistema 104 puede ser volátil (tal como la RAM), no volátil (tal como la ROM), memoria flash, etc.) o alguna combinación de las dos. La memoria del sistema 104 incluye típicamente un sistema operativo 105, uno o más módulos de programa 106, y puede incluir

datos de programas 107. Ejemplos de módulos de programa 106 incluyen el Visual Studio IntelliSense de Microsoft Corporation de Redmond, WA, y otro software de ambientes de programación, que utilizan bibliotecas objeto. Además, los módulos de programa 106 incluyen aplicaciones de soporte lógico creadas utilizando un ambiente de programación de soporte lógico. Cuando estas aplicaciones de soporte lógico se ejecutan sobre una unidad de procesamiento 102, un motor de propiedad procesa la aplicación del soporte lógico de acuerdo con el modelo de programación de la presente invención. El motor de propiedad puede ser parte del sistema operativo 105 o puede ser otro módulo programable 106. esta configuración básica del dispositivo de cómputo 100 se ilustra en la figura 1 por aquellos componentes con línea punteada 108.

El dispositivo de cómputo 100 puede tener características o funcionalidades adicionales. Por ejemplo, el dispositivo de cómputo 100 también pueden incluir dispositivos de almacenamiento de datos adicionales (removibles y/o no removibles) tales como, por ejemplo, discos magnéticos, discos ópticos, o cintas. Tal almacenamiento adicional se ilustra en la figura 1 mediante almacenamiento removible 109, y el almacenamiento no removible 110. los medios de almacenamiento de computadora pueden incluir medios volátiles y no volátiles, removibles y no removibles implementados en cualquier método o tecnología para almacenar la información, tal como instrucciones legibles por computadora, estructuras de datos, módulos de programa, u otros datos. En la memoria del sistema 104, y

el almacenamiento no removible 110 son todos ejemplos de medios de almacenamiento en computadora.

Medios de almacenamiento en computadora incluye, pero no están limitados a, memoria RAM, ROM, EEPROM, FLASH u otra tecnología de memoria, CR-ROM, discos versátiles digitales (DVD) u otros dispositivos de almacenamiento óptico, cartuchos magnéticos, cintas magnéticas, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, o cualquier otro medio que se pueda utilizar para almacenar la información deseada y que se pueda acceder mediante el dispositivo de cómputo 100. Cualquiera de tales medios de almacenamiento en computadora puede ser parte del dispositivo 100. El dispositivo de cómputo 100 también puede tener dispositivo o dispositivos de entrada 112

tales como teclados, ratones, lápices, dispositivos de entrada de voz, dispositivos de entrada por toque. El o los dispositivos de salida 114 tales como la pantalla, los parlantes, la impresora, etc. también se pueden incluir. Estos dispositivos son bien conocidos en la técnica y no necesitan ser discutidos en extensión aquí.

El dispositivo de cómputo 10° también puede contener conexiones de comunicación 116 que le permiten al dispositivo comunicarse con otros dispositivos de cómputo 118, tales como sobre una red. Las conexiones de comunicación 116 son un ejemplo de medios de comunicación son un ejemplo de medios de comunicación. Los medios de comunicación típicamente se pueden ejemplificar por medio de instrucciones legibles en computadora, estructuras de datos, módulos de programa, u otros datos en señales de

datos modulados, tales como una onda portadora u otro mecanismo de transporte, e incluye cualquier medio de suministro de información. El término "señal modulada de datos" significa una señal que tiene una o más de sus características establecida o cambiada de tal manera que codifica la información en la señal. Por vía de ejemplo, y no de limitación, los medios de comunicación incluyen medios cableados tales como redes cableadas conexión cableada directa, y medios inalámbricos tales como medios acústicos, RF, infrarrojos y otros medios inalámbricos. El término "medio legible por computadora" como se utiliza aquí incluye tanto medios de almacenamiento como medios de comunicación.

La figura 2 es una pantalla de ejemplo que se puede crear mediante una aplicación dentro del ambiente de

programación en la figura 1. aunque el modelo de programación de la presente invención se puede utilizar en varios ambientes, por vía de ejemplo no limitante, la figura 2 ilustra el uso del modelo de programación dentro de un ambiente de interfase de usuario. De esta manera, en este ejemplo, una de las aplicaciones en la figura 1 crea la pantalla 200 mostrada en la figura 2. La pantalla 200 incluye una caja de diálogo 202. La caja de diálogo 202 puede incluir cualquier número de otros controles (es decir objetos). En este ejemplo, la caja de diálogo 202 incluye una caja de lista 204, una caja de editar 206, y un objeto de contenedor 208 que tiene dos objetos de botón (por ejemplo un botón de OK 210 y un botón de cancelar 212). Como se discutirá con detalle posteriormente en conjunto con la figura 3, utilizando técnicas de programación convencional, cada uno de estos

objetos (por ejemplo 202-212) es un objeto hijo derivado de un objeto base (por ejemplo una clase de elemento). Adicionalmente, de acuerdo con una modalidad de la presente invención, cada objeto hijo también se puede derivar de una clase nodo. Como se describirá posteriormente, esta clase nodo suministra la característica de "anexión" para las propiedades anexadas.

Aunque la apariencia de la pantalla 200 puede parecer similar a las pantallas creadas utilizando modelos de programación anteriores, el mecanismo para crear la pantalla 200 es muy diferente. En modelos de programación anterior, la pantalla de implementación del código de software 20 habría tenido tanto el comportamiento del objeto como la apariencia del objeto

incorporado dentro de cada objeto mismo o en uno de los objetos padre. Sin embargo, como se describirá en detalle adelante, de acuerdo con la presente invención, las clases separadas suministran las acciones y la apariencia para los objetos 202-212 sin recurrir a una referencia explícita entre las dos clases.

Esto le permite a los desarrolladores modificar la porción de la acción o la porción de la apariencia sin modificar el objeto completo. De esta forma, en este ejemplo de interfase de usuario, cada objeto es fabricado en dos partes. Una parte relaciona la acción (es decir el comportamiento) asociado con el objeto, y, la otra parte relaciona la apariencia, (es decir la ejecución) del objeto. En razón a que la ejecución se separa del comportamiento del objeto, la apariencia del objeto puede

ser fácilmente modificada para cambiar la forma en que aparece el objeto.

Aunque el ejemplo anterior de interfase de usuario factoriza convenientemente las propiedades de los objetos en un grupo de comportamiento y en un grupo de apariencia, los inventores han determinado que las propiedades de muchos objetos en varios ambientes se pueden dividir convenientemente en dos o más grupos relevantes. Entonces, de acuerdo con la presente invención, estos grupos relevantes pueden ser "implícitamente" asociados uno con el otro para implementar completamente el objeto. Esta asociación "implícita" suministra varias ventajas sobre los modelos de programación anterior. Una de las ventajas es que las terceras partes desarrolladoras pueden modificar

convenientemente el grupo relevante que pertenece a su aplicación mientras mantienen los otros grupos sin ser tocados. Esto simplifica de manera notoria la cantidad de información que los desarrolladores necesitan conocer con el fin de cambiar la funcionalidad del objeto. Adicionalmente, la jerarquía del objeto se hace más manejable. Otra ventaja es que el almacenamiento se reduce en razón a que cada objeto no mantiene necesariamente un valor local con un estado corriente. Al contrario, como se describirá con detalle en relación con la figura 6, el estado puede ser recuperado de varias fuentes cuando se requiera.

La figura 3 es una representación gráfica de un modelo de programación 300 que suministra un mecanismo para asociar "implícitamente" grupos relevantes de

propiedades de una clase con otra clase de acuerdo con la presente invención. De manera similar a otros modelos de programación, la presente invención sustenta una clase base 302. la clase base 302 incluye un primer conjunto de propiedades 304 asociado con la clase base 302. adicionalmente, la clase base 302 incluye un conjunto de métodos 306 y un conjunto de eventos 308. sin embargo, de acuerdo con la presente invención, la clase base se deriva de una clase nodo 301. La clase nodo 301 suministra un método de SetValue() 307 y un método GetValue() 309.

Un segundo conjunto de propiedades 324 se incluye dentro de una clase anexa 322. la clase anexa 322 también incluye al menos dos métodos estáticos asociados con cada uno de los segundos conjuntos de propiedades 324

(SetFont() 326 y GetFont() 330). Estos métodos estáticos se pueden enseñar como métodos globales que son accesibles a objetos que han registrado la propiedad asociada con el método estático. El desarrollador que suministra la clase anexa 322 es responsable por escribir el código para cada uno de estos métodos estáticos 326 y 330. El código incluirá un llamado a uno de los métodos suministrados por la clase nodo 301. por ejemplo, un llamado de aplicación SetoFont() 326 para un identificador específico PropertyID y un "Value" de tipo de letra dado, ejecutarán finalmente el llamado al SetValue() 307 en la clase nodo 301. los parámetros que el SetFont() 326 pasa al llamado al SetValue()307 incluirán el identificador específico PropertyID y "Value" del tipo de letra dado que se suministró en el

llamado del `SetFont()` 326. Una función de ejemplo del `SetFont()` puede aparecer como sigue:

```
SetFont(PropertyID,white)
{
    PropertyID->SetValue(PropertyID,White);
}.
```

Un experto en la técnica apreciará que la clase anexa, así, suministra un fuerte digitado para cualquier propiedad anexada. Aunque la presente invención sustenta el llamado de aplicación al `SetValue()` 307 directamente, esto enreda la ventaja de suministrar digitado fuerte durante el tiempo de compilación. Como se mencionó anteriormente, los digitados fuertes permiten que los errores sean detectados antes de la operación del tiempo

de ejecución. En otra modalidad, la clase anexada 322 suministra un método estático adicional (por ejemplo GetFontID() 328) para cada una de las propiedades anexadas 324. Este método estático adicional es operativo para ensayar si la propiedad anexa solicitada se ha registrado. De esta forma, este método estático adicional 328 se utiliza para asegurar que la aplicación haya registrado adecuadamente la propiedad anexada antes de intentar desarrollar el conjunto y conseguir los métodos estáticos sobre la propiedad anexada. Por ejemplo, si el GetFontID() 328 detecta que la propiedad de color anexada no se ha registrado, el GtFontID() puede dar un error, puede registrar automáticamente la propiedad anexada de Color para el objeto que se solicita, o similar.

El siguiente se una modalidad de ejemplo para suministrar el registrar retrasado:

```
Class FontProvider

{   public static Font GetFont(Node n)

    {   return n.GetValue(GetFontID());   }

    {   n.SetValue(),newFont);   }

Public static DynamicProperty GetFontID()

{   //Delay register Font property if
necessary if(FontID=Hill)

    {   FontID = registerProperty("Font",typeof(Font),
;Arial",...);

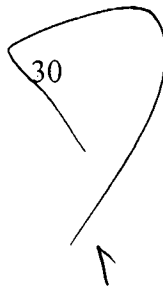
        }   FontID = RegisterProperty("Font",
typeof(Font),"Arial",...);

    }
```

}

```
private static DependencyProperty FontID = null;  
}
```

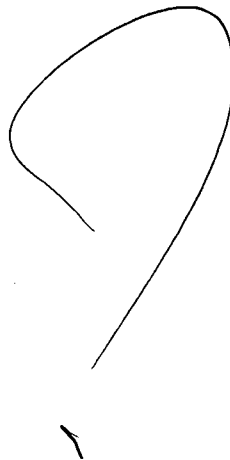
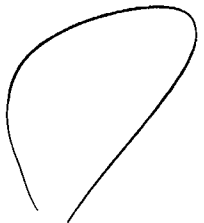
Similares a otros modelos de programación, la presente invención sustenta clases hija (no mostrada) de la clase base 302. Por simplicidad, la figura 3 no ilustra gráficamente ninguna clase hija de la clase base 302 o cualquier otra clase anexada, y, por lo tanto, no ilustra las clases hija que están asociadas con otras clases anexadas. Sin embargo, un experto en la técnica, después de leer la siguiente descripción, será capaz de diseñar fácilmente una aplicación que tenga clases hijas



65

que referencien otras clases anexadas. En general, cualquier clase puede ser un "suministrador" de propiedades anexadas, en tanto que la clase suministre los métodos estáticos necesarios para las propiedades anexadas.

La clase anexada 322 le permite a terceras partes desarrolladoras la capacidad de agregar dinámicamente propiedades y funcionalidad a sus aflicciones sin modificar la clase base 302. de manera similar a la modificación de las propiedades dentro de la clase base 302, las propiedades agregadas (por ejemplo el segundo conjunto de propiedades 324) afectan los objetos dentro de un árbol completo de objeto jerárquico 390. La clase anexada 322 no le es solicitada instancia de la clase base 302, la misma clase anexada (clase anexada 322)



puede suministrar el segundo conjunto de propiedades 324 a cada uno de estos objetos.

Durante el tiempo de ejecución de la aplicación, el objeto base 312 se le solicita instancia. Además, uno o más objetos hijo (por ejemplo los objetos hijo 340-350) se les solicita instancia. La solicitud de instancia de los objetos en el árbol de objeto 390 puede ser a través de control programático (por ejemplo C#), a través del lenguaje de marcadores (por ejemplo XML), o a través de otros medios. Cuando el árbol de objeto 390 se crea a través de un control programático, un compilador típicamente identifica si las instrucciones que implementan la clase anexada 322 tienen cualquier tipo de errores, tales como nombre ilegal o tipo de datos. Como se discutió anteriormente, la clase anexada 322

suministra una digitación fuerte. Por lo tanto, se pueden detectar los errores antes del tiempo de ejecución. Además, las herramientas de desarrollo de soporte lógico pueden detectar los errores durante el desarrollo de la aplicación. Cuando el árbol objeto 390 se crea a través del lenguaje de marcadores, un analizador interpreta los anuncios de marcadores. En general, un nombre de una etiqueta en un lenguaje de marcadores es el nombre de la clase correspondiente.

Cada uno de estos objetos hijo 340-350 puede incluir una o más propiedades hijo (por ejemplo la propiedad hijo 341) dentro del objeto hijo (por ejemplo objeto hijo 340). Los valores para las propiedades hijo se pueden almacenar en el objeto hijo asociado. Sin embargo, de acuerdo con la presente invención, los objetos hijo no

necesariamente tienen almacenamiento local para almacenar un valor. Además, cada uno de estos objetos hijo 340-350 pueden incluir una o más propiedades anexadas (por ejemplo la propiedad anexada 352 representada dentro de la caja punteada) asociada con el objeto hijo. Como se describirá con detalle en conjunto con la figura 6, la propiedad anexada 352 puede obtener un valor del almacenamiento local asociado con el objeto hijo 350, puede obtener su valor a través de la herencia (por ejemplo el objeto base 312), a través de una hoja de propiedad (no mostrada), a través de métodos programáticos (por ejemplo método estático 330, en conjunto con el método 309), y a través de otros medios de acuerdo con la presente invención. Aunque estos valores para las propiedades anexadas están asociados con

el objeto hijo, el almacenamiento para estos valores no es típicamente parte del objeto hijo.

De nuevo, la figura 3 ilustra el modelo de programación en un ambiente de interfase de usuario. De esta forma, se notará que las propiedades dentro del objeto base 312 y los objetos hijo 340-350 se relacionan con el comportamiento de los objetos. El objeto base 312 representa un objeto de diálogo, el objeto hijo 340 representa un objeto de botón, el objeto hijo 350 representa una caja de edición, y el objeto hijo 342 representa un objeto selector que tiene dos objetos hijo 346 y 348, que representan una caja de lista y un árbol, respectivamente. Cada uno de estos objetos hijo 340-350 incluyen una o más propiedades hijo (por ejemplo la propiedad hijo 341), tal como una propiedad presionada

por un botón objeto y la propiedad expandida para el objeto árbol.

Como contraste, las propiedades dentro de la clase anexada 322 se relacionan con la apariencia del objeto. Por ejemplo, el segundo conjunto de propiedades puede incluir una propiedad y un valor por defecto para el tipo de letra, color, y similar. Este modelo de programación se configura para establecer una propiedad al momento al tiempo de de la ejecución sobre los objetos hijo. En ese alcance, la propiedad anexada (por ejemplo la propiedad anexada 352) en cada uno de los objetos hijo se configura para recibir el valor de la clase anexada.

La figura 4 ilustra varias sintaxis de ejemplo para varios ambientes operativos en los cuales el modelo de

programación mostrado en la figura 3 se puede implementar. El uso del término "suministrador" dentro de cada una de las sintaxis de ejemplo denota que los métodos estáticos descritos en la sintaxis se asocian con la clase anexada que tiene un nombre de "suministrador". De esta forma, en razón a que cada una de estas sintaxis le permite a la clase anexada ser nombrada, las propiedades que son nombradas idénticamente dentro de dos diferentes clases anexadas no entrarán en conflicto una con la otra. Esto le permite a los desarrolladores desarrollar las clases de propiedad anexada sin tener que ser prudente de los potenciales conflictos del hombre.

Regresando ahora a la sintaxis de ejemplo, si el ambiente de programación es un lenguaje programático, tal como el VisualBasic o el C# puede aparecer un pseudocódigo

como se muestra en la sintaxis programática 401. la sintaxis programática 401 incluye un identificador de clase 402 que identifica la clase como conteniendo una propiedad anexada. La sintaxis programática 401 incluye además un identificador de método 404, una lista de parámetro que tiene un primer identificador 406 para especificar un nombre para una propiedad y un segundo identificador 408 para especificar un valor asociado con la propiedad especificada en el primer identificador 406. la lista de parámetro está incluida dentro del paréntesis y un punto separa el identificador de clase 402 y el identificador de método 404. en relación a la clase anexada 322 mostrada en la figura 3, para la propiedad anexada "FONT", la sintaxis programática 401 es como sigue:

Attached Class.SetFont(PopertyID, value).

De manera interesante, la sintaxis parece lucir de manera similar a la sintaxis en los modelos de programación previos. Sin embargo, la sintaxis no es idéntica y la sintaxis programática 401 es diferente de la sintaxis anterior. Por ejemplo, en la sintaxis programática 401, el identificador de método 404 identifica un método estático sobre una clase suministradora (es decir, la clase anexada 322 en la figura 3). En modelos de programación anterior, el identificador de método habría identificado un método de instancia sobre un suministrador de instancia. Debido a que la presente invención utiliza la clase de proveedor (ie, la clase adjunta 322 en la figura 3), el modelo de programación puede soportar propiedades extendedoras en

la marcación y las hojas de propiedades debido al tiempo de vida del momento de suministrador que no necesita ser rastreado. Otra característica de la sintaxis programática 401 es que la sintaxis es fuertemente escrita a máquina. Como se mencionó anteriormente, una sintaxis escrita fuertemente permite un soporte de buenas herramientas debido a que los errores potenciales, tal como nombres mal escritos, tipos de datos incorrectos y similares, pueden ser detectados en el tiempo de compilación en vez de en el tiempo de corrida.

Adicionalmente, los ambientes de programación tales como Visual Basic, C# o similares, el modelo de programación también permite una sintaxis de escritura suelta (por ejemplo sintaxis programática 411). La sintaxis programática 411 incluye un identificador de

objeto 412 que identifica un objeto objetivo, un identificador de método 414 que identifica un método estático dentro de una clase de propiedades adjunta, una lista de parámetros que incluye un identificador de propiedades adjunto y un valor 419. el identificador de propiedades adjunto incluye un identificador de clase 416 y un identificador de propiedades 418 que identifica las propiedades adjuntas dentro de la clase adjunta asociada.

En otra modalidad, el modelo de programación puede ser implementado usando sintaxis de lenguaje de marcación, tal como la sintaxis de marcación 421. La sintaxis de marcación 421 incluye una bandera 422, una propiedad adjunta que tiene un designador de suministrador 424 y un identificador de propiedades 426, y un valor 428. En una modalidad, la bandera 422 es el

nombre del objeto objetivo (por ejemplo, el objeto botón 340 en la figura 3). Típicamente, la sintaxis de marcación 421 inicia y finaliza con un símbolo abierto y cerrado "<" y ">", respectivamente. Nuevamente, un analizador lleva a cabo el procesamiento para localizar en donde las propiedades adjuntas existentes debido a que las propiedades adjuntas no están directamente sobre la clase.

En todavía otra modalidad, el modelo de programación puede ser implementado usando un lenguaje de marcación extensible (XML) con hojas de estilo, tales como sintaxis de hoja de estilo 431, mostrada en la figura 4. Para esta modalidad, la sintaxis de hoja de estilo 431 incluye un atributo de "bandera" 432. El atributo de bandera 432 corresponde a la clase que requiere el procesamiento. Un

ejemplo ilustrativo para las propiedades adjuntas en XML es como sigue:

```
<Button xmlns:ap="expandoNameSpace"Pressed="false"  
ap:Expandos.String="string"/>
```

Cuando una aplicación que implementa las propiedades adjuntas de acuerdo con la presente invención se ejecuta, la aplicación muestra inmediatamente los objetos en el árbol de objeto. Entonces, la aplicación, en conjunto con el motor de propiedades, maneja las operaciones de tiempo de corrida, lo que incluye recuperar y establecer los valores para los objetos. Las figuras 5 y 6 ilustran el procesamiento llevado a cabo por la aplicación, en conjunto con el motor de propiedades.

Sin embargo, antes de que cada proceso SetValue() o tGetValue(), mostrado en la figura 5 y 6, sean llevado a cabo, la propiedad adjunta que es especificada dentro de cualquiera de estos dos procesos debe ser registrada. Como se mencionó anteriormente, una modalidad llevara cabo una verificación para determinar si la propiedad ha sido registrada. Esta verificación (por ejemplo, GetFontID() 328) puede ser llevada a cabo dentro de cualquiera de los métodos estáticos que corresponden a la propiedad adjunta (por ejemplo SetFont() 326 y GetFont() 330). Debido a que esta verificación debe ser llevada a cabo cada vez que cualquiera de los métodos estáticos sea llamado, esta modalidad incurre en una falta de comportamiento. Aquellos versados en la técnica podrán apreciar que otras modalidades que optimizan el registro de la propiedad pueden ser implementadas sin salirse de

la presente invención, tal como requerir que la aplicación lleve a cabo el registro.

El registro de la propiedad adjunta regresa un PropertyID único para la propiedad adjunta. Una llamada ilustrativa para registrar una propiedad puede estar en la forma que sigue:

BarDP =

```
RegisterProperty("Bar",0,typeof(string),typeof(Button),0x3);.
```

En esta modalidad, la variable BarDP podrá contener el identificador único para la propiedad adjunta. Este identificador único es entonces usado en las funciones SetValue() y GetValue() de llamadas. En una modalidad, la

propiedad de registro es llamada, directa o indirectamente desde el constructor estático del propietario. El Propietario se refiere a la clase que adjunta la propiedad adjunta. Una vez que la propiedad adjunta es registrada, cualquier momento que case con el tipo de objetivo para la propiedad adjunta puede adjuntar "implícitamente" la propiedad llamando los métodos estáticos Get y Set asociados con la propiedad adjunta.

Como se mostró anteriormente en la llamada ilustrativa para el registro de una propiedad, "Bar" indica o designa un nombre para la propiedad adjunta. El motor de propiedades típicamente no utiliza este nombre. Sin embargo, un analizador usa este nombre para determinar si una propiedad con el mismo nombre ya ha sido registrada por el propietario. El tipo de objetivo

es botón. Para las propiedades adjuntas, el propietario y el objetivo no será el mismo. El valor por defecto es "0". Como será descrito en detalle en conjunto con la figura 6, el motor de propiedades determina si se recupera un valor para la propiedad adjunta. Por lo tanto, la llamada RegisterProperty incluye bits de comportamiento que identifican etapas en las cuales un valor para la propiedad se busca, tal como herencia, hojas de propiedades. En la llamada ilustrativa anterior, los bits de comportamiento "0x3" indica que un valor local, una hoja de propiedades, y de herencia son buscadas para poder determinar el valor para la propiedad adjunta. Así, en una modalidad, la interpretación de estos bits de comportamiento pueden ser codificadas dentro del método estático GetValue (). Alternativamente, la determinación de estas etapas será provista por una

expresión que modela las relaciones entre las propiedades de una manera formal.

En una modalidad, el código para la clase adjunta puede tomar la forma mostrada en la tabla 1.

```
Class BarProvider: Object {  
    public static DependencyProperty BarDP =  
Registerproperty("Bar", 0, typeof(string),  
typeof(Button)):
```

```
    public static string GetBar(Button buton){  
        return (string) button.GetValue(BarDP);}  
    public static void SetBar(Button Button, string  
value){  
        button.SetValue(BarDP, value);}
```

}.
.

Tabla 1.

En el anterior código, la propiedad adjunta llamada "Bar" está definida dentro de la clase adjunta "BarProvider". La Propiedad adjunta llamada "Bar" está definida como un tipo de dato como un hilo y puede solamente estar ligada a los objetos Button. Los métodos estáticos son fuertemente escritos y permiten el acceso a las propiedades y al establecimiento de las propiedades.

Una vez que la propiedad adjunta ha sido registrada, el SetValue() y el GetValue() como funciones pueden ser llevados a cabo. La figura 5 es un diagrama de flujo lógico que ilustra un proceso para establecer un valor de

acuerdo con la presente invención. El procesamiento se inicia en el bloque 501, en donde la función establece ha sido iniciada para cambiar un valor asociado con alguna propiedad. En general, el proceso SetValue 500 almacenará localmente el valor para una propiedad de interés (de aquí en adelante se refiere a la propiedad interesada). El procesamiento continúa con el bloque de decisión 502.

En el bloque de decisión 502, se hace una determinación de si el almacenamiento local existe para la propiedad interesada. Debido a que la presente invención no tiene de manera automática almacenamiento para cada propiedad en cada instante de un objeto, si la propiedad interesada no se le ha dado previamente un valor almacenado para esta, el almacenamiento es asignado (bloque 504). Una vez que el almacenamiento ha sido

asignado, el procesamiento continúa en el bloque 506, como si se hubiera detectado un almacenamiento local en el bloque de decisión 502.

El bloque 506, el valor es copiado dentro del almacenamiento asociado con la propiedad interesada. Una vez que la propiedad interesada ha cambiado de sus estados, envían notificaciones del cambio en el estado (bloque 508). En una modalidad, esta notificación puede involucrar establecer un bit sucio para indicar a las propiedades dependientes que su estado puede no ser válido ahora. Alternativamente, la notificación puede involucrar reportar a cada dependiente que una fuente ha cambiado como se describió en la solicitud de patente anteriormente mencionada. El procesamiento continúa en el bloque de decisión 510.

En el bloque de decisión 510, se hace una determinación de si el valor fue almacenado en el caché para la propiedad interesada. Si no hay un valor en el caché para esta propiedad interesada, el proceso finaliza. Sin embargo, si hay un valor en el caché, el caché es limpiado (bloque 512) debido a que este valor no es ya válido. El proceso entonces finaliza.

La figura 6 es un diagrama de flujo lógico que ilustra un proceso para recuperar un valor de acuerdo con la presente invención. El proceso se inicia en el bloque 601, en donde se inicia una pregunta. El proceso continúa en el bloque 602.

En el bloque 602, la aplicación indica que una de las propiedades de interés (de aquí en adelante será referida como propiedad interesada) requiere ser actualizada. Típicamente, esto puede ocurrir cuando un objeto está buscando propiedades para su estado actual. El proceso 600 determina un valor para las propiedades por medio de verificar diversas etapas. Estas etapas son definidas cuando la propiedad es registrada. Los bloques de decisión 606 a 610 representan etapas ilustrativas, pero otras etapas pueden ser agregadas sin salirse de la presente invención. El proceso continúa en el bloque de decisión 604.

En el bloque de decisión 604, el proceso verifica el caché para determinar si la propiedad interesada ha sido guardada en el caché anteriormente. Típicamente, las

propiedades interesadas son guardadas en el caché para poder optimizar su recuperación. Si la propiedad ha sido guardada anteriormente en el caché, el proceso continúa en el bloque 612, en donde el valor es recuperado del caché. Alternativamente, el proceso continúa en el bloque de decisión 606. en el bloque de decisión 606, se hace una determinación de si el valor para la propiedad interesada es local. El valor será local si un SetValue() ha sido llevado a cabo para establecer un valor localmente para la propiedad, como se describió en la figura 5. Si el valor es local, el proceso continúa en el bloque 612, en donde el valor es recuperado del almacenamiento local. Si el valor no es local, el proceso continúa en el bloque de decisión 608.

En el bloque de decisión 608, se hace una determinación de si el valor para la propiedad interesada está disponible en una hoja de propiedades. Cuando una clase de base tiene una hoja de propiedades asociada, la hoja de propiedades especifica cómo cada uno de los hijos de la clase de base se convertirá una vez que se inicie. Si las propiedades interesadas están disponibles en una hoja de propiedades, el proceso continúa en el bloque 612, en donde el valor es recuperado de la hoja de propiedades. Sin embargo, si el valor no está en la hoja de propiedades, el proceso continúa en el bloque de decisión 610.

El bloque de decisión 610, se hace una determinación de si el valor para la propiedad interesada puede ser obtenido de una herencia (por ejemplo un valor heredado).

Los valores heredados son derivados de un objeto pariente del objeto hijo. Si uno de los objetos parientes tiene una propiedad interesada, el proceso continúa en el bloque 612, en donde el valor es recuperado del objeto pariente. Sin embargo, si el valor no es heredado, un valor por defecto de la clase asociada con la propiedad adjunta es recuperada (bloque 614). El proceso continúa en el bloque 616.

En el bloque 616, la aplicación calcula un peso métrico basado en la etapa en la cual el valor fue recibido. El peso métrico es almacenado y es usado para efectuar cuales propiedades interesadas para poder optimizar recuperaciones futuras. El proceso entonces finaliza.

Adicionalmente, para los procesos SetValue y GetValue descritos anteriormente, el modelo de programación de la presente invención suministra funciones mejoradas tales como grupos de requisiciones o grupos de notificaciones. Así, como se describió, la presente invención suministra un modelo de programación que suministra un sistema de propiedades con base en la dependencia, que soporta relaciones complejas entre las propiedades por medio del uso de clases adjuntas. Las aplicaciones construidas usando este modelo de programación se dimensionan muy bien debido a que la técnica de administración de propiedades no requiere un almacenamiento con base en perinstancia local excesiva. En su lugar, el modelo de programación promueve la reutilización de los valores de propiedades, de modo que a través del uso de hojas de propiedades y herencia al

nivel de clase de base. Como se describió anteriormente, la presente invención suministra un mecanismo de administración de propiedades en donde los objetos en la jerarquía de objetos almacena los valores de propiedades por medio de adjuntarlos.

La anterior especificación, ejemplos y datos suministran una descripción completa de la fabricación y uso de la composición de la invención. Ya que muchas modalidades de la invención pueden ser hechas sin salirse del espíritu y alcance la invención, la invención reside en las reivindicaciones adjuntas aquí.

REIVINDICACIONES

1. Un medio legible por computador que incluye un objeto que tiene una propiedad de un primer conjunto de propiedades, el medio legible por computador comprende adicionalmente:

Una estructura de datos que incluye definiciones para cada uno de un segundo conjunto de propiedades e incluye, por lo menos, un método estático asociado con una propiedad de sacada del segundo conjunto de propiedades, por lo menos un método estático tiene un primer parámetro que identifica de manera única aquella propiedad, el método estático es operativo

para asociar esa una propiedad con el objeto sin especificar una referencia explícita a esa propiedad en el objeto.

2. El medio legible por computador de la reivindicación 1, en donde el método estático soporta una sintaxis de escritura fuerte.

3. El método legible por computador de la reivindicación 1, en donde el método estático es operativo de manera que asocia a aquella una propiedad con el objeto que incluye recuperar un valor para el objeto son tener el valor almacenado localmente en el objeto.

4. El medio legible por computador de la reivindicación 3, en donde recuperar el valor comprende de terminar cual entre una pluralidad de etapas tiene el valor asociado con esa una propiedad.
5. El medio legible por computador de la reivindicación 4, en donde una etapa incluye un objeto pariente del objeto.
6. El medio legible por computador de la reivindicación 4, en donde una etapa incluye una hoja de propiedades.

7. El medio legible por computador de la reivindicación 1, en donde el método estático comprende incluye adicionalmente el segundo parámetro, el segundo parámetro corresponde a un valor dado para aquella propiedad, y en donde el método estático es operativo para asociar dicha una propiedad con el objeto que comprende establecer el valor dado en un almacenamiento local para el objeto.

8. El medio legible por computador de la reivindicación 1, en donde el primer conjunto de propiedades corresponde a comportamientos del objeto, y el segundo

conjunto de propiedades corresponde a una
apariencia del objeto.

9. Un medio legible por computador que tiene
componentes ejecutables por computador,
comprende:

Una clase de base que incluye una pluralidad de
propiedades; y

Una clase adjunta que incluye una pluralidad de
propiedades adjuntas, cada una de las propiedades
adjuntas es capaz de estar asociada con una instante
de un objeto derivado de la clase de base, la clase
adjunta incluye adicionalmente un método estático

para asociar una entre la pluralidad de propiedades adjuntas con el instante del objeto.

10. El medio legible por computador de la reivindicación 9, en donde el objeto derivado de la clase de base hereda en por lo menos un método de una clase de nodo, el un método es llamado por el método estático cuando se asocia con una propiedad adjunta con la instancia del objeto.

11. El medio legible por computador de la reivindicación 10, en donde el método estático y el un método cada uno incluye un primer parámetro para pasar un identificador

único para la una propiedad adjunta que es asociada con el instante del objeto.

12. El medio legible por computador de la reivindicación 11, en donde el identificador único está disponible después de registrar aquella una propiedad adjunta en el tiempo de corrido.

13. El medio legible por computador de la reivindicación 10, en donde el un método soporta una sintaxis de tipo de escritura suelta.

14. El medio legible por computador de la reivindicación 9, en donde el método

estático soporta una sintaxis de escritura fuerte.

15. El medio legible por computador de la reivindicación 9, en donde asociar aquella una propiedad adjunta comprende recuperar un valor para el objeto de un objeto pariente del objeto.

16. El medio legible por computador de la reivindicación 9, en donde asociar aquella una propiedad adjunta comprende recuperar un valor para el objeto de una hoja de propiedades.

17. Un sistema de computador que utiliza
objetos, comprende:

Un primer objeto derivado de una primera clase, el primer objeto incluye un primer conjunto de propiedades;

Una segunda clase que suministra un segundo conjunto de propiedades al primer objeto, la segunda clase incluye un método estático operativo para adjuntar una entre el segundo conjunto de propiedades al primer objeto en respuesta a una operación asociada con el primer objeto.

18. El sistema de computador de la reivindicación 17, en donde el método

estático incluye un primer parámetro para identificar de manera única con una propiedad extraída del segundo conjunto de propiedades la operación es requerida.

19. El sistema de computador de la reivindicación 18, en donde aquella una propiedad es registrada antes de que el primer parámetro pueda identificar de manera única aquella una propiedad.

20. Un método implementado por computador para establecer un estado para una propiedad, comprende:

Declarar un primer objeto que tiene un primer conjunto de propiedades;

Registrar una o más propiedades dinámicas que están asociadas con el primer objeto, las propiedades dinámicas suministran un segundo conjunto de propiedades al primer objeto, en donde el almacenamiento para el segundo conjunto de propiedades no está dentro del primer objeto.

21. El método implementado por computador de la reivindicación (20, en donde registrar la una o más propiedades dinámicas incluye asignar un identificador único a cada una de la una o más propiedades dinámicas.

22. El método implementado por computador de la reivindicación 21, que comprende adicionalmente llamar un método estático que tiene un primer parámetro, el primer parámetro se refiere al identificador único.

23. El método implementado por computador de la reivindicación 22, en donde el método estático es escrito fuertemente.

24. El método implementado por computador de la reivindicación 20, en donde el primer objeto tiene uno o más objetos hijo, cada objeto hijo está por lo tanto asociado con el segundo conjunto de propiedades.

25. El medio legible por computador que tiene por lo menos una instrucción ejecutable por computador, que comprende:

Un medio de instrucción asociado con una propiedad adjunta con una instancia de un objeto derivado de otra clase por medio de un método estático en una clase de propiedades adjunta.

26. El medio legible por computador de la reivindicación 25, en donde el medio de instrucción soporta una sintaxis de escritura fuerte para un lenguaje programático.

27. El medio legible por computador de la reivindicación 25, en donde el medio de instrucción soporta una sintaxis tipo libre suelta para un lenguaje de programación.
28. El medio legible por computador de la reivindicación 25, en donde el medio de instrucción comprende una afirmación en un documento de marcación.
29. El medio legible por computador de la reivindicación 25, en donde el medio de instrucción comprende una formación en una hoja de propiedades.

30. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción-comprende:

Un identificador de clase que identifica una clase de propiedades adjunta;

Un identificador de método que identifica un método en la clase de propiedades adjuntas para afectar una propiedad adjunta asociada con un objeto objetivo; una lista de parámetros, que comprende:

Un identificador de propiedades que identifica la propiedad adjunta.

31. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Un identificador de objeto que identifica un objeto objetivo;

Un identificador de método que identifica un método en una clase de propiedades adjunta para afectar una propiedad adjunta asociada con el objeto objetivo;

Una lista de parámetros, comprende:

Un identificador de propiedad adjunta que incluye un identificador de clase que identifica la clase de propiedades adjunta y un identificador de

propiedades que identifica un nombre para una propiedad adjunta dentro de la clase de propiedades adjunta.

32. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Una bandera que representa un objeto objetivo;

Un identificador de propiedad adjunta que tiene un designador suministrador que identifica una clase de propiedades adjunta y un identificador de propiedades que identifica un nombre para la propiedad adjunta dentro de las clases de propiedades adjunta.

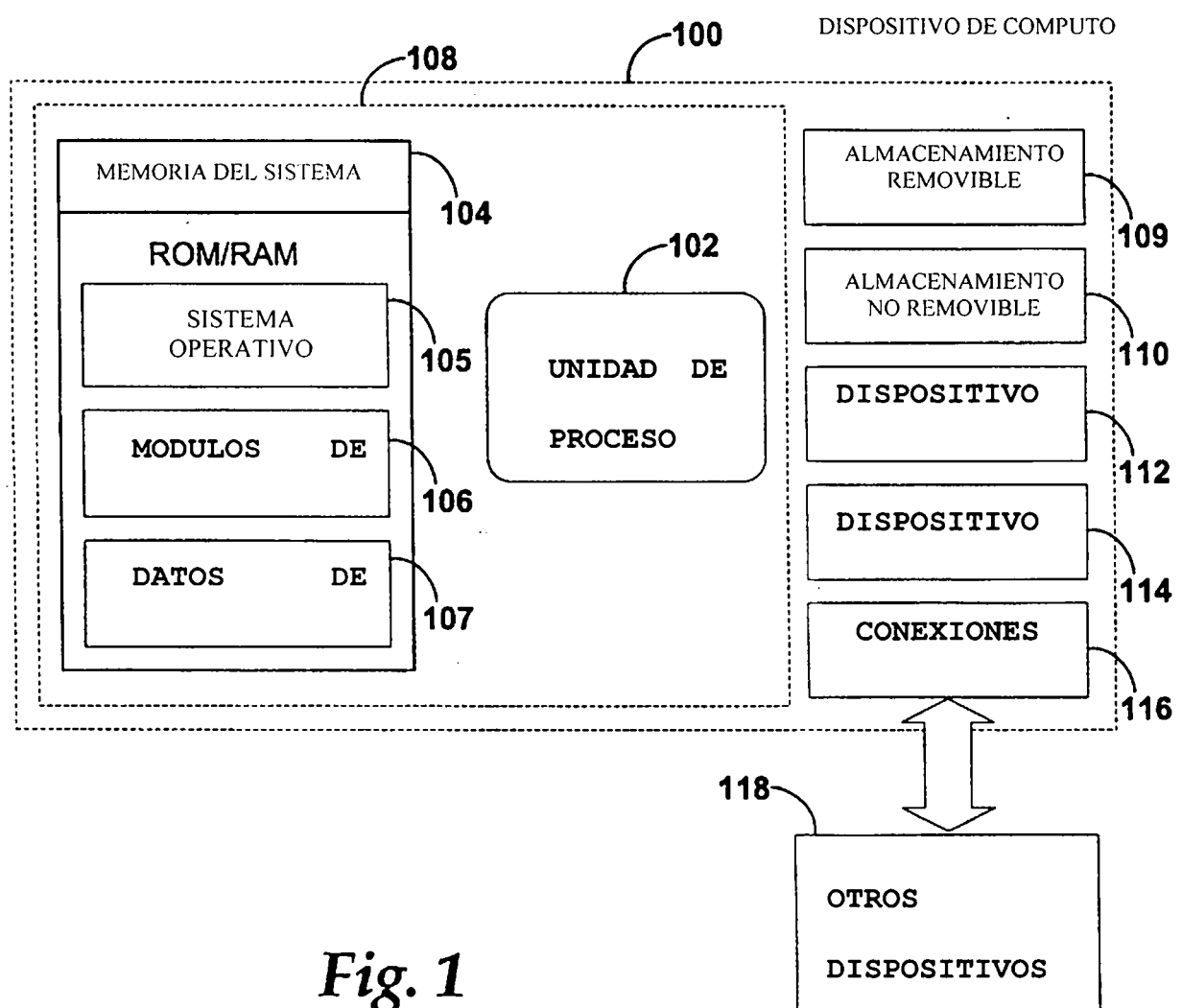
33. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Un esquema XML para definir un objeto objetivo, una clase de propiedades adjunta, y una propiedad adjunta dentro de la clase de propiedades adjunta, de tal manera que la propiedad adjunta se asocia con el objeto objetivo sin una referencia explícita a la propiedad adjunta en el objeto objetivo.

RESUMEN

Se describe aquí un mecanismo para permitir una nueva funcionalidad para un objeto de ser expresado como una propiedad que no está construida dentro de la clase de la cual el objeto se deriva. Más específicamente, el mecanismo asocia propiedades en una clase con otra clase. Un medio legible por computador, que incluye un objeto que tiene una propiedad en un primer conjunto de propiedades, y que incluye adicionalmente una estructura de datos. La estructura de datos incluye definiciones para cada uno de un segundo conjunto de propiedades e incluye por lo menos un método estático. El método estático está asociado con una propiedad sacada del segundo conjunto de propiedades e incluye un primer

parámetro. El primer parámetro identifica de manera única aquella una propiedad. El método estático es operativo para asociar dicha una propiedad con el objeto sin especificar una referencia explícita a dicha una propiedad en el objeto. La propiedad es registrada durante el tiempo de corrida para poder recibir el identificador único.



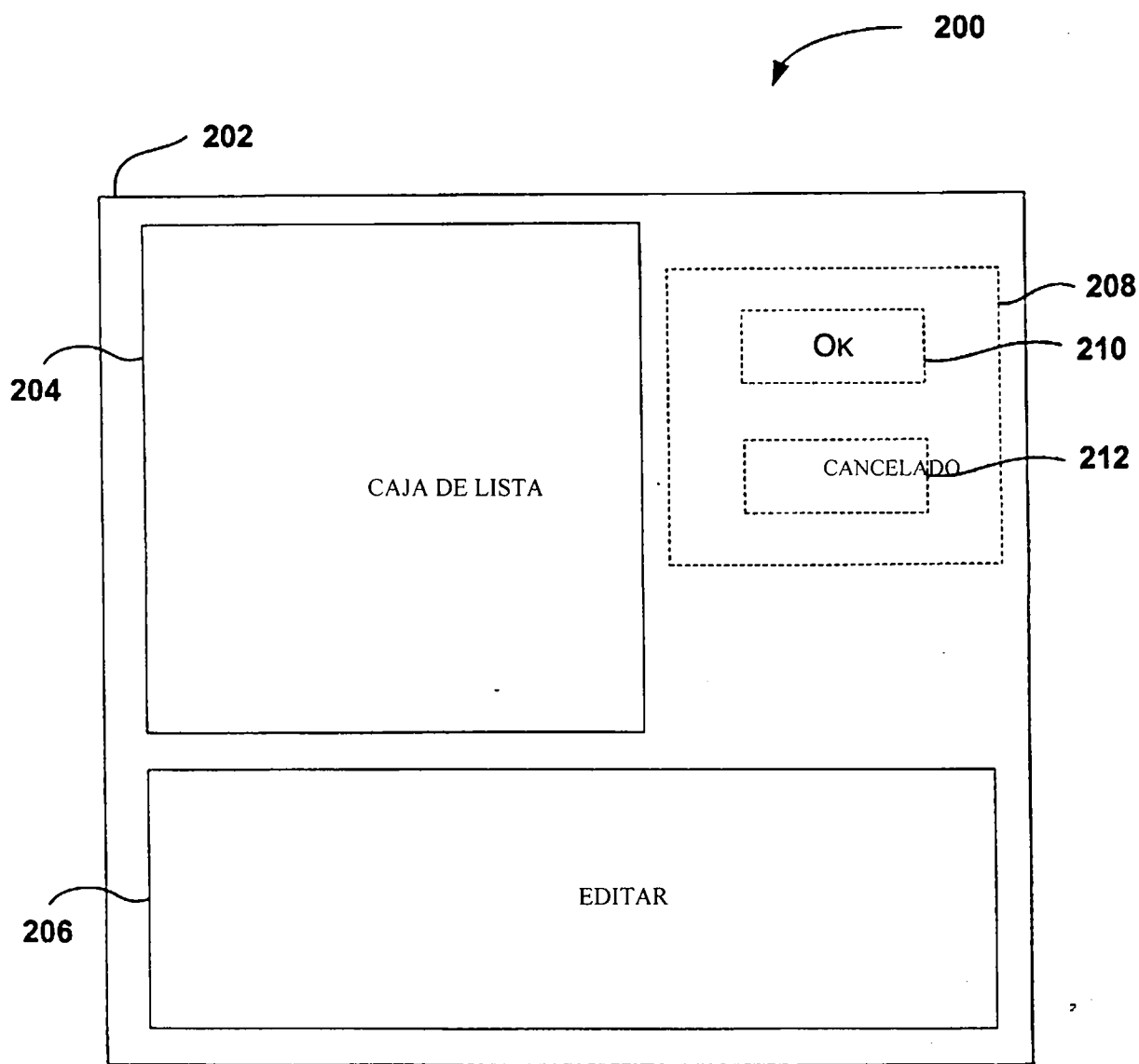


Fig. 2

3/6

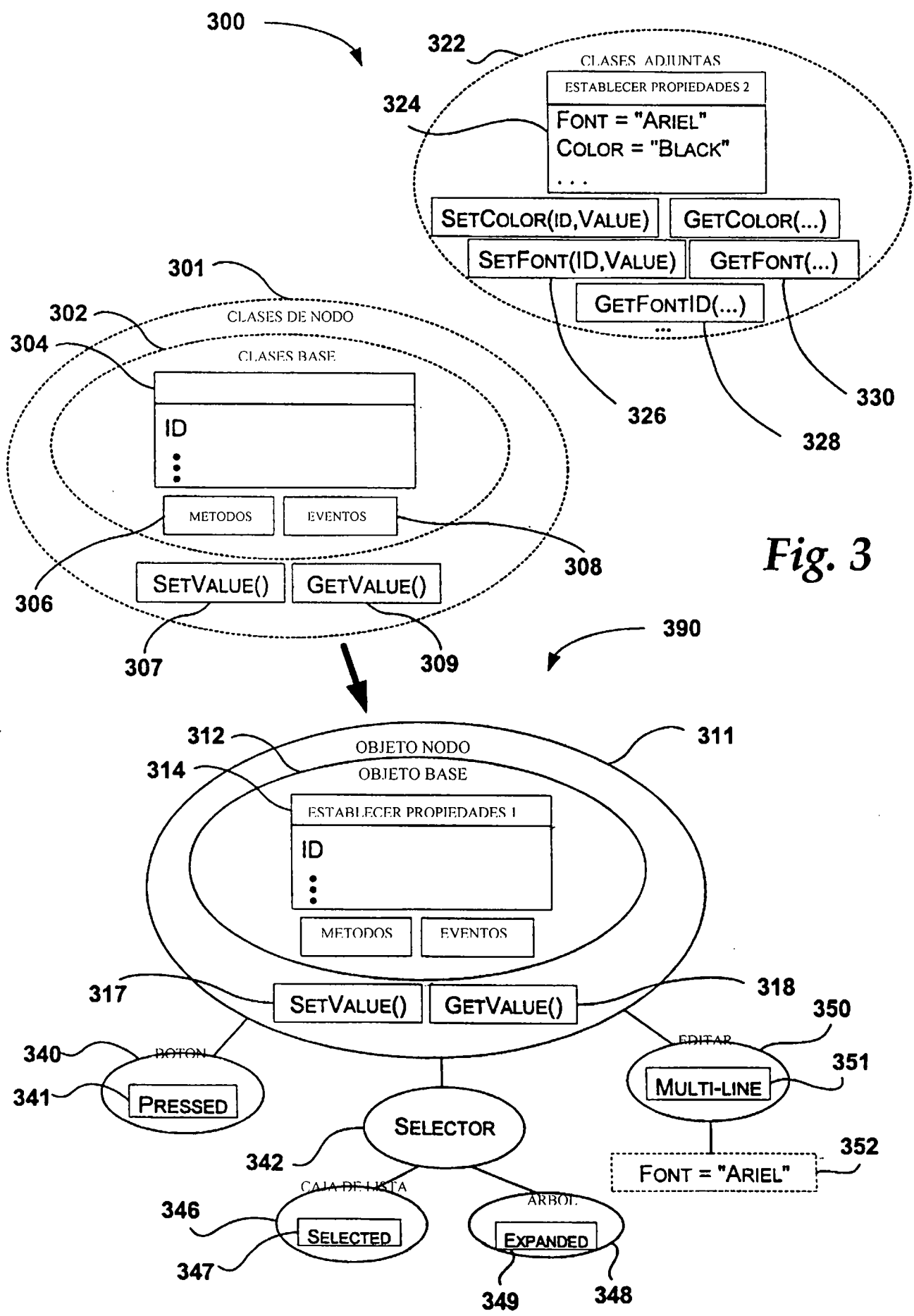


Fig. 3

4/6

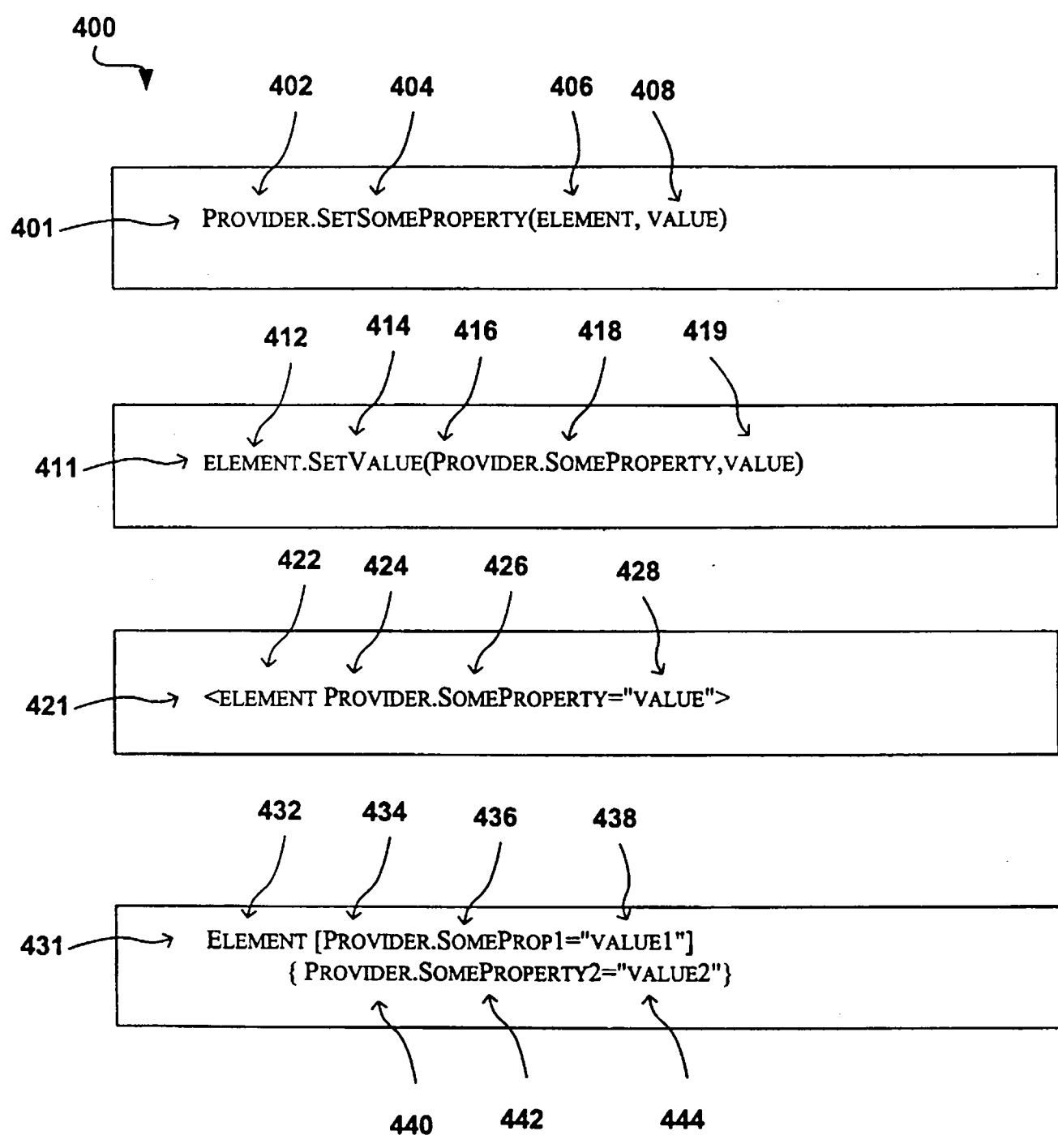


Fig. 4

117

5/6

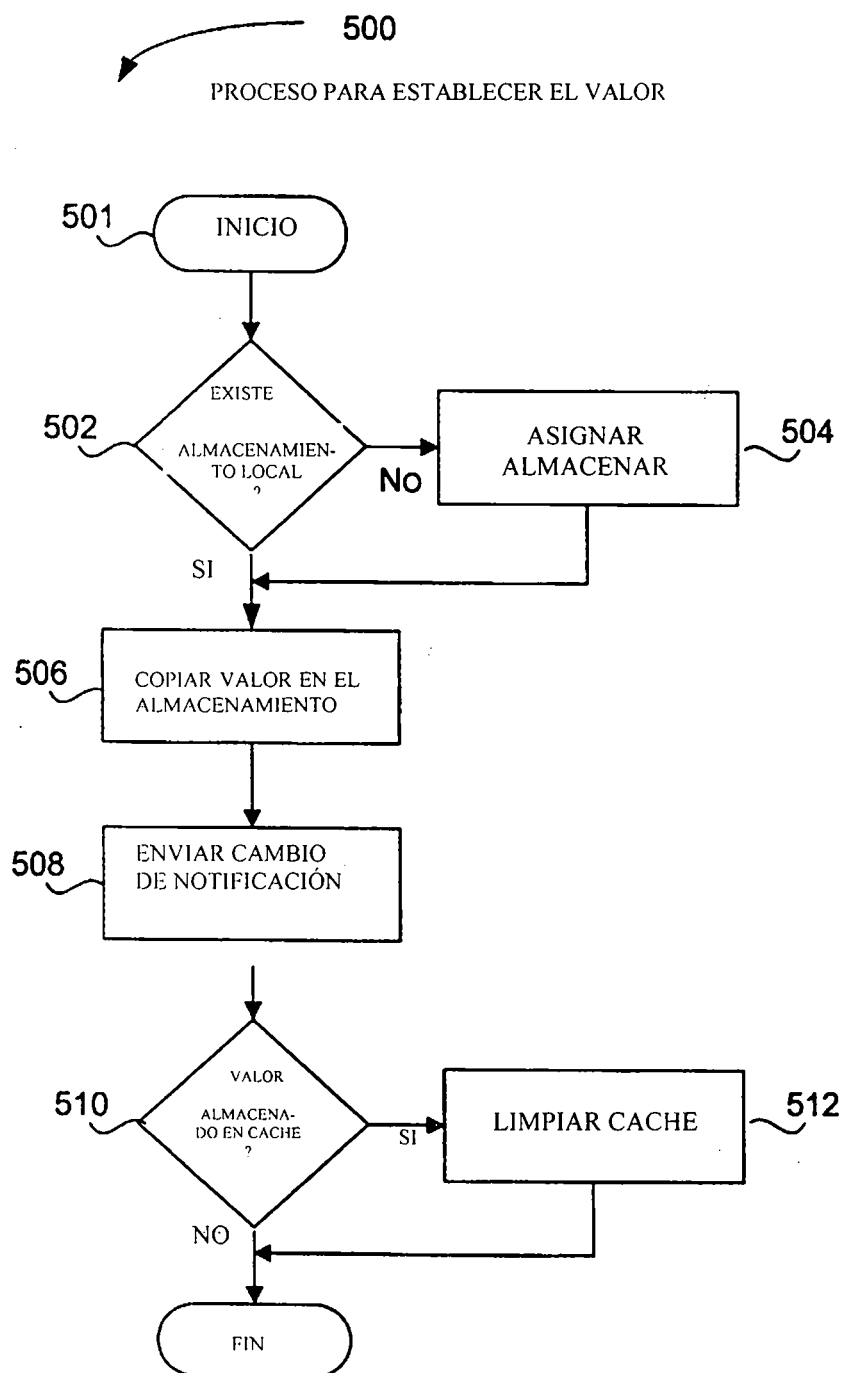


Fig. 5

6/6

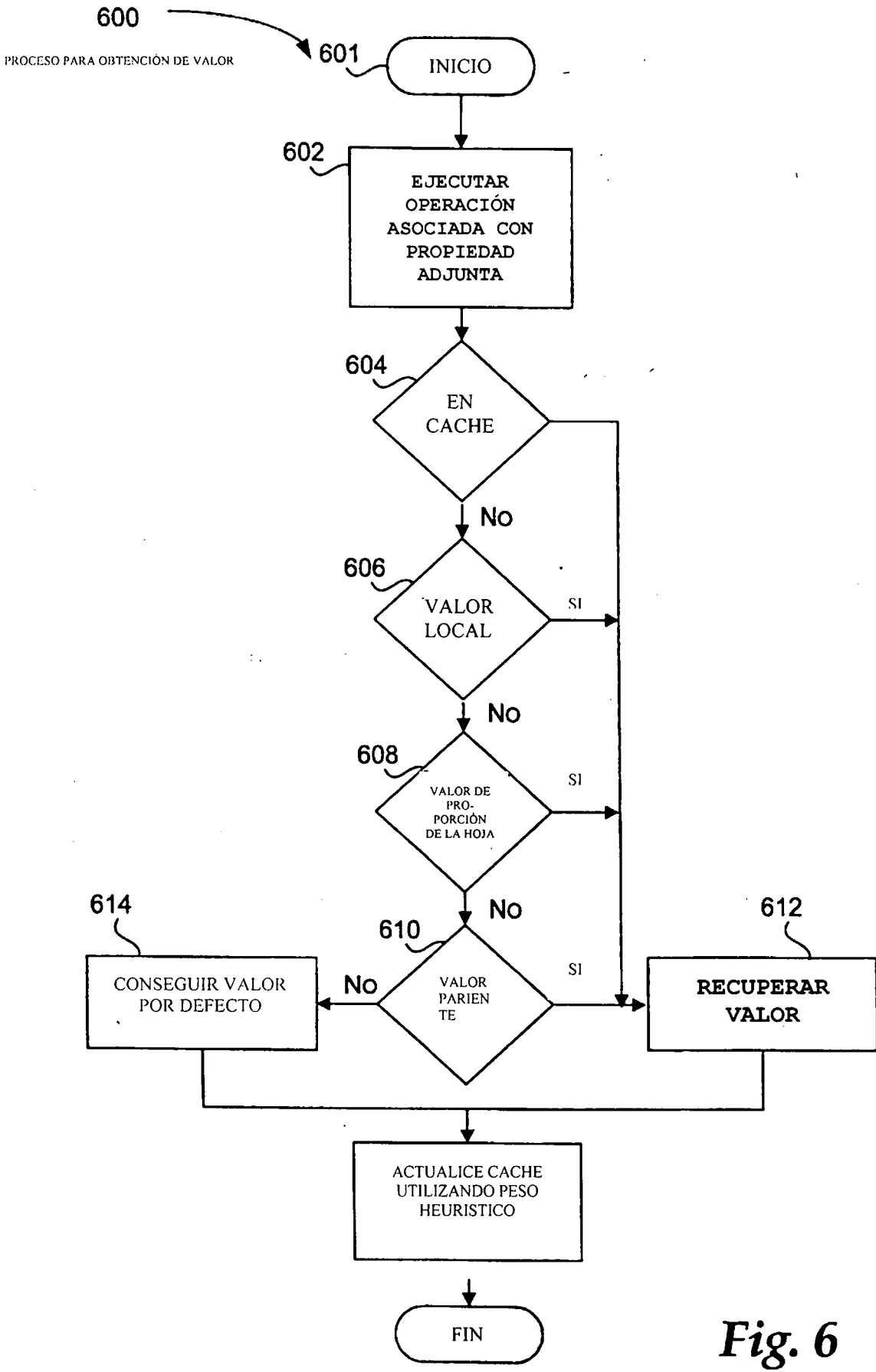


Fig. 6

119

NOTARIAL CERTIFICATE

State of Minnesota)

) ss.

County of Hennepin)

2

I hereby certify that attached hereto is a true and correct copy of the
Assignment to Microsoft Corporation of patent rights, in Patent Application Serial No.
10/187,012, filed June 28, 2002, for " SYSTEM AND METHOD FOR ASSOCIATING
PROPERTIES WITH OBJECTS".

Kristi C. Wheeler
Notary Public



Dated: April 17, 2003

[seal]

UNITED STATES
PATENT AND
TRADEMARK OFFICE

50037.0096US01

SEPTEMBER 13, 2002

PTAS

Under Secretary of Commerce For Intellectual Property and
Director of the United States Patent and Trademark Office
Washington, DC 20231
www.uspto.gov

MERCHANT & GOULD P.C.
MARCIA A. TUNHEIM
P.O. BOX 2903
MINNEAPOLIS, MN 55402-0903



102159506A

UNITED STATES PATENT AND TRADEMARK OFFICE
NOTICE OF RECORDATION OF ASSIGNMENT DOCUMENT

THE ENCLOSED DOCUMENT HAS BEEN RECORDED BY THE ASSIGNMENT DIVISION OF THE U.S. PATENT AND TRADEMARK OFFICE. A COMPLETE MICROFILM COPY IS AVAILABLE AT THE ASSIGNMENT SEARCH ROOM ON THE REEL AND FRAME NUMBER REFERENCED BELOW.

PLEASE REVIEW ALL INFORMATION CONTAINED ON THIS NOTICE. THE INFORMATION CONTAINED ON THIS RECORDATION NOTICE REFLECTS THE DATA PRESENT IN THE PATENT AND TRADEMARK ASSIGNMENT SYSTEM. IF YOU SHOULD FIND ANY ERRORS OR HAVE QUESTIONS CONCERNING THIS NOTICE, YOU MAY CONTACT THE EMPLOYEE WHOSE NAME APPEARS ON THIS NOTICE AT 703-308-9723. PLEASE SEND REQUEST FOR CORRECTION TO: U.S. PATENT AND TRADEMARK OFFICE, ASSIGNMENT DIVISION, BOX ASSIGNMENTS, CG-4, 1213 JEFFERSON DAVIS HWY, SUITE 320, WASHINGTON, D.C. 20231.

RECORDATION DATE: 06/28/2002

REEL/FRAME: 013084/0406
NUMBER OF PAGES: 3

BRIEF: ASSIGNMENT OF ASSIGNOR'S INTEREST (SEE DOCUMENT FOR DETAILS).

ASSIGNOR:

BOGDAN, JEFFREY L.

DOC DATE: 06/28/2002

ASSIGNOR:

FINOCCHIO, MARK J.

DOC DATE: 06/28/2002

ASSIGNOR:

KRAMER, NICHOLAS M.

DOC DATE: 06/28/2002

ASSIGNEE:

MICROSOFT CORPORATION
ONE MICROSOFT WAY
REDMOND, WASHINGTON 98052

SERIAL NUMBER: 10187012
PATENT NUMBER:

FILING DATE: 06/28/2002
ISSUE DATE:

9

013084/0406 PAGE 2

121

PAULA MCCRAY, EXAMINER
ASSIGNMENT DIVISION
OFFICE OF PUBLIC RECORDS

07-18-2002

FORM PTO-1595

(Rev. 6-93)

OMB No. 0651-0011 (exp. 4/94)

M&G- 50037.0096US01



ET

U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Office

102159506

To the Commissioner of Patents and Trademarks: Please record the attached original documents or copy thereof.

1. Name of conveying party(ies):

Jeffrey L. Bogdan
Mark J. Finocchio
Nicholas M. Kramer

6-28-02

2. Name and address of receiving party(ies):

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052Additional name(s) of conveying party(ies) attached? ☐ Yes ☒ NoAdditional name(s) & address(es) attached? ☐ Yes ☒ No

3. Nature of conveyance:

- ☒ Assignment ☐ Merger
☐ Security Agreement ☐ Change of Name
☐ Other:

Execution Date: June 28, 2002

4. Application number(s) or patent number(s):

10187012

If this document is being filed together with a new application, the execution date of the application is: June 28, 2002

A. Patent Application No.(s)

B. Patent No.(s)

Additional numbers attached? ☐ Yes ☒ No

5. Name and address of party to whom correspondence concerning document should be mailed:

Name: Marcia A. Tunheim
Address: Merchant & Gould P.C.
P.O. Box 2903
Minneapolis, MN 55402-0903

6. Total number of applications and patents involved: 1

7. Total fee (37 CFR 3.41): \$40.00

- ☒ Enclosed
☐ Authorized to be charged to deposit account

8. Please charge any additional fees or credit any overpayments to our Deposit account number: 13-2725

DO NOT USE THIS SPACE

9. Statement and signature:

To the best of my knowledge and belief, the foregoing information is true and correct and any attached copy is a true copy of the original document.

Marcia A. Tunheim; Reg. No. 42,189

Name of Person Signing

Signature

June 28, 2002

Date

Total number of pages including cover sheet, attachments, and document:

Do not detach this portion

Mail documents to be recorded with required cover sheet information to:

Box Assignments
Director - U.S. Patent and Trademark Office
Washington, D.C. 20231

Public burden reporting for this sample cover sheet is estimated to average about 30 minutes per document to be recorded, including time for reviewing the document and gathering the data needed, and completing and reviewing the sample cover sheet. Send comments regarding this burden estimate to the U.S. Patent and Trademark Office, Office of information systems, PK2-1000C, Washington, D.C. 20231, and to the Office of Management and Budget, Paperwork Reduction Project (0651-0011), Washington, D.C. 20503.

07/17/2002 AHMED1 00000088 10187012

01 FC:581

40.00 DP

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventorship Jeffrey L. Bogdan et al.
Applicant Microsoft Corporation
Attorney's Docket No. 50037.96US01
MS Docket No. 196295.1
Title: SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS

PATENT ASSIGNMENT**PARTIES TO THE ASSIGNMENT****Assignor(s):**

Jeffrey L. Bogdan
25936 NE 29th Place
Redmond, WA 98053

Mark J. Finocchio
4631 168th Lt. NE
Redmond, WA 98052

Nicholas M. Kramer
3728 Wallingford Avenue
Seattle, WA 98103

Assignee:

Microsoft Corporation
A Corporation in the State of Washington
One Microsoft Way
Redmond, WA 98052

AGREEMENT

WHEREAS, ASSIGNORS (listed above) are inventors of an invention entitled SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS as described and claimed in the specification forming part of an application for United States letters patent referenced above;

WHEREAS, Microsoft Corporation (hereinafter referred to as ASSIGNEE), a corporation of the State of Washington having a place of business at One Microsoft Way, Redmond, WA 98052, is desirous of acquiring the entire right, title and interest in and to the invention and in and to any letters patent that may be granted therefor in the United States and in any and all foreign countries;

124

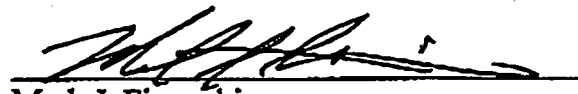
NOW, THEREFORE, in exchange for good and valuable consideration, the receipt of which is hereby acknowledged, ASSIGNORS hereby sell, assign and transfer unto ASSIGNEE, the entire right, title and interest in and to said invention, said application and any and all letters patent which may be granted for said invention in the United States of America and its territorial possessions and in any and all foreign countries, and in any and all divisions, reissues and continuations thereof, including the right to file foreign applications directly in the name of ASSIGNEE and to claim priority rights deriving from said United States application to which said foreign applications are entitled by virtue of international convention, treaty or otherwise, said invention, application and all letters patent on said invention to be held and enjoyed by ASSIGNEE and its successors and assigns for their use and benefit and of their successors and assigns as fully and entirely as the same would have been held and enjoyed by ASSIGNORS had this assignment, transfer and sale not been made. ASSIGNORS hereby authorize and request the Commissioner of Patents and Trademarks to issue all letters patent on said invention to ASSIGNEE. ASSIGNORS agree to execute all instruments and documents required for the making and prosecution of applications for United States and foreign letters patent on said invention, for litigation regarding said letters patent, or for the purpose of protecting title to said invention or letters patent therefor.

6-28-2002

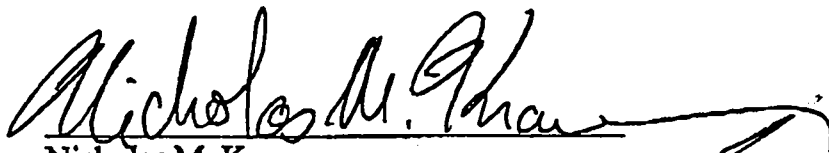
Date


Jeffrey L. Bogdan6-28-02

Date


Mark J. Finocchio6/28/2002

Date


Nicholas M. Kramer

Handwritten initials and date:
125

CERTIFICADO NOTARIAL

Estado de Minnesota)

) ss.

Condado de Hennepin)

Por medio de la presente certifico que los documentos adjuntos son una copia fiel y correcta de los derechos de patente traspasados a Microsoft Corporation, en la solicitud de patente número de serie 10/187,012, presentada en Junio 28, 20002, para "SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS".

(Aparece firma)

Notario Público

(Aparece sello que dice:

KRISTI C. WHEELER

Notario Público

Minnessota

Mi Comisión expira en Enero 31, 2005)

Fechado Abril, 17, 2003.

[Sello]

126

**OFICINA DE MARCAS Y PATENTES
DE LOS ESTADOS UNIDOS**

Conforme la Secretaría de Comercio para la Propiedad
Intelectual y

El Director de la Oficina de Marcas y Patentes de los
Estados Unidos

Washington, DC 20231

www.uspto.gov

Septiembre 13, 2002.

PTAS

MERCHANT & GOULD P. C.

MARCIA A. TUNHEIM

P.O. BOX 2903

MINNEAPOLIS, MN 55402-0903

OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS
CERTIFICADO DE REGISTRO DE UN DOCUMENTO DE TRASPASO

LOS DOCUMENTOS ADJUNTOS HAN SIDO REGISTRADOS POR LA
DIVISIÓN DE TRASPASOS DE LA OFICINA DE MARCAS Y PATENTES

127

DE LOS ESTADOS UNIDOS. UNA COPIA MICROFILMADA COMPLETA SE
ENCUENTRA DISPONIBLE EN LA OFICINA DE BÚSQUEDA DE
TRASPASO BAJO EL NÚMERO DE ROLLO Y CUADRO REFERENCIADO
ADELANTE.

POR FAVOR REVISE TODA LA INFORMACIÓN CONTENIDA EN ESTE
ANUNCIO. LA INFORMACIÓN CONTENIDA EN ESTE CERTIFICADO DE
REGISTRO REFLEJA LOS DATOS ACTUALES EN EL SISTEMA DE
TRASPASOS DE MARCAS Y PATENTES. SI USTED ENCUENTRA
ERRORES O TIENE PREGUNTAS RELACIONADAS CON ESTE ANUNCIO,
PUEDE CONTACTAR AL EMPLEADO CUYO NOMBRE APARECE EN ESTE
ANUNCIO EN 703-308-9723. POR FAVOR ENVIAR UNA SOLICITUD
PARA CORRECCIÓN A: OFICINA DE MARCAS Y PATENTES DE LOS
ESTADOS UNIDOS, DIVISIÓN DE TRASPASOS, CASILLA DE
TRASPASOS, CG-4, 1213 JEFFERSON DAVIS HWY, SUIT 320,
WASHINGTON, D.C. 20231.

FECHA DE REGISTRO: 06/28/202

ROLLO/CUADRO: 013084/0406

NÚMERO DE PÁGINAS:3

RESUMEN: TRASPASO DE INTERÉS DEL CEDENTE (VER DOCUMENTOS
PARA DETALLES).

1782
CEDENTE:

BOGDAN, JEFFREY L. G.

FECHA DOCUMENTO:

06/28/2002

CEDENTE:

FINOCCHIO, MARK J.

FECHA

DOCUMENTO:

06/28/2002

CEDENTE:

KRAMER, NICHOLAS M.

FECHA

DOCUMENTO:

06/28/2002

CESIONARIO:

MICROSOFT CORPORATION

ONE MICROSOFT WAY

REDMOND, WASHINGTON 98052

NÚMERO DE SERIE: 10187012

FECHA DE PRESENTACIÓN:

06/28/2002

NÚMERO DE PATENTE:

FECHA DE PUBLICACIÓN:

013084/0406 PÁGINA 2

PAULA MCCRAY, EXAMINADOR

DIVISIÓN DE TRASPASOS

OFICINA DE REGISTROS PÚBLICOS

07-18-2002

102159506

DE PTO-1595

(Rev.6-3)

ET

DEPARTAMENTO DE COMERCIO

DE ESTADOS UNIDOS

Oficina de Marcas y Patentes

OMB No. 0651-0011 (exp. 4/94)

M & G - 50037.0096US01.

Al Comisario de Marcas y Patentes: Por favor registrar los documentos originales adjuntos o copia de los mismos.

1. Nombre de la(s) parte(s) que traspasa(n):

Jeffrey L. Bodgan

Mark J. Finocchio

Nicholas M. kramer

6-28-02

Nombre(s) adicional(es) de los individuos que traspasan anexas? ☐ Si ☒ No

2. Nombre y Dirección de individuo(s) que recibe(n):

130

Microsoft Corporation

One Microsoft Way

Redmon, Wa 98052

Nombres y direcciones de la(s) parte(s) que reciben?

☐ Si ☒ No

3. Naturaleza del Documento:

☒ Traspaso

☐ Fusión

☐ Acuerdo de seguridad

☐ Cambio de Nombre

☐ Otro:

Fecha de Ejecución: Junio 28, 2002

4. Número(s) de solicitud o Número(s) de patente(s):

10187012

Si este documento está siendo presentado junto con una nueva solicitud la fecha de solicitud de la ejecución es:
Junio 28, 2002

131

A. Solicitud de Patente No. (s)

B. Patente No. (s)

Números adicionales adjuntos? ☐ Si ☒ No

5. Nombre y dirección de individuos a quienes corresponda los documentos concernientes que puedan ser enviados por correo:

Nombre Marcia A. Tunheim

Dirección: Merchant & Gould P. C.

P.O. Box 2903

Minneapolis, MN 55402-0903



6. Número total de solicitudes y patentes involucradas: 1

7. Tasa total (37 CFR 3.41): \$40.00

☒ Incluidas

☐ Autorizada para ser cargada a cuenta de depósito

132

8. Por favor cargar cualquier crédito o tasa adicional o sobre costo a nuestro número de cuenta de depósito: 13-2725

NO USE ESTE ESPACIO

9. Declaración y firmas:

De acuerdo con mi leal saber y entender, la información anterior es verdadera y correcta y cualquier copia adjunta es una copia cierta del documento original.

Marcia A. Reg. No. 42,189; Reg. No. 42,189

Fecha Junio 28, 2002

Nombre de la persona firmante firma

Número total de páginas incluyendo portada, documento y documentos adjuntos:

No despegar esta parte

133

Documentos de correspondencia a ser registrados con la información de portada solicitada a:

Box Assignments

Director - U.S. Patent and Trademark Office

Washington, D. C. 20231

El reporte de tema principal público para esta portada se estima en un promedio aproximado de 30 minutos por documento a ser registrado, incluyendo el tiempo para revisión del documento y reunión de los datos necesarios, y completar y revisar la muestra de portada. Enviar comentarios respecto al tema principal estimado a la Oficina de Marcas y Patentes de los Estados Unidos, Oficina de Sistemas de Información, PK2-1000C, Washington, D. C. 20231, y a la Oficina de Administración y Presupuesto, Proyecto de Reducción de Papel de Trabajo (0651-0011), Washington, D.C. 20503.

134

EN LA OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

Inventores.....Jefrey L. Bodgan et al.

Solicitante.....Microsoft Corporation

Certificación de Abogado

No.....50037.96US01

Certificación MS No.196295.1

Título:SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON
OBJETOS

TRASPASO-DE-PATENTE

PARTES EN EL TRASPASO

Cedente(s):

Jeffrey L. Bodgan
25936 NE 29th place
Redmond, WA 98053

Mark J. Finocchio
4631 168th Lt Ne
Redmond, WA 98052

~~Nicholas M. Kramer~~

3728 Wallingford Avenue

Seattle, WA 98103

Cesionario:

Microsoft Coporation

A Corporation in the State of Washington

One Microsoft Way

Redmond, WA 98052

ACUERDO

CONSIDERANDO, QUE LOS CEDENTES (listados anteriormente) son inventores de la invención titulada SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS como se describió y reivindicó en la especificación que forma parte de la solicitud para cartas de patente de Estados Unidos, referenciadas anteriormente;

CONSIDERANDO, que Microsoft Corporation (llamado de aquí en adelante como CESIONARIO), una corporación del Estado de Washington que tiene lugar de negocio en One Microsoft Way, Redmond, WA 98052, deseosa de adquirir los derechos

136

completos, título e interés en y a la invención y en cualquier carta de patente que por lo tanto pueda ser otorgada en los Estados Unidos y en cualquier otro país extranjero;

Página 1 de 2

POR LO TANTO, AHORA, en intercambio de buena y valiosa consideración, el recibo del cual se reconoce aquí, los CEDENTES por medio del presente venden, traspasan y transfieren al CESIONARIO, los derechos completos, título, beneficio en y a dicha invención, dicha solicitud y cualquiera y todos los títulos de patente que puedan ser otorgadas a dicha invención en los Estados Unidos de América, y sus posesiones territoriales y en cualquiera de los países extranjeros, y en cualquiera y todas las reotorgamientos y continuaciones de la misma, incluyendo el derecho a presentar solicitudes en el extranjero directamente a nombre del CESIONARIO y a reivindicar los derechos de propiedad derivados de dicha solicitud Estadounidense al cual dichas solicitudes extranjeras tienen derecho por virtud de la convención, tratado internacional, o de otra manera, dicha invención, solicitud y todos los títulos de patente sobre dicha invención a ser poseídos y disfrutados por los CESIONARIOS y sus sucesores y cede en su uso y beneficio y el de sus sucesores y cede tan completa y enteramente como si la misma hubiera sido poseída y disfrutada por el cedente si este traspaso o venta no hubiera sido hecho. Por medio de la presente los CEDENTES autorizan y solicitan al Comisionado de Marcas y Patentes a expedir

todos los títulos de patente sobre esta invención al
CESIONARIO. Los CEDENTES están de acuerdo para firmar
todos los instrumentos y documentos requeridos par
efectuar y tramitar las solicitudes para los títulos de
patente en los Estados Unidos y en el exterior, para
litigar con respecto a dichos títulos de patente, o con
el propósito de proteger el título a dicha invención o
título de patente.

6-28-2002

Fecha

(Aparece Firma)

Jeffrey L. Bodgan

6-28-02

Fecha

(Aparece Firma)

Mar J. Finocchio

6/28/2002

Fecha

(Aparece Firma)

Nicholas M. Kramer

139

OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

COMISARIO DE PATENTES
OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS
WASHINGTON, D.C. 20231
WWW.USPTO.GOV

Número de solicitud	10/187,012
Fecha de Presentación	06/28/2002
Unidad técnica de grupo	2171
Fil Fee Rec'D	1478
Registro de abogado No.	50037.0096US01
Dibujos	6
Total reivindicaciones	33
Reivindicaciones independientes	9

CONFIRMACIÓN No. 1314
RECIBO PRESENTACIÓN

27488
MERCHANT & GOULD
P.O. BOX 2903
MINNEAPOLIS, MN 55402-0903

Fecha de Correspondencia: 09/12/2002

Se reconoce el recibo de esta solicitud de patente no provisional. Se considerará en su orden y se le

[Handwritten signature]
140

notificarán los resultados del examen. Asegure suministrar el NÚMERO DE SOLICITUD ESTADOUNIDENSE, FECHA DE PRESENTACIÓN, NOMBRE DEL SOLICITANTE, y TÍTULO DE LA INVENCIÓN cuando se pregunte por esta solicitud. Las tasas de tramitación por cheque o letras de cambio están sujetas a recaudación. Por favor verifique la exactitud de los datos presentados en este recibo. **Si se nota un error en este recibo de presentación, por favor escriba a la Oficina de Examen de Patente Inicial Correcciones de Recibo de Presentación, facsímil número 703-746-9195. Por favor suministre una copia de este Recibo de Presentación con los cambios anotados en este. Si usted recibe una "Nota para Presentar las Partes que Faltan" para esta solicitud por favor envíe las correcciones a este Recibo de Presentación con su respuesta a la Nota. Cuando la USPTO procesa la respuesta a la nota, la USPTO generará otro recibo de presentación incorporando las correcciones solicitadas (si es apropiado).**

Solicitante(s)

Jeffrey L. Bodgan, Redmond, WA;

Mark J. Finocchio, Redmond, WA;

Nicholas M. Kramer, Seattle, WA;

099
141

Traspaso Para Solicitud De Patente Publicada

Microsoft Corporation, Redmond, WA

**Datos de prioridad doméstica reivindicados por el
solicitante**

Solicitudes Extranjeras

**Si se requiere, licencia de presentación extranjera
otorgada 09/12/2002**

Fecha de Publicación Proyectada: 01/01/2004

Solicitud de No Publicación: No.

Solicitud de Publicación Temprana: No

Título: Sistema y Método Para Asociar Propiedades con
Objetos

PA 965670

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

February 25, 2003

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

APPLICATION NUMBER: 10/187,012

FILING DATE: June 28, 2002

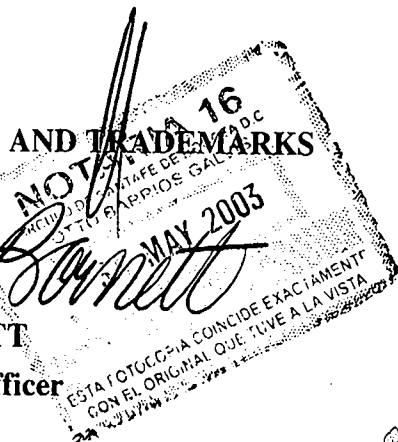


By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

E. Bornett

E. BORNETT

Certifying Officer



07-92192012.062802

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Jeffrey L. Bogdan et al.
Docket: 50037.0096US01
Title: SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS

CERTIFICATE UNDER 37 CFR 1.10

'Express Mail' mailing label number: ET886907746US
Date of Deposit June 28, 2002

I hereby certify that this paper or fee is being deposited with the United States Postal Service 'Express Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to the Commissioner for Patents, Washington, D.C. 20231

By: *Luci Arcvalo*
Name: Luci Arcvalo

BOX PATENT APPLICATION
Commissioner for Patents
Washington, D.C. 20231

Sir:

We are transmitting herewith the attached:

- ☒ Transmittal sheet, in duplicate, containing Certificate under 37 CFR 1.10.
- ☒ Utility Patent Application: Spec. 18 pgs; 33 claims; Abstract 1 pgs.
The fee has been calculated as shown below in the 'Claims as Filed' table.
- ☒ 6 sheets of drawings
- ☒ A signed Combined Declaration and Power of Attorney
- ☒ Assignment of the invention to Microsoft Corporation, Recordation Form Cover Sheet
- ☒ A check in the amount of \$1478.00 to cover the Filing Fee
- ☒ A check for \$40.00 to cover the Assignment Recording Fee.
- ☒ Application Data Sheet, 4 pages.
- ☒ Return postcard

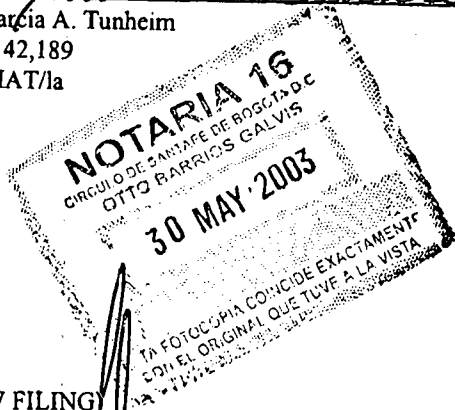
CLAIMS AS FILED

Number of Claims Filed	In Excess of:	Number Extra	Rate	Fee
Basic Filing Fee				\$740.00
Total Claims				
33	20	13	x 18.00	\$234.00
Independent Claims				
9	3	6	x 84.00	\$504.00
MULTIPLE DEPENDENT CLAIM FEE				\$0.00
TOTAL FILING FEE				\$1478.00

Please charge any additional fees or credit overpayment to Deposit Account No. 13-2725. A duplicate of this sheet is enclosed.

MERCHANT & GOULD P.C.
P.O. Box 2903, Minneapolis, MN 55402-0903
(206) 342-6200

By: *Marcia A. Tunheim*
Name: Marcia A. Tunheim
Reg. No.: 42,189
Initials: MAT/la



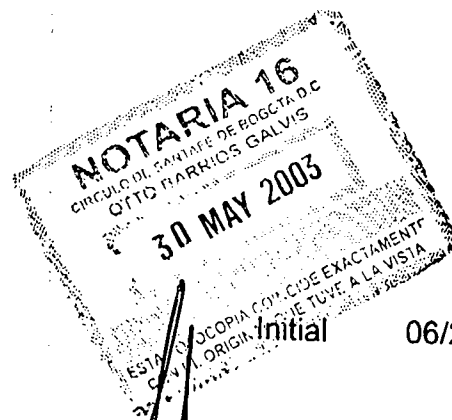
(PTO TRANSMITTAL - NEW FILING)

144

Application Data Sheet

Application Information

Application Type::	Regular
Subject Matter::	Utility
Suggested Classification::	
Suggested Group Art Unit::	
CD-ROM or CD_R?::	None
Number of CD disks::	
Number of copies of CDs::	
Sequence Submission::	No
Computer Readable Form (CRF)?::	No
Title::	SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS
Attorney Docket Number::	50037.0096US01
Request For Early Publication::	No
Request For Non-Publication::	No
Suggested Drawing Figure::	
Total Drawing Sheets::	6
Small Entity::	No
Latin Name::	
Variety Denomination Name::	
Petition Included::	No
Petition Type::	
Licensed US Govt. Agency::	
Contract or Grant Numbers::	
Secrecy Order in Parent Appl.?::	No



145

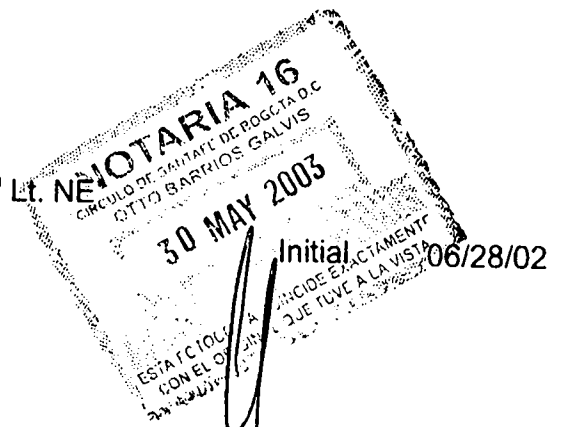
Applicant Information

Applicant Authority Type:: Inventor
 Primary Citizenship Country:: USA
 Status:: Full Capacity
 Given Name:: Jeffrey
 Middle Name:: L.
 Family Name:: Bogdan
 Name Suffix::
 City of Residence:: Redmond
 State or Province of Residence:: WA
 Country of Residence:: USA
 Street of mailing address:: 25936 NE 29th Place
 City of mailing address:: Redmond
 State or Province of mailing address:: WA
 Country of mailing address:: USA
 Postal or Zip Code of mailing address:: 98053

Applicant Information

Applicant Authority Type:: Inventor
 Primary Citizenship Country:: USA
 Status:: Full Capacity
 Given Name:: Mark
 Middle Name:: J.
 Family Name:: Finocchio
 Name Suffix::
 City of Residence:: Redmond
 State or Province of Residence:: WA
 Country of Residence:: USA
 Street of mailing address:: 4631 168th Lt.

Page # 2



146

City of mailing address:: Redmond
State or Province of mailing address:: WA
Country of mailing address:: USA
Postal or Zip Code of mailing address:: 98052

Applicant Information

Applicant Authority Type:: Inventor
Primary Citizenship Country:: USA
Status:: Full Capacity
Given Name:: Nicholas
Middle Name:: M.
Family Name:: Kramer
Name Suffix::
City of Residence:: Seattle
State or Province of Residence:: WA
Country of Residence:: USA
Street of mailing address:: 3728 Wallingford Ave.
City of mailing address:: Seattle
State or Province of mailing address:: WA
Country of mailing address:: USA
Postal or Zip Code of mailing address:: 98103

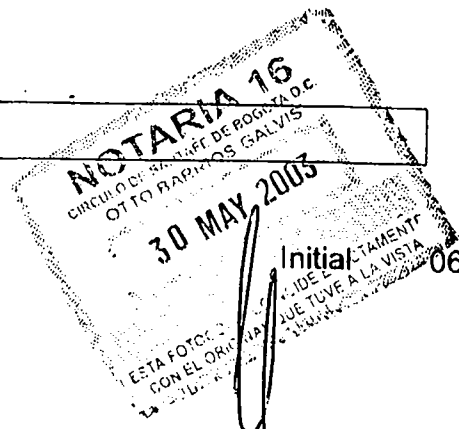
Correspondence Information

Correspondence Customer Number:: 27488

Representative Information

Representative Customer Number::	27488
----------------------------------	-------

Page # 3



06/28/02

10187

Assignee Information

Assignee Name:: Microsoft Corporation
Street of mailing address:: One Microsoft Way
City of mailing address:: Redmond
State or Province of mailing address:: WA
Country of mailing address:: USA
Postal or Zip Code of mailing address:: 98052

Page # 4



148

M&G No. 50037.0096US01/196295.1

SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS

Field of the Invention

5 The present invention relates generally to software applications, and, more particularly, to mechanisms for managing properties of objects within a software application.

Background of the Invention

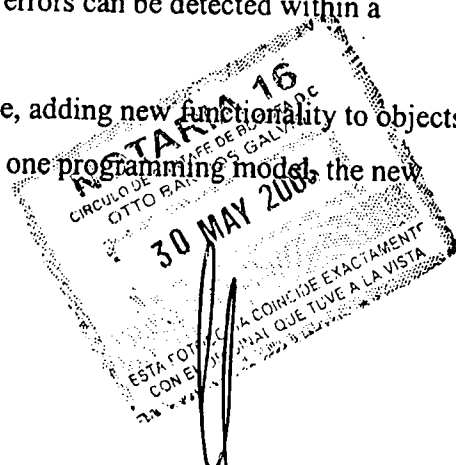
10 Most programming models today support the concept of classes. These classes are typically structured in a hierarchical tree having branches that represent different classes in the class hierarchy. When two branches are at different levels, the lower branch represents a child class. The child class inherits information from the class associated with the upper branch (i.e., the parent class). When two branches are at the same level, the classes are referred to as sibling classes. For the purposes of the

15 following discussion, when referring to a child class in relation to a parent class, the terms, lower class and upper class, may be used to refer to the child class and the parent class, respectively. The uppermost branch in the hierarchy represents a base class in the hierarchical class tree. Typically, information in each class includes properties, methods, and events. The properties describe characteristics associated with the class.

20 For example, a button class may have properties such as width, background color, font type, visible, and pressed. When one of these classes is instantiated, an object of that class is created. Each property in the object has an associated value, which can be queried and set during run-time operation. The value that is queried or set may be expected to conform to a specific data type if the syntax is strongly typed. Having

25 syntax that is strongly typed is desirable because errors can be detected within a software application before run-time operation.

 Once the class hierarchy is in place, adding new functionality to objects within the class hierarchy presents a problem. In one programming model, the new



149

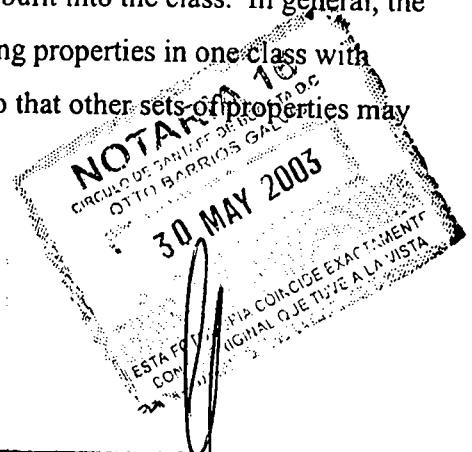
functionality is forced into the base class. When this is done, the base class becomes very large (e.g., one hundred methods, fifty properties, and twenty events), which results in an almost unmanageable object hierarchy. One undesirable outcome of this programming model is that the number of properties, methods, and events within the base class becomes overwhelming for developers to fully understand before implementing desired features into objects they create. Yet another undesirable outcome of this programming model is that the memory requirements become enormous due to the fact that values for the properties are stored locally. Because the values are stored locally, applications created with this programming model do not scale well.

This programming model also causes other problems for third party developers. Third party developers who wish to add new functionality must add a child class at the bottom of the hierarchy. Because the new child class is at the bottom of the hierarchy, the new functionality is not available to other classes within the hierarchy. Therefore, third party developers may need to add the new functionality to several child classes. As one can imagine, this results in code duplication, which impacts the maintainability of the object hierarchy. For the above reasons, this programming model is not very desirable.

Until the present invention, a programming model that allowed new functionality to be provided to a class within an existing class hierarchy without the above shortcomings has eluded those skilled in the art.

Summary of the Invention

The present invention provides a mechanism that allows new functionality to be provided to a class without having the new functionality become a permanent part of the class. In addition, the present invention allows the new functionality to be expressed as a property that is not built into the class. In general, the present invention provides a mechanism for associating properties in one class with another class. This association is easily modifiable so that other sets of properties may be associated with the class.



180

In one embodiment, a computer-readable medium that includes an object having a property in a first set of properties, further includes a data structure. The data structure includes definitions for each of a second set of properties and includes at least one static method. The static method is associated with one property out of the second set of properties and includes a first parameter. The first parameter uniquely identifies the one property. The static method is operative to associate the one property with the object without specifying an explicit reference to the one property in the object.

In one aspect of the invention, the static method supports a strongly typed syntax.

10 In another aspect of the invention, the static method includes retrieving a value for the object without having the value stored locally on the object. The value may be retrieved from several stages, such as a parent object or a property sheet.

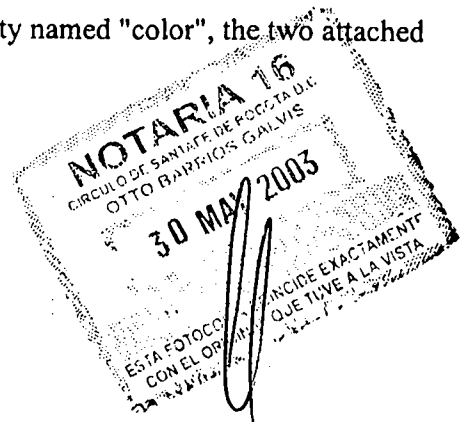
One advantage of the present invention is that the dividing of properties into one or more subsets allows each subset of properties to be more easily maintained. 15 Another advantage of the present invention is that the object hierarchy is more extensible and allows developers to add functionality to the object hierarchy that impacts the base class and all lower classes.

Another advantage of the present invention is that managing storage for the properties in the objects becomes more efficient and convenient.

20 Yet another advantage of the present invention is that the programming model operates within a programmatic or a markup environment. In addition, the programming model operates in strongly typed programming languages, such as C++ and C#. In addition, the programming model supports property sheets, change notifications and value inheritance.

25 Another advantage of the present invention is that independent libraries can coexist because name conflicts are less likely because the name of the attached class effectively becomes part of the property name. Therefore, if two different developers each create an attached property having a property named "color", the two attached properties will not conflict.

3



Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention.

FIGURE 2 is an exemplary display that may be created with the computing device of FIGURE 1.

FIGURE 3 is a graphical representation of a programming model that allows properties from one class to be attached to another class in accordance with the present invention.

FIGURE 4 illustrates several exemplary syntaxes for implementing the programming model shown in FIGURE 3.

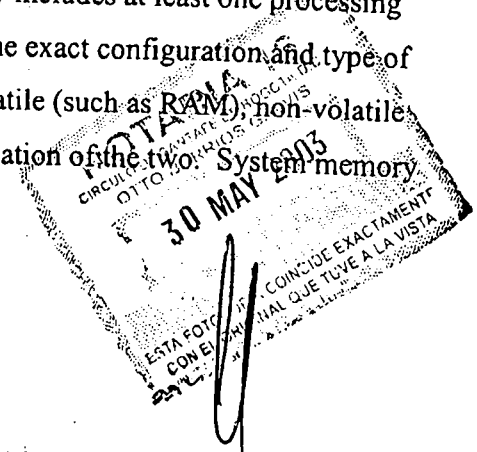
FIGURE 5 is a logical flow diagram illustrating a process for setting a value in accordance with the present invention.

FIGURE 6 is a logical flow diagram illustrating a process for retrieving a value in accordance with the present invention.

Detailed Description of the Preferred Embodiment

Briefly stated, the present invention provides a programming model that allows new functionality to be provided to a class without having the new functionality become a permanent part of the class. In addition, the present invention allows the new functionality to be expressed as a property that is not built into the class. In general, the present invention provides a mechanism for associating properties in one class with another class. As will become apparent after reading the detailed description below, the programming model of the present invention provides dynamic properties to objects without building the properties into the object.

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device 100. In a very basic configuration, computing device 100 typically includes at least one processing unit 102 and system memory 104. Depending on the exact configuration and type of computing device, system memory 104 may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory

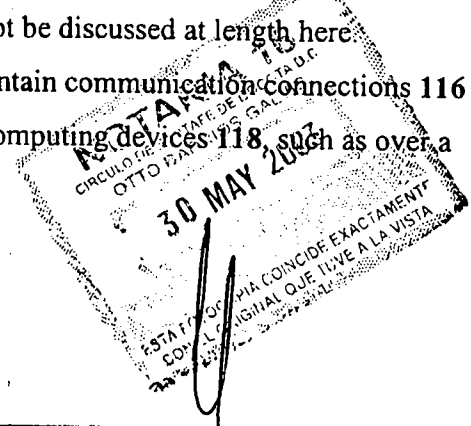


PMB
152

104 typically includes an operating system 105, one or more program modules 106, and may include program data 107. Examples of program modules 106 include Visual Studio IntelliSense from Microsoft Corporation of Redmond, WA, and other software programming environments, which utilize object libraries. In addition, program
5 modules 106 include software applications created using a software-programming environment. When these software applications execute on processing unit 102, a property engine processes the software application in accordance with the programming model of the present invention. The property engine may be part of operating system 105 or may be another program module 106. This basic configuration of computing
10 device 100 is illustrated in FIGURE 1 by those components within dashed line 108.

Computing device 100 may have additional features or functionality. For example, computing device 100 may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage 109
15 and non-removable storage 110. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory 104, removable storage 109 and non-removable storage 110 are all examples of computer storage
20 media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device 100. Any such computer
25 storage media may be part of device 100. Computing device 100 may also have input device(s) 112 such as keyboard, mouse, pen, voice input device, touch input device, etc. Output device(s) 114 such as a display, speakers, printer, etc. may also be included. These devices are well known in the art and need not be discussed at length here.

Computing device 100 may also contain communication connections 116
30 that allow the device to communicate with other computing devices 118, such as over a



5

15

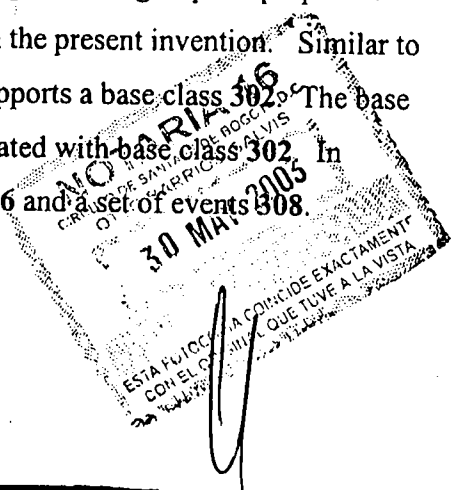
30

be described in detail below, in accordance with the present invention, separate classes provide the actions and the appearance for objects 202 - 212 without resorting to an explicit reference between the two classes.

5 This allows developers to modify either the action portion or the appearance portion without modifying the entire object. Thus, in this user-interface example, each object is factored into two parts. One part relates to the action (i.e., behavior) associated with the object, and, the other part relates to the appearance (i.e., rendering) of the object. Because the rendering is separated from the behavior of the object, the appearance of the object may be easily modified to change the way the object appears.

10 While the above user-interface example conveniently factors the properties for the objects into a behavior group and an appearance group, the inventors have determined that properties of many objects in various environments can be conveniently divided into two or more relevant groups. Then, in accordance with the present invention, these relevant groups can be "implicitly" associated with each other to fully implement the object. This "implicit" association provides several advantages over prior programming models. One of the advantages is that third party developers can conveniently modify the relevant group that pertains to their application while keeping the other groups untouched. This greatly simplifies the amount of information that the developer needs to know in order to change the functionality of the object. In addition, the object hierarchy becomes more manageable. Another advantage is that storage is reduced because each object does not necessarily maintain a local value with a current state. Rather, as will be described in detail in conjunction with FIGURE 6, the state may be retrieved from various sources when needed.

25 FIGURE 3 is a graphical representation of a programming model 300 that provides a mechanism for "implicitly" associating relevant groups of properties from one class with another class in accordance with the present invention. Similar to other programming models, the present invention supports a base class 302. The base class 302 includes a first set of properties 304 associated with base class 302. In addition, base class 302 includes a set of methods 306 and a set of events 308.



155

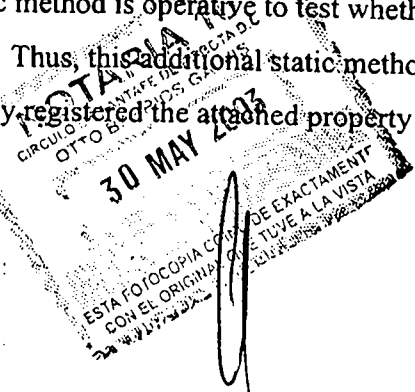
However, in accordance with the present invention, the base class is derived from a node class 301. The node class 301 provides a SetValue() method 307 and a GetValue() method 309.

A second set of properties 324 is included within an attached class 322.

- 5 The attached class 322 also includes at least two static methods associated with each of the second set of properties 324 (e.g., SetFont() 326 and GetFont() 330). These static methods may be thought of as global methods that are accessible to objects that have registered the property associated with the static method. The developer supplying the attached class 322 is responsible for writing the code for each of these static methods
- 10 326 and 330. The code will include a call to one of the methods provided by the node class 301. For example, an application calling SetFont() 326 for a specific identifier PropertyID and a given font Value, will ultimately execute the call to SetValue() 307 in the node class 301. The parameters that SetFont() 326 passes to the call to SetValue() 307 will include the specific identifier PropertyID and the given font Value that were
- 15 provided in the SetFont() 326 call. An exemplary SetFont() function may appear as follows:

```
SetFont(PropertyID,white)
{ PropertyID->SetValue(PropertyID,white);
20 }
```

- One skilled in the art will appreciate that the attached class, thus, provides strong typing for any attached property. While the present invention supports the application calling the SetValue() 307 directly, this circumvents the advantage of
- 25 providing strong typing during compile time. As mentioned earlier, strong typing allows errors to be detected before run-time operation. In another embodiment, the attached class 322 provides an additional static method (e.g., GetFontID() 328) for each of the attached properties 324. This additional static method is operative to test whether the requested attached property has been registered. Thus, this additional static method
- 30 328 is used to insure that the application has properly registered the attached property



156

before attempting to perform the set and get static methods on the attached property. For example, if GetFontID() 328 detects that the Color attached property has not been registered, GetFontID() may give an error, may automatically register the Color attached property for the requesting object, or the like.

5 The following is an exemplary embodiment for providing the delayed registration:

```

class FontProvider
{
    public static Font GetFont(Node n)
10 { return n.GetValue(GetFontID()); }

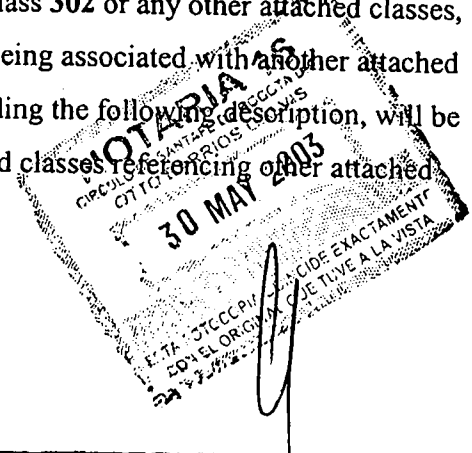
    public static void SetFont(Node n, Font newFont)
    { n.SetValue(GetFontID(), newFont); }

15 public static DynamicProperty GetFontID()
    { // Delay register Font property if necessary
      if (FontID == null)
        { FontID = RegisterProperty("Font", typeof(Font), "Arial", ...);
        }
20 }

    private static DynamicProperty FontID = null;
}

```

25 Similar to other programming models, the present invention supports child classes (not shown) of base class 302. For simplicity, FIGURE 3 does not graphically illustrate any child classes of base class 302 or any other attached classes, and, therefore, does not illustrate child classes being associated with another attached class. However, one skilled in the art, after reading the following description, will be
30 able to easily design an application that has child classes referencing other attached

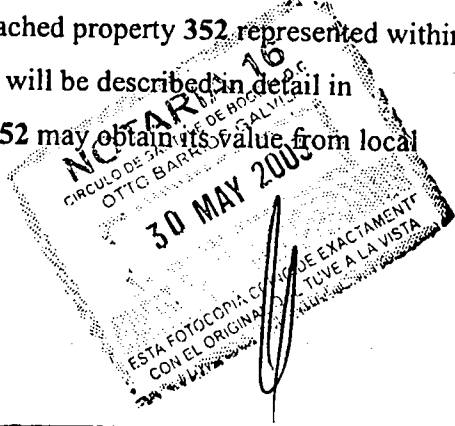


classes. In general, any class may be a "provider" of attached properties, as long as the class provides the necessary static methods for the attached properties.

The attached class 322 allows third party developers the ability to dynamically add properties and functionality to their applications without modifying the base class 302. Similar to modifying properties within base class 302, the added properties (e.g., second set of properties 324) affect objects within an entire hierarchical object tree 390. The attached class 322 is not instantiated during the execution of the application. Therefore, for each object instantiated from base class 302, the same attached class (attached class 322) may provide the second set of properties 324 to each of these objects.

During run-time of the application, a base object 312 becomes instantiated. In addition, one or more child objects (e.g., child objects 340 - 350) become instantiated. The instantiation of the objects in object tree 390 may be through programmatic control (e.g., C#), through markup language (e.g., XML), or through other means. When the object tree 390 is created through programmatic control, a compiler typically identifies whether the instructions implementing the attached class 322 has any errors, such as illegal name or data type. As discussed above, the attached class 322 provides strong typing. Therefore, errors may be caught before run-time. In addition, software development tools may detect errors during development of the application. When object tree 390 is created through markup language, a parser interprets markup statements. In general, a name of a tag in the markup language is the name for the corresponding class.

Each of these child objects 340 - 350 may include one or more child properties (e.g., child property 341) within child object (e.g., child object 340). Values for child properties may be stored in the associated child object. However, in accordance with the present invention, the child objects do not necessarily have local storage for storing a value. In addition, each of these child objects 340 - 350 may include one or more attached properties (e.g., attached property 352 represented within dashed box) associated with the child object. As will be described in detail in conjunction with FIGURE 6, attached property 352 may obtain its value from local



158

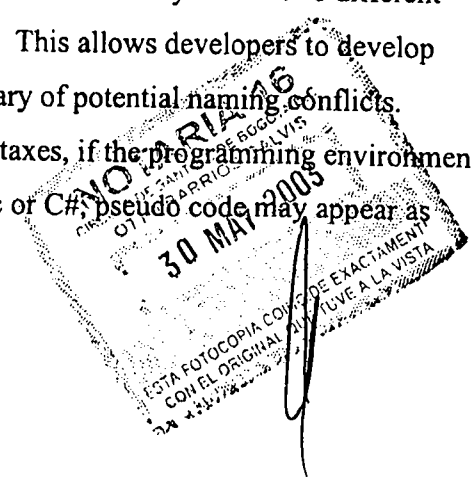
storage associated with child object 350, may obtain its value through inheritance (e.g., base object 312), through a property sheet (not shown), through programmatic methods (e.g., static method 330 in conjunction with method 309), and through other means in accordance with the present invention. While these values for the attached properties are associated with the child object, the storage for these values is not typically part of the child object.

Again, FIGURE 3 illustrates the programming model in a user-interface environment. Thus, one will note that the properties within base object 312 and child objects 340 -350 relate to the behavior of the objects. Base object 312 represents a dialog object, child object 340 represents a button object, child object 350 represents an edit box, and child object 342 represents a selector object having two child objects 346 and 348, representing a list box and a tree, respectively. Each of these child objects 340 - 350 includes one or more child properties (e.g., child property 341), such as a pressed property for button object and expanded property for tree object.

In contrast, the properties within attached class 322 relate to the appearance for the object. For example, the second set of properties may include a property and default value for font type, color, and the like. This programming model is configured to set a property at run-time on the child objects. To that extent, the attached property (e.g., attached property 352) in each of the child objects is configured to receive the value from the attached class.

FIGURE 4 illustrates several exemplary syntaxes for various operating environments in which the programming model shown in FIGURE 3 may be implemented. The use of the term "Provider" within each of the exemplary syntaxes denotes that the static methods described in the syntaxes are associated with an attached class having a name of "Provider". Thus, because each of these syntaxes allows the attached class to be named, properties that are named identically within two different attached classes will not conflict with each other. This allows developers to develop attached property classes without having to be wary of potential naming conflicts.

Now turning to the exemplary syntaxes, if the programming environment is a programmatic language, such as Visual Basic or C#, pseudo code may appear as



159

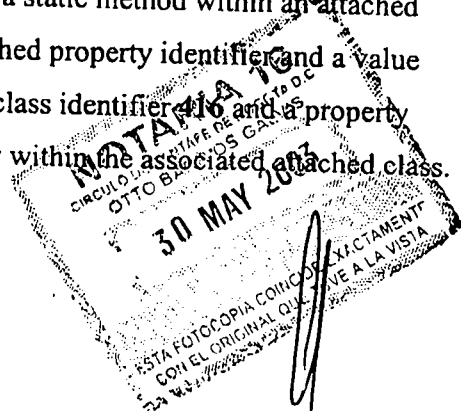
shown in programmatic syntax 401. Programmatic syntax 401 includes a class identifier 402 that identifies the class as containing an attached property.

Programmatic syntax 401 further includes a method identifier 404, a parameter list having a first identifier 406 for specifying a name for a property and a second identifier 408 for specifying a value associated with the property specified in the first identifier 406. The parameter list is enclosed within parenthesis and a period separates the class identifier 402 and method identifier 404. Referring to the attached class 322 shown in FIGURE 3, for attached property "FONT", the programmatic syntax 401 is as follows:

Attached Class.SetFont(PropertyID, value).

Interestingly, this syntax appears to look similar to syntaxes in previous programming models. However, the syntax is not identical and programmatic syntax 401 is different than prior syntaxes. For example, in programmatic syntax 401, method identifier 404 identifies a static method on a Provider class (i.e., attached class 322 in FIGURE 3). In prior programming models, method identifier would have identified an instance method on a Provider instance. Because the present invention utilizes the Provider class (i.e., the Attached Class 322 in FIGURE 3), the programming model can support extender properties in markup and property sheets because the lifetime of the Provider instance does not need to be tracked. Another feature of programmatic syntax 401 is that the syntax is strongly typed. As mentioned earlier, a strongly typed syntax enables good tool support because potential errors, such as misspelled names, incorrect data types, and the like, can be detected at compile time rather than at run-time.

In addition, for programming environments such as Visual Basic, C# or the like, the programming model also allows a loosely typed syntax (i.e., programmatic syntax 411). Programmatic syntax 411 includes an object identifier 412 identifying a target object, a method identifier 414 identifying a static method within an attached property class, a parameter list including an attached property identifier and a value 419. The attached property identifier includes a class identifier 416 and a property identifier 418 that identifies the attached property within the associated attached class.



5

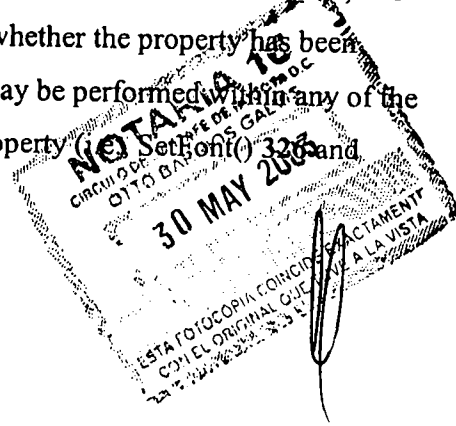
10

```
<Button xmlns:ap="expandoNameSpace" Pressed="false"
ap:Expandos.String="string"/>
```

20

25

13



161

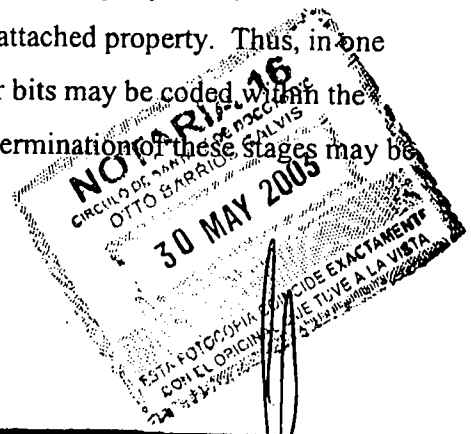
GetFont() 330). Because this check must be performed each time any of the static methods are called, this embodiment incurs a performance penalty. Those skilled in the art will appreciate that other embodiments that optimize the registration of the property may be implemented without departing from the present invention, such as requiring the application to perform the registration.

The registration of the attached property returns a unique PropertyID for the attached property. One illustrative call for registering a property may be in the form as follows:

```
BarDP = RegisterProperty("Bar",0,typeof(string),typeof(Button),0x3);.
```

In this embodiment, the BarDP variable will then contain the unique identifier for the attached property. This unique identifier is then used in the SetValue() and the GetValue() function calls. In one embodiment, the RegisterProperty is called, directly or indirectly, from the owner's static constructor. The owner refers to the class that attaches the attached property. Once the attached property is registered, any instance that matches the target type for the attached property may "implicitly" attach the property by calling the Get and Set static methods associated with the attached property.

As shown above in the exemplary call for registering a property, "Bar" designates a name for the attached property. The property engine typically does not use this name. However, a parser uses this name to determine whether a property with the same name as already been registered to the owner. The target type is Button. For attached properties, the owner and the target will not be the same. The default value is "0". As will be described in detail in conjunction with FIGURE 6, the property engine determines where to retrieve a value for the attached property. Therefore, the registerProperty call includes behavior bits that identify stages in which a value for the property is searched, such as inheritance, property sheets. In the above exemplary call, the behavior bits "0x3" indicates that a local value, a property sheet, and inheritance are searched in order to determine the value for the attached property. Thus, in one embodiment, the interpretation of these behavior bits may be coded within the GetValue() static method. Alternatively, the determination of these stages may be



provided by an expression that models relationships between properties in a formal manner as described in the commonly assigned co-pending application entitled "Extensible On-Demand Property System", Patent Application No. _____, filed on _____, which is hereby incorporated by reference.

In one embodiment, the code for an attached class may take the form shown in Table 1.

```
class BarProvider: Object {
    public static DependencyProperty BarDP = RegisterProperty(
        "Bar", 0, typeof(string), typeof(Button));

    public static string GetBar(Button button) {
        return (string) button.GetValue(BarDP); }

    public static void SetBar(Button button, string value) {
        button.SetValue(BarDP, value); }
}.
```

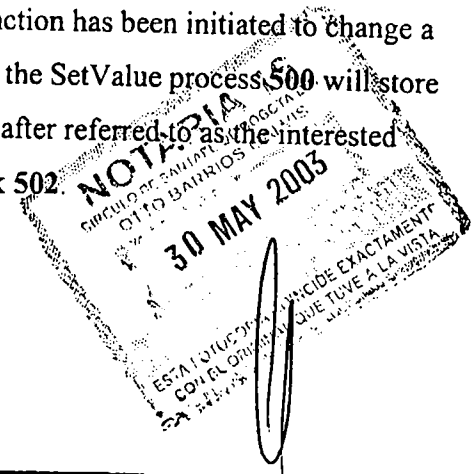
Table 1.

In the above code, the attached property named "Bar" is defined within attached class "BarProvider". The attached property name "Bar" is defined with a data type as string and can only be attached to Button objects. The static methods are strongly typed and allow access to the property and setting the property.

Once the attached property has been registered, the SetValue() and GetValue() functions may be performed. FIGURE 5 is a logical flow diagram illustrating a process for setting a value in accordance with the present invention. Processing begins at block 501, where the set function has been initiated to change a value associated with some property. In general, the SetValue process 500 will store the value locally for a property of interest (hereinafter referred to as the interested property). Processing continues at decision block 502.

property). Processing continues at decision block 502

15



At decision block 502, a determination is made whether local storage exists for the interested property. Because the present invention does not automatically have storage for each property for each instance of an object, if the interested property has not previously had a value stored for it, storage is allocated (block 504). Once
 5 storage has been allocated, processing continues to block 506, as it would have if it had detected local storage at decision block 502.

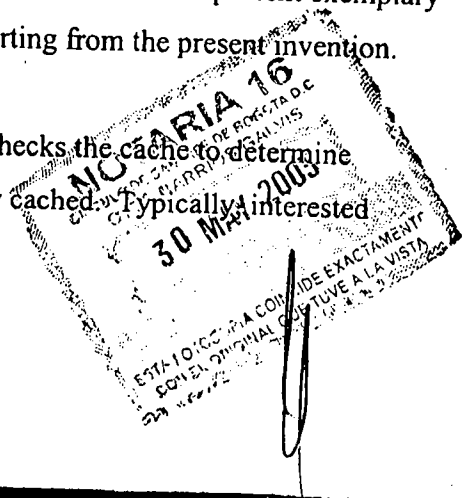
At block 506, the value is copied into the storage associated with the interested property. Once the interested property has changed states, notifications are sent of the change in state (block 508). In one embodiment, this notification may
 10 involve setting a dirty bit to indicate to dependent properties that their state may not be valid anymore. Alternatively, the notification may involve reporting to each dependent that a source has changed as described in the above-mentioned patent application. Processing continues at decision block 510.

At decision block 510, a determination is made whether a value was
 15 stored in the cache for the interested property. If there is not a value in the cache for this interested property, the process ends. However, if there is a value in the cache, the cache is cleared (block 512) because this value is no longer valid. The process then end.

FIGURE 6 is a logical flow diagram illustrating a process for retrieving a
 20 value in accordance with the present invention. Processing begins at block 601, where the query has been initiated. Processing continues at block 602.

At block 602, the application indicates that one of the properties of interest (hereinafter, referred to as interested property) needs updating. Typically, this may occur when an object is querying properties for their current state. The process
 25 600 determines a value for the property by checking various stages. These stages are defined when the property is registered. Decision blocks 606-610 represent exemplary stages, but other stages may be added without departing from the present invention. Processing continues at decision block 604.

At decision block 604, the process checks the cache to determine
 30 whether the interested property has been previously cached. Typically, interested



164

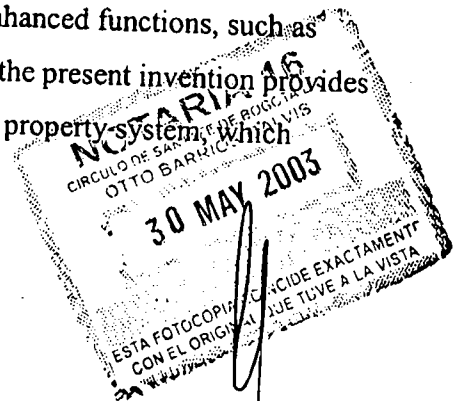
properties are cached in order to optimize retrieval. If the property has been previously cached, the process continues at block 612, where the value is retrieved from the cache. Alternatively, the process continues at decision block 606. At decision block 606, a determination is made whether the value for the interested property is local. The value will be local if a SetValue() has been performed to set a value locally for the property, as described in FIGURE 5. If the value is local, the process continues at block 612, where the value is retrieved from local storage. If the value is not local, the process continues at decision block 608.

At decision block 608, a determination is made whether the value for the interested property is available in a property sheet. When a base class has an associated property sheet, the property sheet specifies how each of the children of the base class will be rendered once instantiated. If the interesting property is available in a property sheet, processing continues at block 612, where the value is retrieved from the property sheet. However, if the value is not in the property sheet, processing continues at decision block 610.

At decision block 610, a determination is made whether the value for the interested property may be obtained through inheritance (i.e., an inherited value). Inherited values are derived from a parent object of the child object. If one of the parent objects has the interesting property, processing proceeds to block 612, where the value is retrieved from the parent object. However, if the value is not inherited, a default value from the attached class associated with the attached property is retrieved (block 614). Processing continues at block 616.

At block 616, the application calculates a weight metric based on the stage at which the value was received. The weight metric is stored and is used to make educated decisions on which interested properties to cache in order to optimize future retrievals. Processing then ends.

In addition, to the SetValue and GetValue processes described above, the programming model of the present invention provides enhanced functions, such as group queries or group notification. Thus, as described, the present invention provides a programming model that provides a dependency-based property system, which



165

supports complex relationships between properties through the use of attached classes. Applications built using this programming model scale very well because the property management technique does not require excessive local, per-instance based storage. Instead, the programming model promotes reuse of property values, such as through the use of property sheets and inheritance at the base class level. As described above, the present invention provides a property management mechanism whereby objects in the object hierarchy store property values through attaching.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

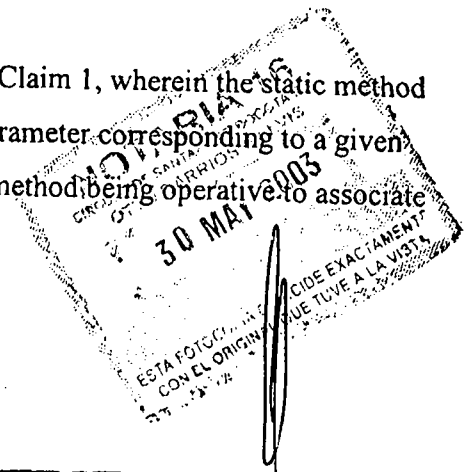
18



166

WE CLAIM:

1. A computer-readable medium including an object having a property in a first set of properties, the computer-readable medium further comprising:
a data structure including definitions for each of a second set of properties and including at least one static method associated with one property out of the second set of properties, the at least one static method having a first parameter that uniquely identifies the one property, the static method being operative to associate the one property with the object without specifying an explicit reference to the one property in the object.
2. The computer-readable medium of Claim 1, wherein the static method supports a strongly typed syntax.
3. The computer-readable medium of Claim 1, wherein the static method being operative to associate the one property with the object includes retrieving a value for the object without having the value stored locally on the object.
4. The computer-readable medium of Claim 3, wherein retrieving the value comprises determining which one of a plurality of stages holds the value associated with the one property.
5. The computer-readable medium of Claim 4, wherein one stage includes a parent object of the object.
6. The computer-readable medium of Claim 4, wherein one stage includes a property sheet.
7. The computer-readable medium of Claim 1, wherein the static method further includes a second parameter, the second parameter corresponding to a given value for the one property, and wherein the static method being operative to associate



the one property with the object comprises setting the given value in a local storage for the object.

8. The computer-readable medium of Claim 1, wherein the first set of properties corresponds to behaviors of the object, and the second set of properties corresponds to an appearance of the object.

9. A computer-readable medium having computer-executable components, comprising:

a base class including a plurality of properties; and

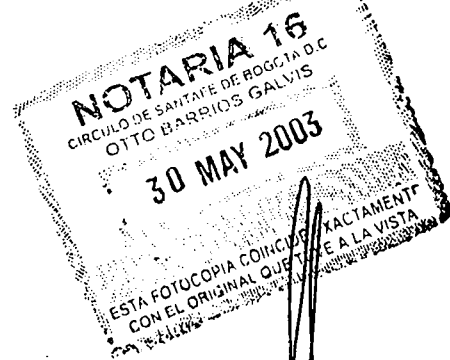
an attached class including a plurality of attached properties, each of the attached properties capable of being associated with an instance of an object derived from the base class, the attached class further including a static method for associating one of the plurality of attached properties with the instance of the object.

10. The computer-readable medium of Claim 9, wherein the object derived from the base class inherits at least one method from a node class, the one method being called by the static method when associating the one attached property with the instance of the object.

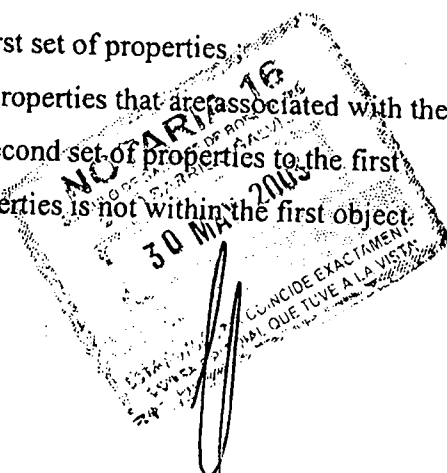
11. The computer-readable medium of Claim 10, wherein the static method and the one method each include a first parameter for passing a unique identifier for the one attached property that is being associated with the instance of the object.

12. The computer-readable medium of Claim 11, wherein the unique identifier is available after registering the one attached property at run-time.

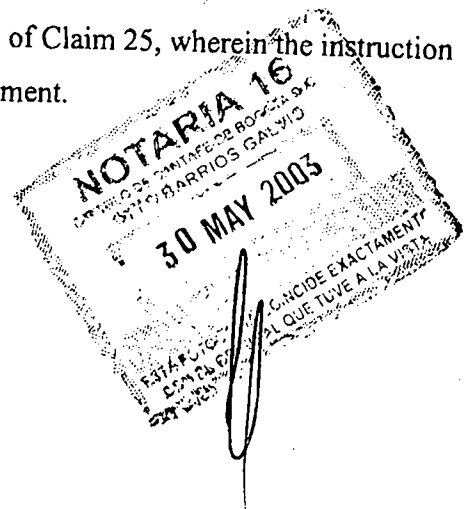
13. The computer-readable medium of Claim 10, wherein the one method supports a loosely typed syntax.



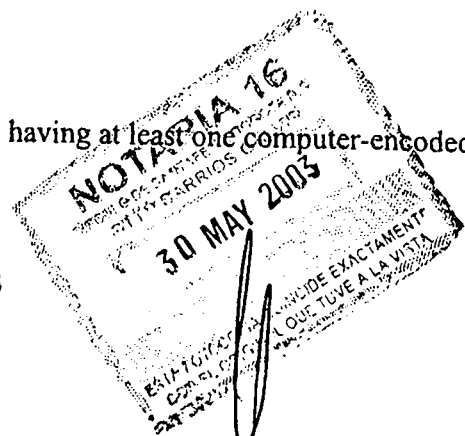
14. The computer-readable medium of Claim 9, wherein the static method supports a strongly typed syntax.
15. The computer-readable medium of Claim 9, wherein associating the one attached property comprises retrieving a value for the object from a parent object of the object.
16. The computer-readable medium of Claim 9, wherein associating the one attached property comprises retrieving a value for the object from a property sheet.
17. A computer system that uses objects, comprising:
a first object derived from a first class, the first object including a first set of properties;
a second class providing a second set of properties to the first object, the second class including a static method operative to attach one of the second set of properties to the first object in response to an operation associated with the first object.
18. The computer system of Claim 17, wherein the static method includes a first parameter for uniquely identifying which one property out of the second set of properties the operation is requesting.
19. The computer system of Claim 18, wherein the one property is registered before the first parameter can uniquely identify the one property.
20. A computer-implemented method for setting a state for a property, comprising:
declaring a first object having a first set of properties;
registering one or more dynamic properties that are associated with the first object, the dynamic properties providing a second set of properties to the first object, wherein storage for the second set of properties is not within the first object.



21. The computer-implemented method of Claim 20, wherein registering the one or more dynamic properties includes assigning a unique identifier to each of the one or more dynamic properties.
22. The computer-implemented method of Claim 21, further comprising calling a static method having a first parameter, the first parameter referring to the unique identifier.
23. The computer-implemented method of Claim 22, wherein the static method is strongly typed.
24. The computer-implemented method of Claim 20, wherein the first object has one or more associated child objects, each child object thereby being associated with the second set of properties.
25. A computer-readable medium having at least one computer executable instruction, comprising:
an instruction means associating an attached property with an instance of an object derived from another class through a static method in an attached property class.
26. The computer-readable medium of Claim 25, wherein the instruction means supports a strongly typed syntax for a programmatic language.
27. The computer-readable medium of Claim 25, wherein the instruction means supports loosely-type syntax for a programming language.
28. The computer-readable medium of Claim 25, wherein the instruction means comprises a statement in a markup document.



29. The computer-readable medium of Claim 25, wherein the instruction means comprises a statement in a property sheet.
30. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:
- a class identifier identifying an attached property class;
 - a method identifier identifying a method in the attached property class for affecting an attached property associated with a target object;
 - a parameter list, comprising:
 - a property identifier identifying the attached property.
31. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:
- an object identifier identifying a target object;
 - a method identifier identifying a method in an attached property class for affecting an attached property associated with the target object;
 - a parameter list, comprising:
 - an attached property identifier including a class identifier identifying the attached property class and a property identifier identifying a name for the attached property within the attached property class.
32. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:
- a tag representing a target object;
 - an attached property identifier having a provider designator identifying an attached property class and a property identifier identifying a name for the attached property within the attached property class.
33. A computer-readable medium having at least one computer-encoded instruction, the instruction comprising:



010
271

a XML schema for defining a target object, an attached property class, and an attached property within the attached property class, in such a manner that the attached property becomes associated with the target object without an explicit reference to the attached property in the target object.



Handwritten signature

Abstract

Described is a mechanism for allowing new functionality for an object to be expressed as a property that is not built into the class from which the object derives. More specifically, the mechanism associates properties in one class with another class. A computer-readable medium, that includes an object having a property in a first set of properties, further includes a data structure. The data structure includes definitions for each of a second set of properties and includes at least one static method. The static method is associated with one property out of the second set of properties and includes a first parameter. The first parameter uniquely identifies the one property. The static method is operative to associate the one property with the object without specifying an explicit reference to the one property in the object. The property is registered during run-time in order to receive the unique identifier.



PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY

ATTORNEY DOCKET NO. 5003796US01

MS DOCKET NO. 196295.1

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name:

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: **SYSTEM AND METHOD FOR ASSOCIATING PROPERTIES WITH OBJECTS** the specification of which is attached hereto unless the following box is checked:

☐ which was filed on _____ as US Application Serial No. or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of an foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C.
			Yes: ☐ No: ☐

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION NUMBER	FILING DATE

U.S. Priority Claim

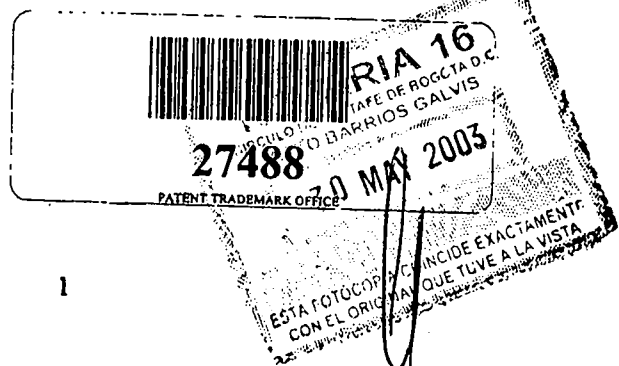
I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected herewith:

Customer No. 27488

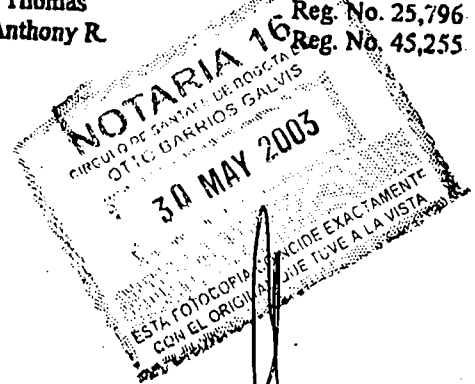


Albrecht, John W.
 Ali, M. Jeffer
 Altera, Allan G.
 Anderson, Gregg L.
 Batzli, Brian H.
 Beard, John L.
 Berns, John M.
 Branch, John W.
 Bremer, Dennis C.
 Brown, Jeffrey C.
 Bruess, Steven C.
 Byrne, Linda M.
 Campbell, Keith
 Carlson, Alan G.
 Caspers, Philip P.
 Clifford, John A.
 Daignault, Ronald A.
 Daley, Dennis R.
 Dalglish, Leslie E.
 Daulton, Julie R.
 DeVries Smith, Katherine M.
 DiPietro, Mark J.
 Doscotch, Matthew A.
 Edell, Robert T.
 Epp Ryan, Sandra
 Glance, Robert J.
 Goggin, Matthew J.
 Golla, Charles E.
 Gorman, Alan G.
 Gould, John D.
 Gregson, Richard
 Gresens, John J.
 Hamer, Samuel A.
 Hamre, Curtis B.
 Harrison, Kevin C.
 Hertzberg, Brett A.
 Hillson, Randall A.
 Holzer, Jr., Richard J.
 Hope, Leonard J.
 Jardine, John S.
 Johnston, Scott W.
 Kadievitch, Natalie D.
 Kaseburg, Frederick A.
 Kettelberger, Denise
 Keys, Jeramie J.
 Knearl, Homer L.
 Kowalchyk, Alan W.
 Kowalchyk, Katherine M.
 Lacy, Paul E.

Reg. No. 40,481
 Reg. No. 46,359
 Reg. No. 40,274
 Reg. No. 28,828
 Reg. No. 32,960
 Reg. No. 27,612
 Reg. No. 43,496
 Reg. No. 41,633
 Reg. No. 40,528
 Reg. No. 41,643
 Reg. No. 34,130
 Reg. No. 32,404
 Reg. No. 46,597
 Reg. No. 25,959
 Reg. No. 33,227
 Reg. No. 30,247
 Reg. No. 25,968
 Reg. No. 34,994
 Reg. No. 40,579
 Reg. No. 36,414
 Reg. No. 42,157
 Reg. No. 28,707
 Reg. No. 48,957
 Reg. No. 20,187
 Reg. No. 39,667
 Reg. No. 40,620
 Reg. No. 44,125
 Reg. No. 26,896
 Reg. No. 38,472
 Reg. No. 18,223
 Reg. No. 41,804
 Reg. No. 33,112
 Reg. No. 46,754
 Reg. No. 29,165
 Reg. No. 46,759
 Reg. No. 42,660
 Reg. No. 31,838
 Reg. No. 42,668
 Reg. No. 44,774
 Reg. No. 48,835
 Reg. No. 39,721
 Reg. No. 34,196
 Reg. No. 47,695
 Reg. No. 33,924
 Reg. No. 42,724
 Reg. No. 21,197
 Reg. No. 31,535
 Reg. No. 36,848
 Reg. No. 38,946

Larson, James A.
 Leonard, Christopher J.
 Liepa, Mara E.
 Lindquist, Timothy A.
 Lown, Jean A.
 Mayfield, Denise L.
 McDonald, Daniel W.
 McIntyre, Jr., William F.
 Mitchem, M. Todd
 Mueller, Douglas P.
 Nelson, Anna M.
 Parsons, Nancy J.
 Pauly, Daniel M.
 Phillips, John B.
 Pino, Mark J.
 Prendergast, Paul
 Pytel, Melissa J.
 Qualey, Terry
 Reich, John C.
 Reiland, Earl D.
 Roberts, Fred
 Samuels, Lisa A.
 Schmalz, David G.
 Schuman, Mark D.
 Schumann, Michael D.
 Scull, Timothy B.
 Sebald, Gregory A.
 Skoog, Mark T.
 Spellman, Steven J.
 Stoli-DeBell, Kirstin L.
 Sullivan, Timothy
 Sumner, John P.
 Swenson, Erik G.
 Tellekson, David K.
 Trembath, Jon R.
 Tunheim, Marcia A.
 Underhill, Albert L.
 Vandenburg, J. Derek
 Wahl, John R.
 Weaver, Karrie G.
 Welter, Paul A.
 Whipps, Brian
 Whitaker, John E.
 Williams, Douglas J.
 Withers, James D.
 Witt, Jonelle
 Wu, Tong
 Young, Thomas
 Zeuli, Anthony R.

Reg. No. 40,443
 Reg. No. 41,940
 Reg. No. 40,066
 Reg. No. 40,701
 Reg. No. 48,428
 Reg. No. 33,732
 Reg. No. 32,044
 Reg. No. 44,921
 Reg. No. 40,731
 Reg. No. 30,300
 Reg. No. 48,935
 Reg. No. 40,364
 Reg. No. 40,123
 Reg. No. 37,206
 Reg. No. 43,858
 Reg. No. 46,068
 Reg. No. 41,512
 Reg. No. 25,148
 Reg. No. 37,703
 Reg. No. 25,767
 Reg. No. 34,707
 Reg. No. 43,080
 Reg. No. 39,828
 Reg. No. 31,197
 Reg. No. 30,422
 Reg. No. 42,137
 Reg. No. 33,280
 Reg. No. 40,178
 Reg. No. 45,124
 Reg. No. 43,164
 Reg. No. 47,981
 Reg. No. 29,114
 Reg. No. 45,147
 Reg. No. 32,314
 Reg. No. 38,344
 Reg. No. 42,189
 Reg. No. 27,403
 Reg. No. 32,179
 Reg. No. 33,044
 Reg. No. 43,245
 Reg. No. 20,890
 Reg. No. 43,261
 Reg. No. 42,222
 Reg. No. 27,054
 Reg. No. 40,376
 Reg. No. 41,980
 Reg. No. 43,361
 Reg. No. 25,796
 Reg. No. 45,255



In addition, I also hereby appoint the following attorneys to prosecute this application and to transact all business in the U.S. Patent and Trademark Office in connection therewith:

Kate E. Sako, Reg. No. 32,628
Daniel D. Crouse, Reg. No. 32,022

Send Correspondence to: Marcia A. Tunhelm; Reg. No. 42,189 Merchant & Gould P.C. P.O. Box 2903 Minneapolis, MN 55402-0903	Direct Telephone Calls To: Contact: Marcia A. Tunhelm (206) 342.6259
--	---

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like are so made punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Jeffrey L. Bogdan

Citizenship: US

Residence: Redmond, WA

Post Office Address: 25936 NE 29th Place, Redmond, WA 98053

[Signature]
Inventor's Signature

6-28-2002
Date

Full Name of Inventor: Mark J. Finocchio

Citizenship: US

Residence: Redmond, WA

Post Office Address: 4631 168th Lt. NE, Redmond, WA 98052

[Signature]
Inventor's Signature

6-28-02
Date

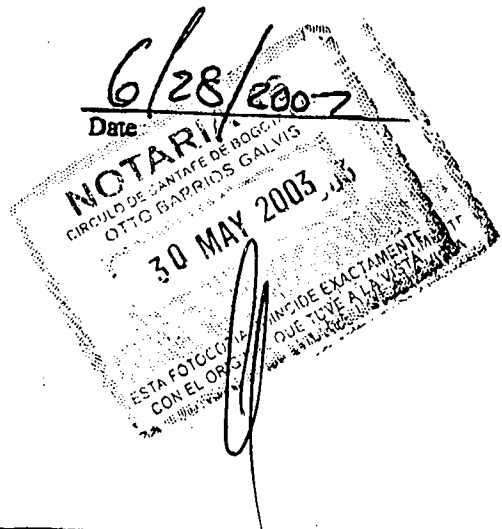
Full Name of Inventor: Nicholas M. Kramer

Citizenship: US

Residence: Seattle, WA 98103

Post Office Address: 3728 Wallingford Ave., Seattle, WA 98103

[Signature]
Inventor's Signature



1/6

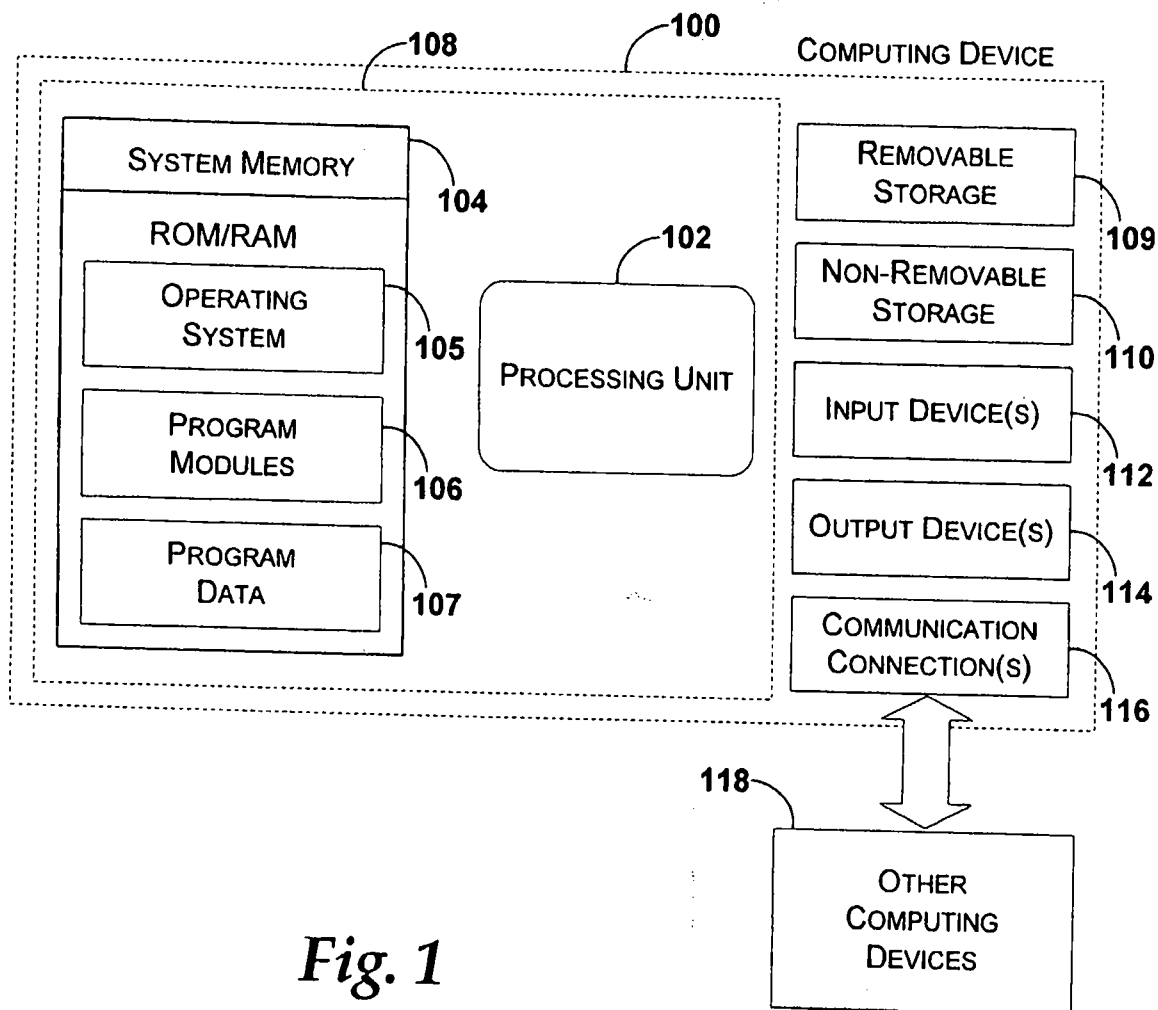
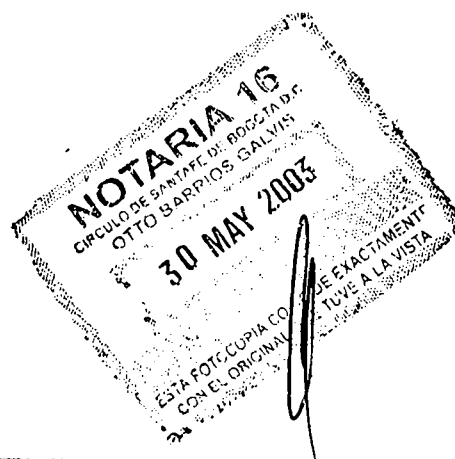


Fig. 1



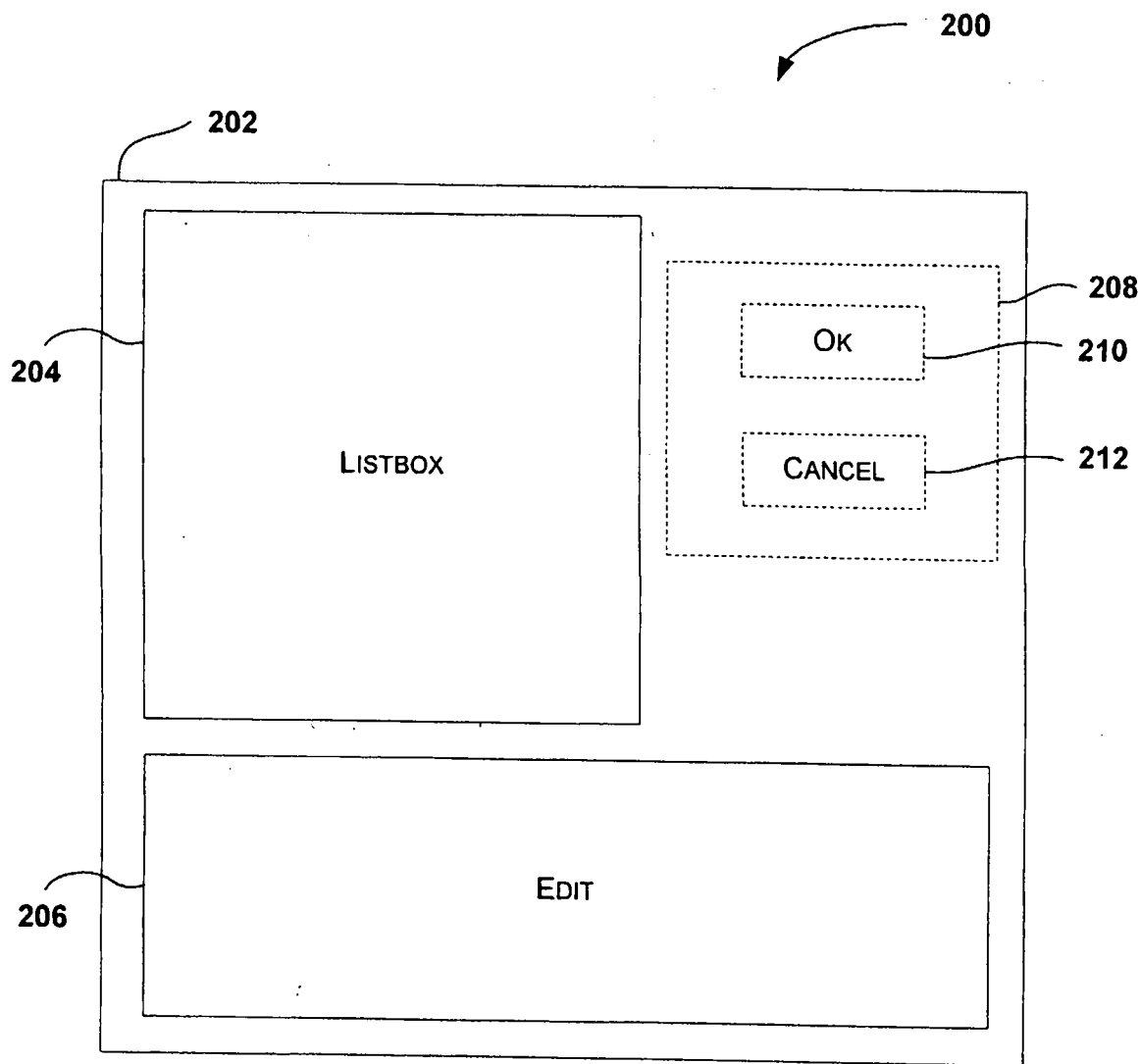


Fig. 2



3/6

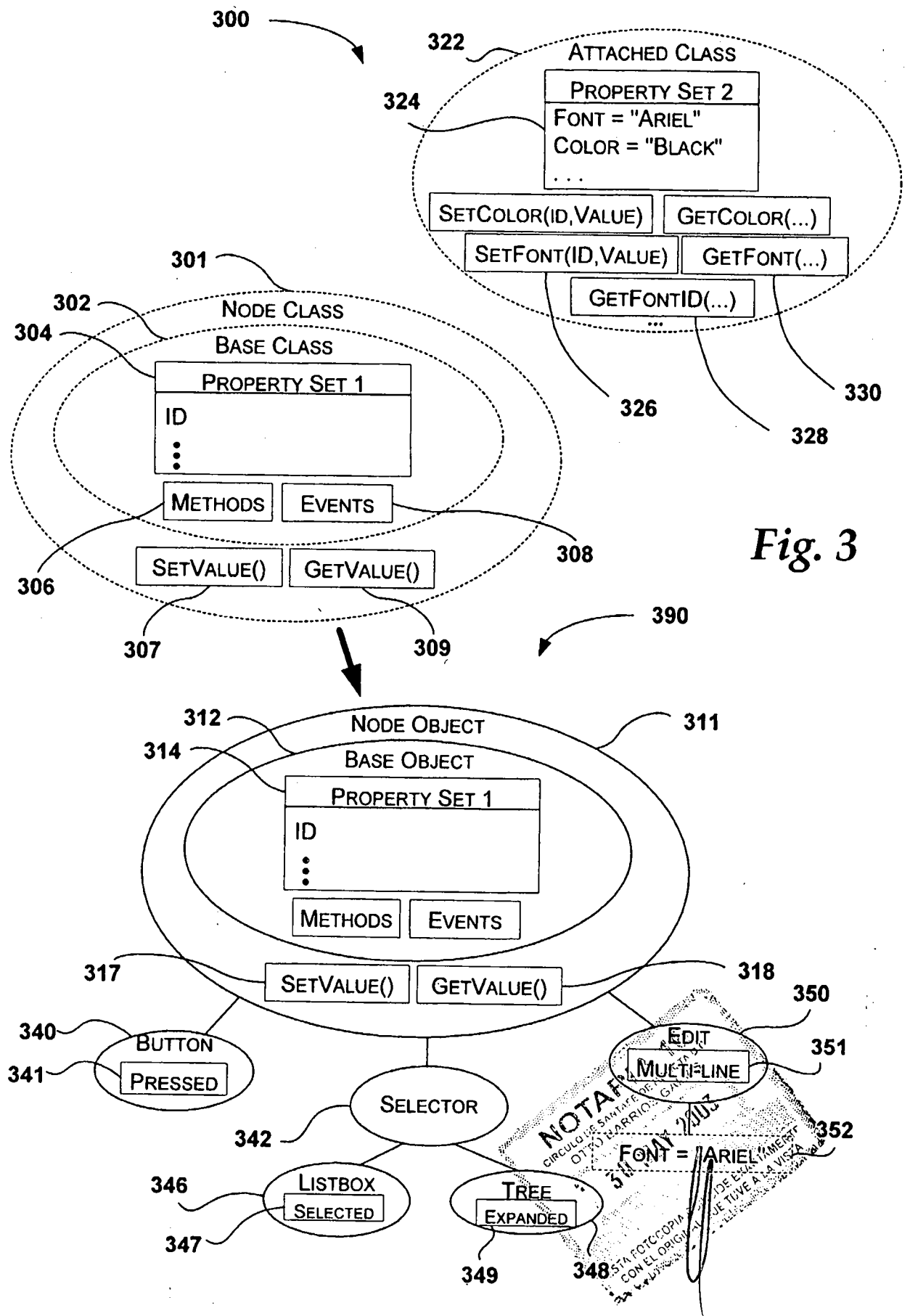


Fig. 3

179

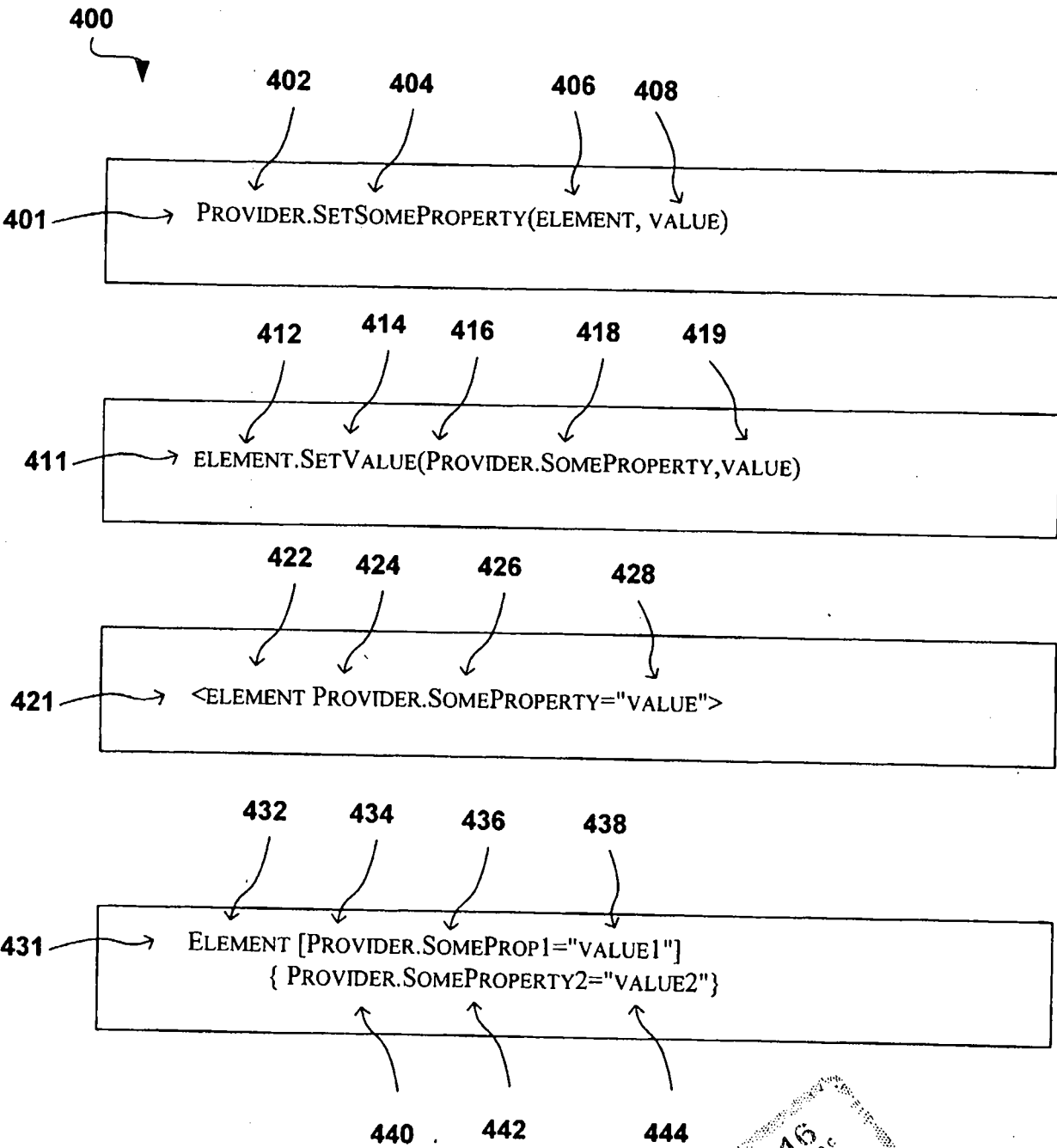
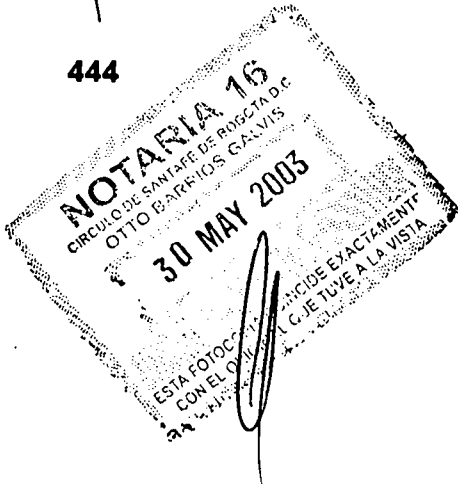


Fig. 4



5/6

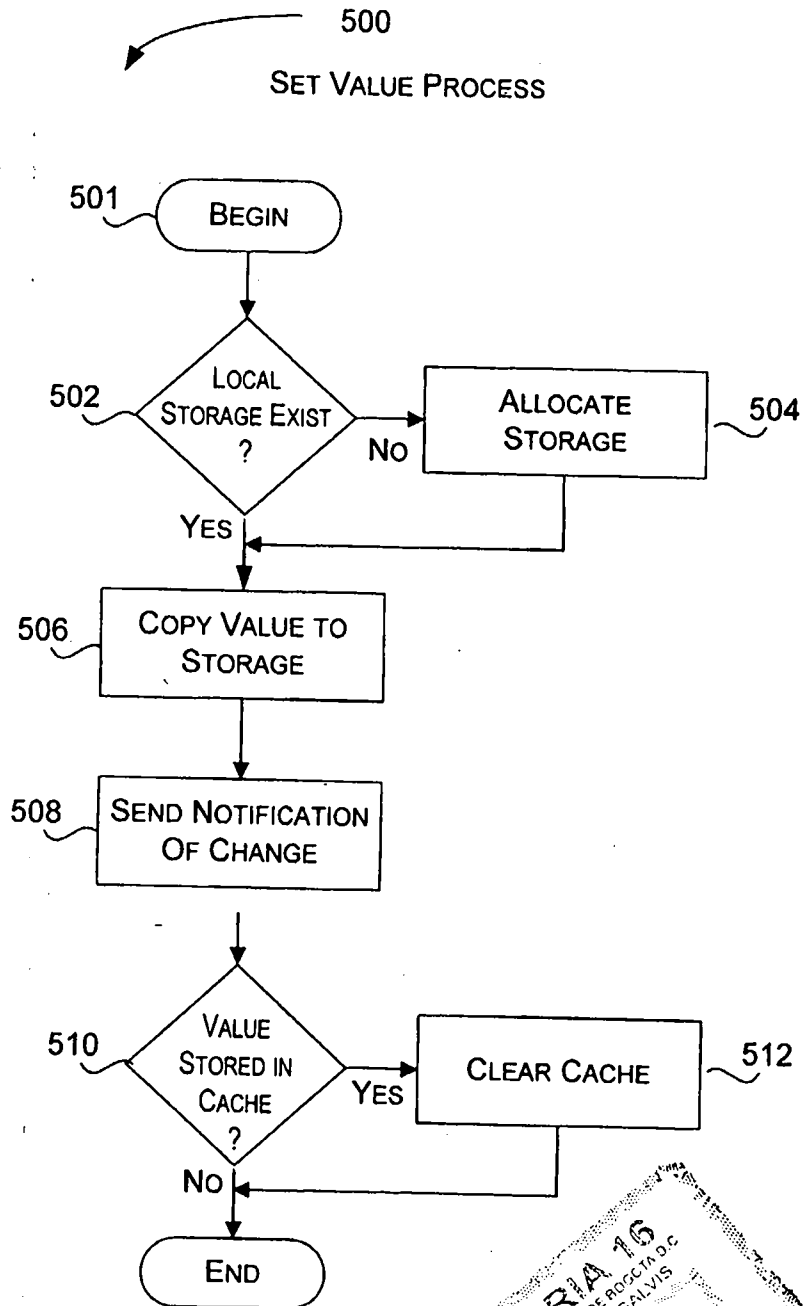
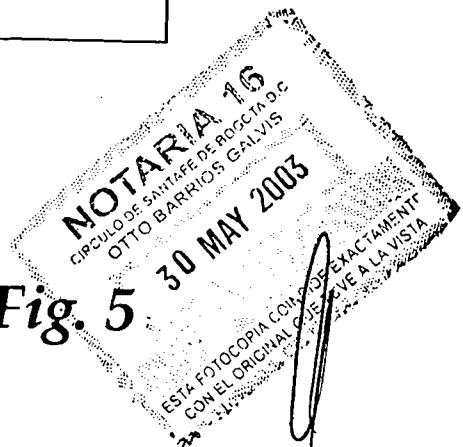


Fig. 5



6/6

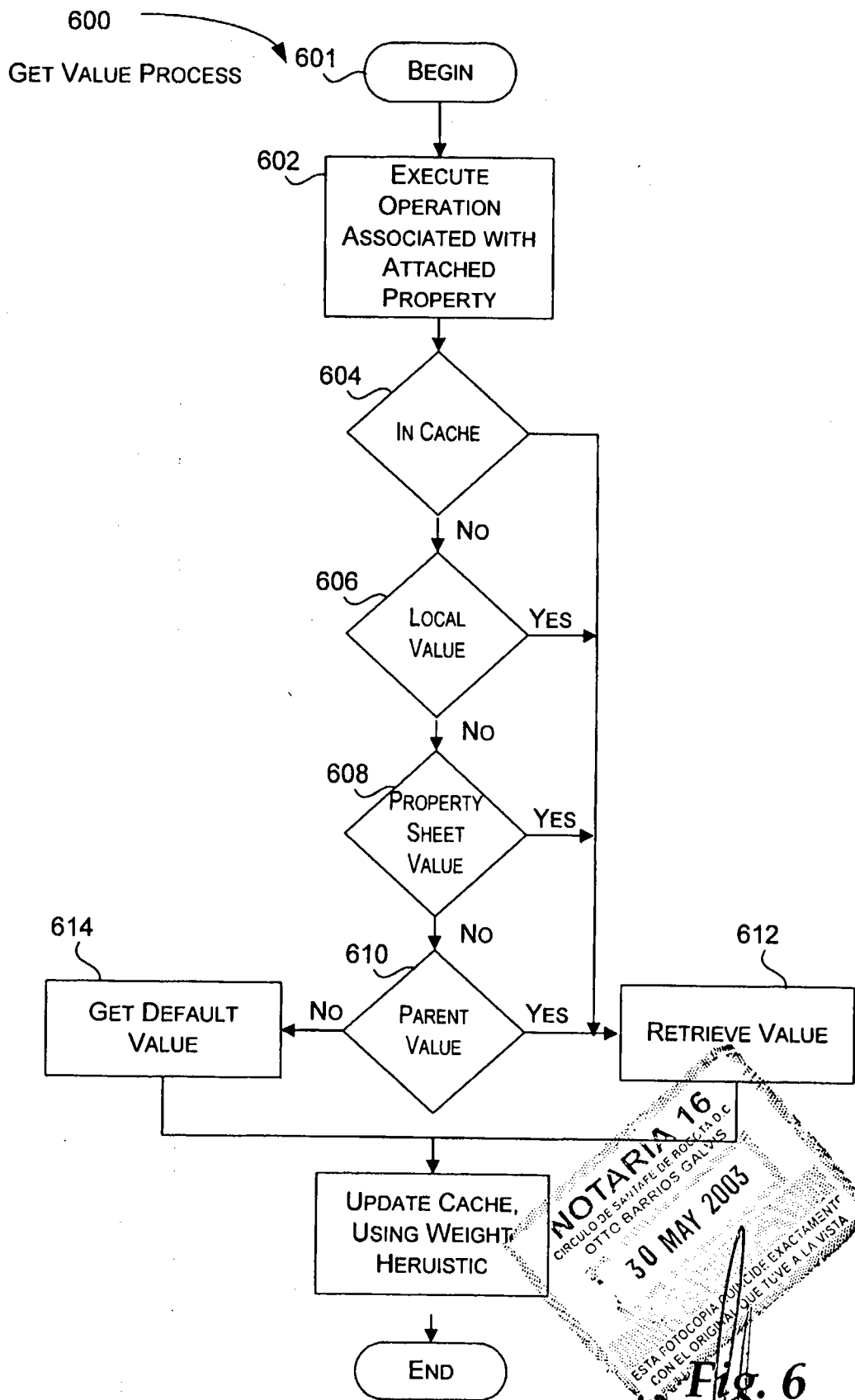


Fig. 6

[Handwritten signature]
182

De un documento escrito en inglés, la cual para su identificación lleva el sello de BEATRIZ EUGENIA ALARCÓN GUTIÉRREZ, Traductora/Intérprete Oficial, según Resolución 2512 del 15 de septiembre de 1993 del Ministerio de Justicia.



[Handwritten signature]
BEATRIZ EUGENIA ALARCÓN GUTIÉRREZ
TRADUCTORA E INTÉRPRETE OFICIAL
RESOLUCIÓN 2512 DE SEPT. 15/93 MINISTERIO DE JUSTICIA

183



LOS ESTADOS UNIDOS DE AMERICA

A TODO AQUEL A QUIEN CONCIERNA LO PRESENTE
DEPARTAMENTO DE COMERCIO DE LOS ESTADOS UNIDOS
Oficina de Patentes y Marcas de los Estados Unidos

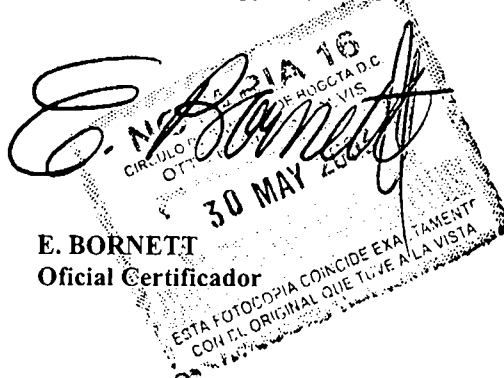
Febrero 25, 2003

LA PRESENTE ES PARA CERTIFICAR QUE ADJUNTA A LA MISMA HAY
UNA COPIA FIEL DE LOS REGISTROS DE LA OFICINA DE PATENTES Y
MARCAS DE LOS ESTADOS UNIDOS DE LOS DOCUMENTOS DE LA
SOLICITUD DE PATENTE IDENTIFICADA EN LO QUE SIGUE, QUE
CUMPLEN LOS REQUISITOS PARA OTORGARLE UNA FECHA DE
PRESENTACIÓN BAJO 35 USC 111.

NÚMERO DE SOLICITUD: 10/187,012
FECHA DE PRESENTACIÓN: Junio 28, 2002



Por la Autoridad del
COMISARIO DE PATENTES Y MARCAS



E. BORNETT
Oficial Certificador

184

07-02-02

EN LA OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

Solicitante: Jeffrey L. Bogdan et al.
Registro: 50037.0096US01
Título: SISTEMA Y MÉTODO PARA ASOCIAR
PROPIEDADES CON OBJETOS

BAJO EL CERTIFICADO CFR 37 1 10
Número de etiqueta de correo "Correo Expreso": ET886907746
Fecha de depósito Junio 28, 2002
Por medio de la presente certifico que este documento o tasa
está siendo depositada con el Servicio Postal de los Estados
Unidos "Oficina Postal Correo Expreso al Destinatario"
conforme al servicio CFR 37 1.10. y se destina al Comisario
para Patentes, Washington, DC 20231

Por: (Aparece firma)
Nombre: Lucy Arevalo

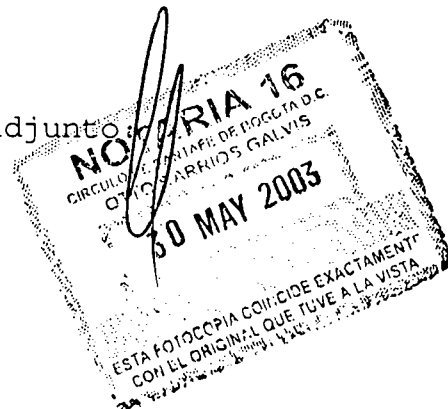
CASILLA DE SOLICITUD DE PATENTE

Comisario de Patentes

Washington, D.C. 20231

Señor:

Estamos transmitiendo junto con esto lo adjunto:



185

☒ Hoja de transmisión, en duplicado, contiene
Certificado conforme al CFR 37 1.10.

☒ Utilidad de la Solicitud de Patente: Descripción: 18
páginas; Reivindicaciones 33; Resumen 1 página.

La tasa ha sido calculada como se muestra en
adelante en la tabla "Reivindicaciones Presentadas"

☒ 6 hojas de dibujos

☒ Una Declaración Combinada y Poder de Abogado firmado

☒ Asignación de la invención a Microsoft Corporation,
Registro de Portada

☒ Un cheque en la cantidad de \$1478,00 para cubrir la
tasa de presentación

☒ Un Cheque por \$40,00 para cubrir la tasa de registro
de asignación

☒ Hoja de datos de solicitud, 4 páginas

☒ Estampilla de retorno



186

REIVINDICACIONES PRESENTADAS

Número de reivindicaciones presentadas		En exceso de:		Número extra		Proporción		Tasa
Tasa de Presentación Básica								\$740,00
Total Reivindicaciones								
33	-	20	=	13	X	18,00	=	\$234,00
Reivindicaciones Independientes								
9	-	3	=	6	X	84,00	=	\$504,00
TASAS DE MÚLTIPLES REIVINDICACIONES DEPENDIENTES								\$0,00
TOTAL TASA DE PRESENTACIÓN								\$1478,00

Por favor cargue cualquier tasa adicional o crédito de sobrepago a la cuenta de depósito No. 13-2725. Un duplicado de esta hoja se incluye.

MERCHANT & GOULD P.C.

P.O. Box 2903, Minneapolis, MN 55402-0903

(206) 342-6200

Por: (aparece firma)

Nombre: Marcia A. Tunheim



RECEIVED OFFICE MARCIA TUNHEIM
2003 MAY 30 10 10 AM
MINNEAPOLIS, MN 55402-0903

187

Reg. No.: 42,189

Iniciales: MAT/la



(TRANSMISIÓN PTO - NUEVA PRESENTACIÓN)



188

Hoja de Datos de Solicitud

Información de Solicitud

Tipo de Solicitud:: Regular

Materia del Asunto:: Utilidad

Clasificación Sugerida::

Unidad Técnica de Grupo Sugerida::

CD-ROM o CD_R?:: Ninguno

Número de Discos de CD::

Número de Copias de CDs::

Secuencia de Presentación:: No

Forma legible en Computador (CRF)?:: No

Título:: SISTEMA Y MÉTODO PARA ASOCIAR
PROPIEDADES CON OBJETOS

Número de Registro de Abogado:: 50037.0096US01

Solicitud para Publicación Temprana:: No

Solicitud para No Publicación:: No

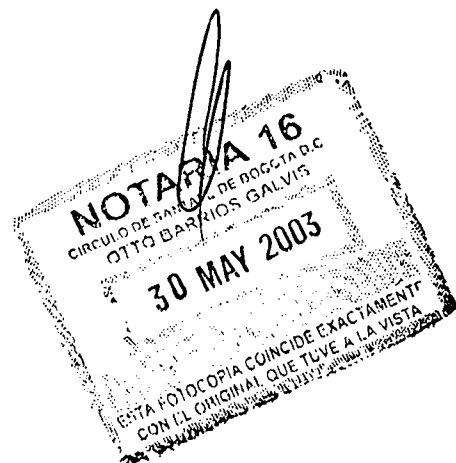
Figura de Dibujo Sugerida::

Total de hojas de Dibujo:: 6

Entidad Pequeña::

Nombre en Latín::

Nombre de Denominación de Variedad:



Petición Incluida:: No
Tipo de Petición::
Agencia Gubernamental Estadounidense Licenciada::
Números de Contrato u Otorgamiento::
Orden de Reserva en Solicitud Padre:: No

Información del Solicitante

Tipo de Autoridad Solicitante:: Inventor
País de Ciudadanía primaria:: USA
Estado:: Capacidad Llena
Primer Nombre:: Jeffrey
Segundo Nombre:: L.
Primer Apellido:: Bodgan
Segundo Apellido::
Ciudad de Residencia:: Redmond
Estado o Provincia de Residencia:: WA
País de Residencia:: USA
Dirección de Residencia:: 25936 NE 29th Place
Ciudad de Residencia:: Redmond
Estado o Provincia de Dirección de Residencia::
WA
País de Dirección de Residencia:: USA



~~190~~
190

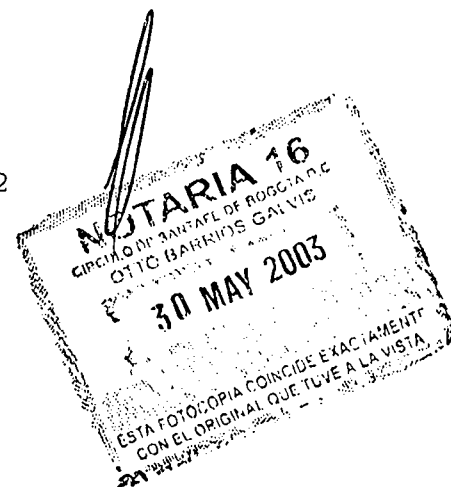
Dirección de Correo Postal o Código Zip::

98053

Información del Solicitante

Tipo de Autoridad Solicitante:: Inventor
País de Ciudadanía Primaria:: USA
Estado:: Capacidad Llena
Primer Nombre:: Mark
Segundo Nombre:: J.
Primer apellido:: Finocchio
Segundo apellido::
Ciudad de Residencia:: Redmond
Estado o Provincia de Residencia:: WA
País de Residencia:: USA
Dirección de Residencia:: 4631 168th Lt. NE
Ciudad de Residencia:: Redmond
Estado o Provincia de Dirección de Residencia::
WA
País de Dirección de Residencia:: USA
Dirección de Correo Postal o Código Zip::

98052



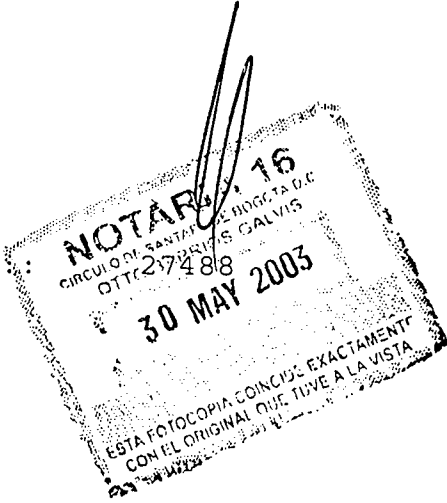
191

Información del Solicitante

Tipo de Autoridad Solicitante:: Inventor
País de Ciudadanía Primaria:: USA
Estado:: Capacidad Llena
Primer Nombre:: Nicholas
Segundo Nombre:: M.
Primer apellido:: Kramer
Segundo apellido::
Ciudad de Residencia:: Seattle
Estado o Provincia de Residencia:: WA
País de Residencia:: USA
Dirección de Residencia:: 3728 Wallingford Ave.
Ciudad de Residencia:: Seattle
Estado o Provincia de Dirección de Residencia::
WA
País de Dirección de Residencia:: USA
Dirección de Correo Postal o Código Zip::
98103

Información de Correspondencia

Número de Correspondencia del Cliente:



192

Información Representativa

Número Representativo del Cliente	27488
-----------------------------------	-------

Información del Cesionario

Nombre del Cesionario:: Microsoft Corporation

Dirección de Residencia:: One Microsoft Way

Ciudad de Residencia:: Redmond

Estado o Provincia de Residencia:: WA

País de Residencia:: USA

Dirección de Correo Postal o Código Zip:: 98052



M&G No. 50037.0096US01/196295.1

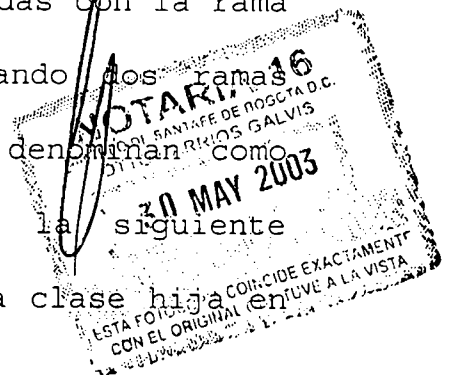
SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

CAMPO DE LA INVENCION

La presente invención se relaciona de manera general con aplicaciones de soporte lógico, y más particularmente, con mecanismos para manejar propiedades de objetos dentro de una aplicación de soporte lógico.

ANTECEDENTES DE LA INVENCION

La mayoría de los modelos de programación actuales soportan el concepto de clases. Estas clases son típicamente estructuradas en un árbol jerárquico que tiene ramas que representan diferentes clases en la clase jerárquica. Cuando dos ramas están en diferentes niveles, la rama inferior representa la clase hija. La clase hija hereda la información de las clases asociadas con la rama superior (es decir la clase padre). Cuando dos ramas están en el mismo nivel, las clases se denominan como clases hermanas. Para el propósito de la siguiente discusión, cuando se hace referencia a una clase hija en



194

relación a una clase padre, los términos, clase inferior y clase superior, se pueden utilizar para referirse a la clase hija y a la clase padre, respectivamente, la rama más superior en la jerarquía representa una clase base en el árbol de la clase jerárquica. Típicamente, la información en cada clase incluye propiedades, métodos, y eventos. Las propiedades describen características asociadas con la clase. Por ejemplo, una clase botón puede tener propiedades tales como amplitud, color de trasfondo, tipo de letra, visible, y presionada. Cuando una de las clases se inicia, se crea un objeto de esa clase. Cada propiedad en el objeto tiene un valor asociado, que puede ser requerido y establecido durante una operación del tiempo de ejecución. El valor que es requerido o establecido se puede esperar para que se adecue a un tipo de datos específico si la sintaxis es fuertemente digitada. Teniendo sintaxis que es fuertemente digitada es deseable entonces que los errores se puedan detectar dentro de la aplicación del software antes de la operación del tiempo de ejecución.

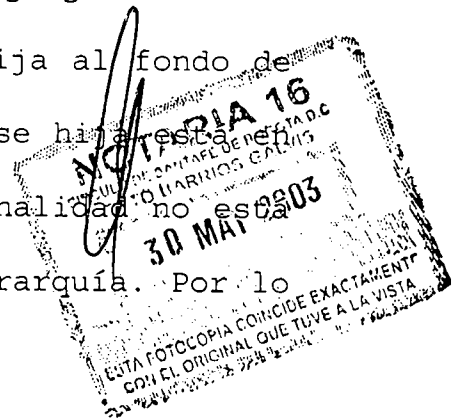
Una vez la clase jerárquica está en su lugar 16 agregar nueva funcionalidad a objetos dentro de la clase jerárquica representa un problema. En un modelo de



195

programación, la nueva funcionalidad es forzada dentro de la clase base. Cuando se hace esto, la clase base se hace muy grande (por ejemplo 100 métodos, 50 propiedades y 20 eventos), que dan como resultado una jerarquía de objeto casi inmanejable. Un resultado indeseable de este modelo de programación es que el número de propiedades, métodos, y eventos dentro de la clase base se hace opresiva para que los desarrolladores la entiendan completamente antes de implementar las características deseadas en los objetos que crean ellos. Aún otro resultado indeseable de este modelo de programación es que los requisitos de memoria se hacen enormes debido al hecho de que los valores para las propiedades son almacenados localmente. En razón a que los valores son almacenados localmente, las aplicaciones creadas dentro de este modelo de programación no se escalan bien.

Este modelo de programación también origina otros problemas a terceras partes desarrolladoras. Las terceras partes desarrolladoras que desean agregar nuevas funcionalidades deben agregar una clase hija al fondo de la jerarquía. En razón a que la nueva clase hija se agrega al fondo de la jerarquía, la nueva funcionalidad no es disponible a otras clases dentro de la jerarquía. Por lo



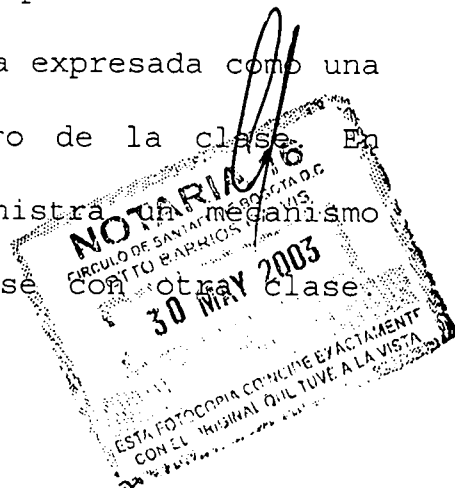
196

tanto, las terceras partes desarrolladoras podrían necesitar agregar nueva funcionalidad a varias nuevas clases hija. Como uno puede imaginar, esto da como resultado una duplicación de código, que impacta la mantenibilidad de la jerarquía de objeto. Por las anteriores razones, el modelo de programación no es muy deseable.

Hasta la presente invención, un modelo de programación que permitía que se suministrara nueva funcionalidad a una clase dentro de una jerarquía de clase existente sin los inconvenientes anteriores ha eludido aquellos expertos en la técnica.

RESUMEN DE LA INVENCION

La presente invención suministra un mecanismo que permite que se suministre nueva funcionalidad a una clase sin tener que volverse la nueva funcionalidad parte permanente de la clase. Además, la presente invención permite que la nueva funcionalidad sea expresada como una propiedad que no se construye dentro de la clase. En general, la presente invención suministra un mecanismo para asociar propiedades en una clase con otra clase.

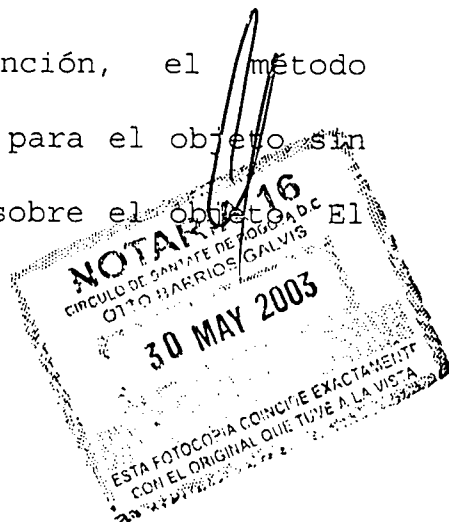


Este asocio es fácilmente modificable de tal forma que otros conjuntos de propiedades se pueden asociar con la clase.

En una modalidad, un medio legible por computadora que incluye un objeto que tiene una propiedad en el primer conjunto de propiedades, incluye además una estructura de datos. La estructura de datos incluye definiciones para cada una de un segundo conjunto de propiedades e incluye al menos un método estadístico. El método estadístico se asocia con una propiedad además del segundo conjunto de propiedades e incluye un primer parámetro. El primer parámetro identifica únicamente una propiedad. El método estadístico es operativo para asociar una propiedad con el objeto sin especificar una referencia explícita a una propiedad en el objeto.

En un aspecto de la invención, el método estadístico soporta una sintaxis fuertemente digitada.

En otro aspecto de la invención, el método estadístico incluye retornar un valor para el objeto sin tener el valor almacenado localmente sobre el objeto. El

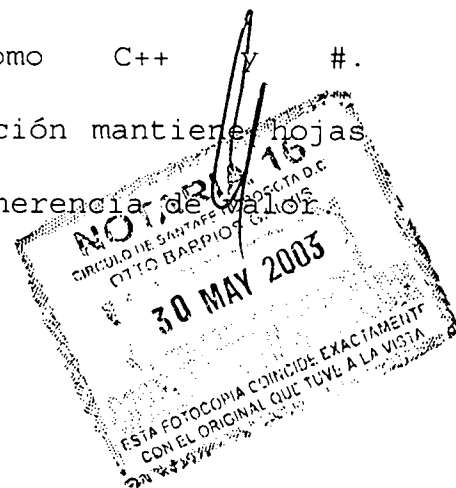


valor se puede regresar desde varias etapas, tales como un objeto padre o una hoja de propiedad.

Una ventaja de la presente invención es que la división de las propiedades en uno o más subconjuntos le permite a cada subconjunto de propiedades ser más fácilmente mantenido. Otra ventaja de la presente invención es que la jerarquía del objeto es más extensible y le permite a los desarrolladores agregar funcionalidad a la jerarquía de objeto que impacta la clase base y todas las clases inferiores.

Otra ventaja de la presente invención es que el manejo del almacenamiento para las propiedades en los objetos se hace más eficiente y conveniente.

Aún otra ventaja de la presente invención es que el modelo de programación opera dentro de un ambiente programático o de marcadores. Además, el modelo de programación opera en lenguajes de programación fuertemente digitados, tales como C++ y #. Adicionalmente, el modelo de programación mantiene hojas de propiedad, cambia notificaciones y herencia de valores.



199

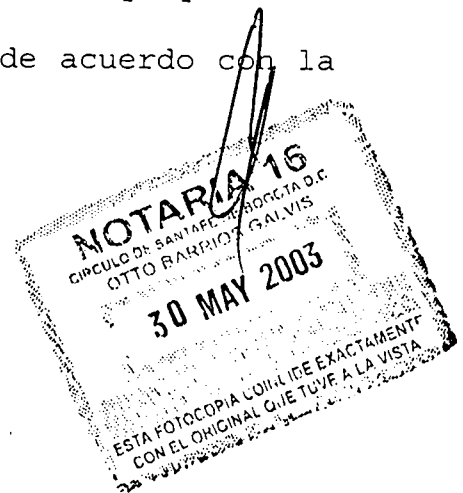
Otra ventaja de la presente invención es que las bibliotecas independientes pueden coexistir en razón a que los conflictos de nombre son menos probables porque el nombre de la clase anexa se hace efectivamente parte del nombre propiedad. Por lo tanto, si dos diferentes desarrolladores quedan cada uno una propiedad anexa que tenga una propiedad denominada "color", las dos propiedades anexas no entrarán en conflicto.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

LA FIGURA 1 ilustra un ejemplo de un dispositivo de cómputo que se puede utilizar en una modalidad de ejemplo de la presente invención.

LA FIGURA 2 es una pantalla de ejemplo que se puede crear con el dispositivo de cómputo de la figura 1.

LA FIGURA 3 es una representación gráfica de un modelo de programación que le permite a las propiedades de una clase ser anexadas a otra clase de acuerdo con la presente invención.



200

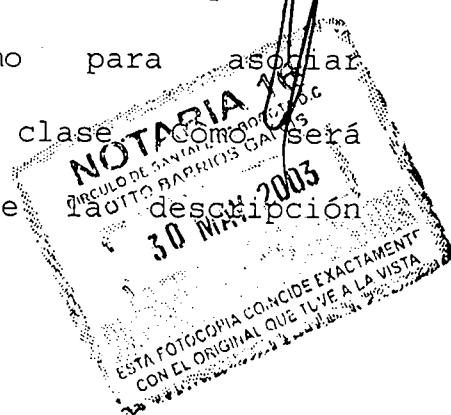
LA FIGURA 4 ilustra varias sintaxis de ejemplo para implementar el modelo de programación mostrado en la figura 3.

LA FIGURA 5 es un diagrama de flujo lógico que ilustra un proceso para establecer un valor de acuerdo con la presente invención.

LA FIGURA 6 es un diagrama de flujo lógico que ilustra un proceso para recuperar un valor de acuerdo con la presente invención.

DESCRIPCIÓN DETALLADA DE LA MODALIDAD PREFERIDA

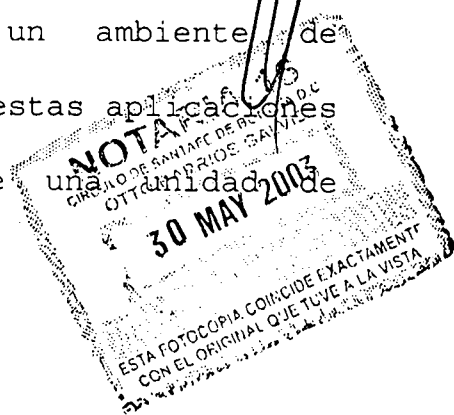
De manera resumida, la presente invención suministra un modelo de programación que permite que una nueva funcionalidad se suministre a una clase sin tener, la nueva funcionalidad, que hacer parte permanente de la clase. Además, la presente invención le permite a la nueva funcionalidad ser expresada como una propiedad no se construye dentro de la clase. En general, la presente invención suministra un mecanismo para asociar propiedades en una clase con otra clase. Como se será evidente después de la lectura de la descripción



201

detallada que sigue, el modelo de programación de la presente invención suministra propiedades dinámicas a objetos sin construir las propiedades dentro del objeto.

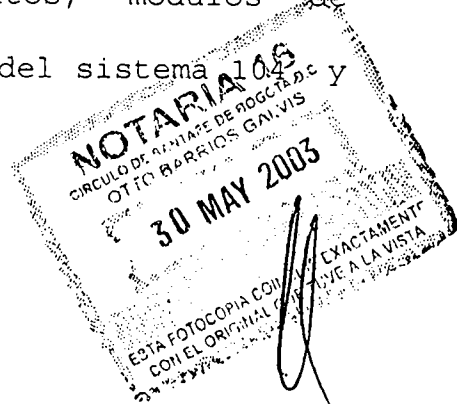
Con referencia a la figura 1, un sistema de ejemplo para implementar la invención incluye un dispositivo de cómputo, tal como un dispositivo de cómputo 100. En una configuración muy básica, el dispositivo de cómputo 100 típicamente incluye al menos una unida de procesamiento 102 y una memoria de sistema 104. dependiendo de la configuración exacta y del tipo de dispositivo de cómputo, la memoria del sistema 104 puede ser volátil (tal como la RAM), no volátil (tal como la ROM), memoria flash, etc.) o alguna combinación de las dos. La memoria del sistema 104 incluye típicamente un sistema operativo 105, uno o más módulos de programa 106, y puede incluir datos de programas 107. Ejemplos de módulos de programa 106 incluyen el Visual Studio IntelliSense de Microsoft Corporation de Redmond, WA, y otro software de ambientes de programación, que utilizan bibliotecas objeto. Además, los módulos de programa 106 incluyen aplicaciones de soporte lógico creadas utilizando un ambiente de programación de soporte lógico. Cuando estas aplicaciones de soporte lógico se ejecutan sobre una



Handwritten signature and initials

procesamiento 102, un motor de propiedad procesa la aplicación del soporte lógico de acuerdo con el modelo de programación de la presente invención. El motor de propiedad puede ser parte del sistema operativo 105 o puede ser otro módulo programable 106. esta configuración básica del dispositivo de cómputo 100 se ilustra en la figura 1 por aquellos componentes con línea punteada 108.

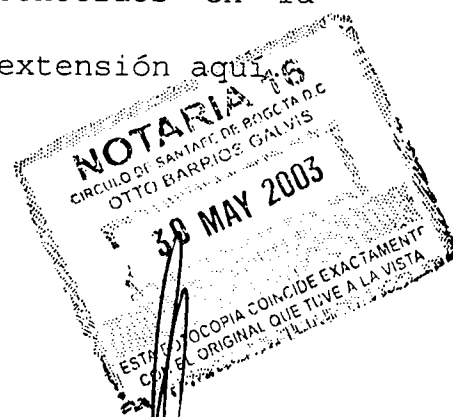
El dispositivo de cómputo 100 puede tener características o funcionalidades adicionales. Por ejemplo, el dispositivo de cómputo 100 también pueden incluir dispositivos de almacenamiento de datos adicionales (removibles y/o no removibles) tales como, por ejemplo, discos magnéticos, discos ópticos, o cintas. Tal almacenamiento adicional se ilustra en la figura 1 mediante almacenamiento removible 109, y el almacenamiento no removible 110. los medios de almacenamiento de computadora pueden incluir medios volátiles y no volátiles, removibles y no removibles implementados en cualquier método o tecnología para almacenar la información, tal como instrucciones legibles por computadora, estructuras de datos, módulos de programa, u otros datos. En la memoria del sistema 104 y



203

el almacenamiento no removible 110 son todos ejemplos de medios de almacenamiento en computadora.

Medios de almacenamiento en computadora incluye, pero no están limitados a, memoria RAM, ROM, EEPROM, FLASH u otra tecnología de memoria, CR-ROM, discos versátiles digitales (DVD) u otros dispositivos de almacenamiento óptico, cartuchos magnéticos, cintas magnéticas, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, o cualquier otro medio que se pueda utilizar para almacenar la información deseada y que se pueda acceder mediante el dispositivo de cómputo 100. Cualquiera de tales medios de almacenamiento en computadora puede ser parte del dispositivo 100. El dispositivo de cómputo 100 también puede tener dispositivo o dispositivos de entrada 112 tales como teclados, ratones, lápices, dispositivos de entrada de voz, dispositivos de entrada por toque. El o los dispositivos de salida 114 tales como la pantalla, los parlantes, la impresora, etc. también se pueden incluir. Estos dispositivos son bien conocidos en la técnica y no necesitan ser discutidos en extensión aquí.



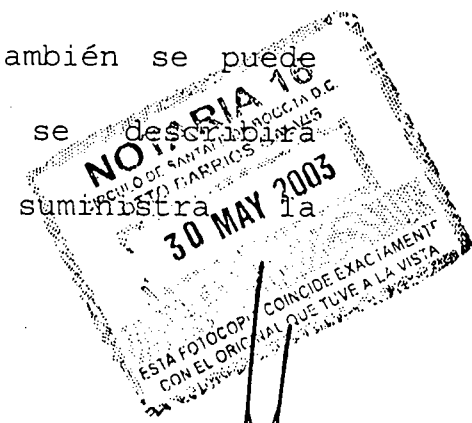
204

El dispositivo de cómputo 10° también puede contener conexiones de comunicación 116 que le permiten al dispositivo comunicarse con otros dispositivos de cómputo 118, tales como sobre una red. Las conexiones de comunicación 116 son un ejemplo de medios de comunicación son un ejemplo de medios de comunicación. Los medios de comunicación típicamente se pueden ejemplificar por medio de instrucciones legibles en computadora, estructuras de datos, módulos de programa, u otros datos en señales de datos modulados, tales como una onda portadora u otro mecanismo de transporte, e incluye cualquier medio de suministro de información. El término "señal modulada de datos" significa una señal que tiene una o más de sus características establecida o cambiada de tal manera que codifica la información en la señal. Por vía de ejemplo, y no de limitación, los medios de comunicación incluyen medios cableados tales como redes cableadas conexión cableada directa, y medios inalámbricos tales como medios acústicos, RF, infrarrojos y otros medios inalámbricos. El término "medio legible por computadora" como se utiliza aquí incluye tanto medios de almacenamiento como medios de comunicación.



205

La figura 2 es una pantalla de ejemplo que se puede crear mediante una aplicación dentro del ambiente de programación en la figura 1. aunque el modelo de programación de la presente invención se puede utilizar en varios ambientes, por vía de ejemplo no limitante, la figura 2 ilustra el uso del modelo de programación dentro de un ambiente de interfase de usuario. De esta manera, en este ejemplo, una de las aplicaciones en la figura 1 crea la pantalla 200 mostrada en la figura 2. La pantalla 200 incluye una caja de diálogo 202. La caja de diálogo 202 puede incluir cualquier número de otros controles (es decir objetos). En este ejemplo, la caja de diálogo 202 incluye una caja de lista 204, una caja de editar 206, y un objeto de contenedor 208 que tiene dos objetos de botón (por ejemplo un botón de OK 210 y un botón de cancelar 212). Como se discutirá con detalle posteriormente en conjunto con la figura 3, utilizando técnicas de programación convencional, cada uno de estos objetos (por ejemplo 202-212) es un objeto hijo derivado de un objeto base (por ejemplo una clase de elemento). Adicionalmente, de acuerdo con una modalidad de la presente invención, cada objeto hijo también se puede derivar de una clase nodo. Como se describirá posteriormente, esta clase nodo suministra la

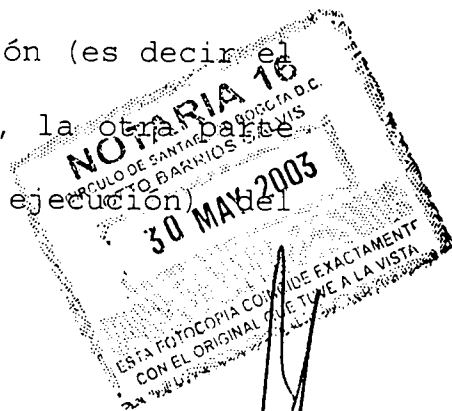


206

característica de "anexión" para las propiedades anexadas.

Aunque la apariencia de la pantalla 200 puede parecer similar a las pantallas creadas utilizando modelos de programación anteriores, el mecanismo para crear la pantalla 200 es muy diferente. En modelos de programación anterior, la pantalla de implementación del código de software 20 habría tenido tanto el comportamiento del objeto como la apariencia del objeto incorporado dentro de cada objeto mismo o en uno de los objetos padre. Sin embargo, como se describirá en detalle adelante, de acuerdo con la presente invención, las clases separadas suministran las acciones y la apariencia para los objetos 202-212 sin recurrir a una referencia explícita entre las dos clases.

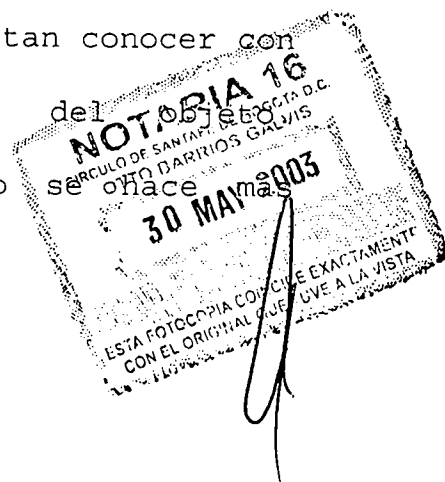
Esto le permite a los desarrolladores modificar la porción de la acción o la porción de la apariencia sin modificar el objeto completo. De esta forma, en este ejemplo de interfase de usuario, cada objeto es fabricado en dos partes. Una parte relaciona la acción (es decir el comportamiento) asociado con el objeto, y, la otra parte relaciona la apariencia, (es decir la ejecución) del



207

objeto. En razón a que la ejecución se separa del comportamiento del objeto, la apariencia del objeto puede ser fácilmente modificada para cambiar la forma en que aparece el objeto.

Aunque el ejemplo anterior de interfase de usuario factoriza convenientemente las propiedades de los objetos en un grupo de comportamiento y en un grupo de apariencia, los inventores han determinado que las propiedades de muchos objetos en varios ambientes se pueden dividir convenientemente en dos o más grupos relevantes. Entonces, de acuerdo con la presente invención, estos grupos relevantes pueden ser "implícitamente" asociados uno con el otro para implementar completamente el objeto. Esta asociación "implícita" suministra varias ventajas sobre los modelos de programación anterior. Una de las ventajas es que las terceras partes desarrolladoras pueden modificar convenientemente el grupo relevante que pertenece a su aplicación mientras mantienen los otros grupos sin ser tocados. Esto simplifica de manera notoria la cantidad de información que los desarrolladores necesitan conocer con el fin de cambiar la funcionalidad del objeto. Adicionalmente, la jerarquía del objeto

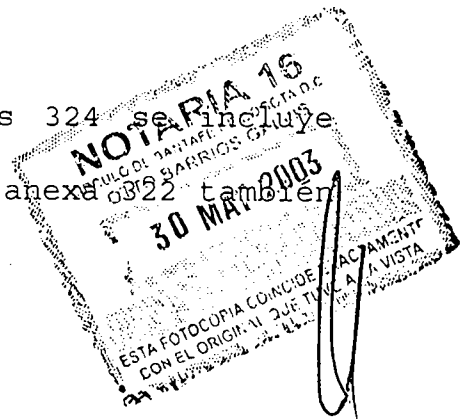


208

manejable. Otra ventaja es que el almacenamiento se reduce en razón a que cada objeto no mantiene necesariamente un valor local con un estado corriente. Al contrario, como se describirá con detalle en relación con la figura 6, el estado puede ser recuperado de varias fuentes cuando se requiera.

La figura 3 es una representación gráfica de un modelo de programación 300 que suministra un mecanismo para asociar "implícitamente" grupos relevantes de propiedades de una clase con otra clase de acuerdo con la presente invención. De manera similar a otros modelos de programación, la presente invención sustenta una clase base 302. la clase base 302 incluye un primer conjunto de propiedades 304 asociado con la clase base 302. adicionalmente, la clase base 302 incluye un conjunto de métodos 306 y un conjunto de eventos 308. sin embargo, de acuerdo con la presente invención, la clase base se deriva de una clase nodo 301. La clase nodo 301 suministra un método de SetValue() 307 y un método GetValue() 309.

Un segundo conjunto de propiedades 324 se incluye dentro de una clase anexa 322. la clase anexa 322 también

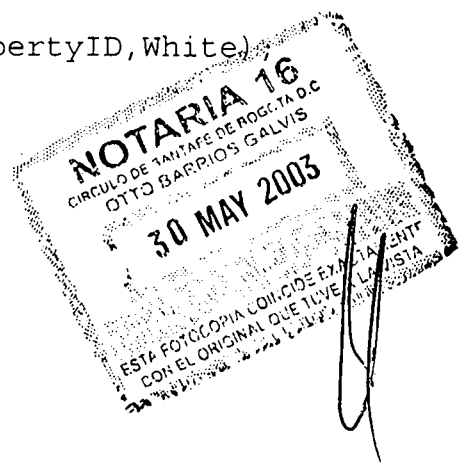


209

incluye al menos dos métodos estáticos asociados con cada uno de los segundos conjuntos de propiedades 324 (SetFont() 326 y GetFont() 330). Estos métodos estáticos se pueden enseñar como métodos globales que son accesibles a objetos que han registrado la propiedad asociada con el método estático. El desarrollador que suministra la clase anexa 322 es responsable por escribir el código para cada uno de estos métodos estáticos 326 y 330. El código incluirá un llamado a uno de los métodos suministrados por la clase nodo 301. por ejemplo, un llamado de aplicación SetoFont() 326 para un identificador específico PropertyID y un "Value" de tipo de letra dado, ejecutarán finalmente el llamado al SetValue() 307 en la clase nodo 301. los parámetros que el SetFont() 326 pasa al llamado al SetValue()307 incluirán el identificador específico PropertyID y "Value" del tipo de letra dado que se suministró en el llamado del SentFont() 326. Una función de ejemplo del SetFont() puede aparecer como sigue:

```
SetFont (PropertyID,white)

{   PropertyID->SetValue (PropertyID,White)
}.
```



210

Un experto en la técnica apreciará que la clase anexa, así, suministra un fuerte digitado para cualquier propiedad anexada. Aunque la presente invención sustenta el llamado de aplicación al SetValue() 307 directamente, esto enreda la ventaja de suministrar digitado fuerte durante el tiempo de compilación. Como se mencionó anteriormente, los digitados fuertes permiten que los errores sean detectados antes de la operación del tiempo de ejecución. En otra modalidad, la clase anexada 322 suministra un método estático adicional (por ejemplo GetFontID() 328) para cada una de las propiedades anexadas 324. Este método estático adicional es operativo para ensayar si la propiedad anexa solicitada se ha registrado. De esta forma, este método estático adicional 328 se utiliza para asegurar que la aplicación haya registrado adecuadamente la propiedad anexada antes de intentar desarrollar el conjunto y conseguir los métodos estáticos sobre la propiedad anexada. Por ejemplo, si el GetFontID() 328 detecta que la propiedad de color anexada no se ha registrado, el GtFontID() puede dar un error, puede registrar automáticamente la propiedad anexada de Color para el objeto que se solicita, o similar.



211

El siguiente se una modalidad de ejemplo para suministrar el registrar retrasado:

```
Class FontProvider
{
    public static Font GetFont(Node n)
    {
        return n.GetValue(GetFontID());
    }
    {
        n.SetValue(),newFont);
    }

    Public static DynamicProperty GetFontID()
    {
        //Delay register Font property if
necessary if(FontID=Hill)
    {
        FontID = registerProperty("Font",typeof(Font),
;Arial",...);
    }
    FontID = RegisterProperty("Font",
typeof(Font),"Arial",...);
}
}

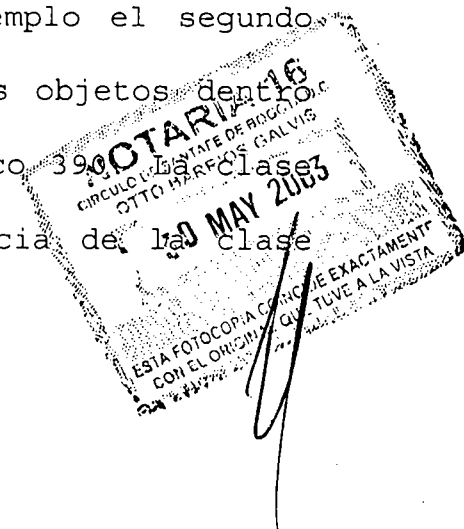
private static DynamicProperty FontID = null;
}
```



212

Similares a otros modelos de programación, la presente invención sustenta clases hija (no mostrada) de la clase base 302. Por simplicidad, la figura 3 no ilustra gráficamente ninguna clase hija de la clase base 302 o cualquier otra clase anexada, y, por lo tanto, no ilustra las clases hija que están asociadas con otras clases anexadas. Sin embargo, un experto en la técnica, después de leer la siguiente descripción, será capaz de diseñar fácilmente una aplicación que tenga clases hijas que referencien otras clases anexadas. En general, cualquier clase puede ser un "suministrador" de propiedades anexadas, en tanto que la clase suministre los métodos estáticos necesarios para las propiedades anexadas.

La clase anexada 322 le permite a terceras partes desarrolladoras la capacidad de agregar dinámicamente propiedades y funcionalidad a sus aflicciones sin modificar la clase base 302. de manera similar a la modificación de las propiedades dentro de la clase base 302, las propiedades agregadas (por ejemplo el segundo conjunto de propiedades 324) afectan los objetos dentro de un árbol completo de objeto jerárquico 390. Las clases anexada 322 no le es solicitada instancia de la clase



213

base 302, la misma clase anexada (clase anexada 322) puede suministrar el segundo conjunto de propiedades 324 a cada uno de estos objetos.

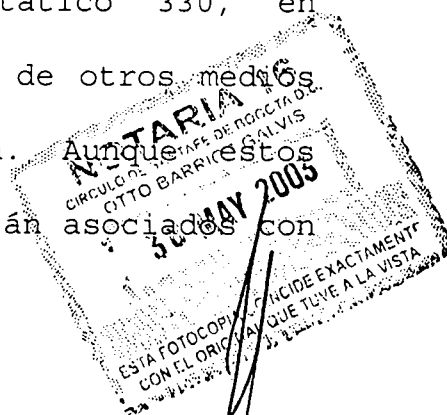
Durante el tiempo de ejecución de la aplicación, el objeto base 312 se le solicita instancia. Además, uno o más objetos hijo (por ejemplo los objetos hijo 340-350) se les solicita instancia. La solicitud de instancia de los objetos en el árbol de objeto 390 puede ser a través de control programático (por ejemplo C#), a través del lenguaje de marcadores (por ejemplo XML), o a través de otros medios. Cuando el árbol de objeto 390 se crea a través de un control programático, un compilador típicamente identifica si las instrucciones que implementan la clase anexada 322 tienen cualquier tipo de errores, tales como nombre ilegal o tipo de datos. Como se discutió anteriormente, la clase anexada 322 suministra una digitación fuerte. Por lo tanto, se pueden detectar los errores antes del tiempo de ejecución. Además, las herramientas de desarrollo de soporte lógico pueden detectar los errores durante el desarrollo de la aplicación. Cuando el árbol objeto 390 se crea a través del lenguaje de marcadores, un analizador interpreta los anuncios de marcadores. En general, un nombre de una

NOTARIA
CIRCULO DE OTTO BARRIOS SALVÉS
5 MAY 2003
ESTA FOTOCOPIA COINCIDE EXACTAMENTE
CON EL ORIGINAL QUE TIENE A LA VISTA

214

etiqueta en un lenguaje de marcadores es el nombre de la clase correspondiente.

Cada uno de estos objetos hijo 340-350 puede incluir una o más propiedades hijo (por ejemplo la propiedad hijo 341) dentro del objeto hijo (por ejemplo objeto hijo 340). Los valores para las propiedades hijo se pueden almacenar en el objeto hijo asociado. Sin embargo, de acuerdo con la presente invención, los objetos hijo no necesariamente tienen almacenamiento local para almacenar un valor. Además, cada uno de estos objetos hijo 340-350 pueden incluir una o más propiedades anexadas (por ejemplo la propiedad anexada 352 representada dentro de la caja punteada) asociada con el objeto hijo. Como se describirá con detalle en conjunto con la figura 6, la propiedad anexada 352 puede obtener un valor del almacenamiento local asociado con el objeto hijo 350, puede obtener su valor a través de la herencia (por ejemplo el objeto base 312), a través de una hoja de propiedad (no mostrada), a través de métodos programáticos (por ejemplo método estático 330, en conjunto con el método 309), y a través de otros medios de acuerdo con la presente invención. Aunque estos valores para las propiedades anexadas están asociados con



2000
215

el objeto hijo, el almacenamiento para estos valores no es típicamente parte del objeto hijo.

De nuevo, la figura 3 ilustra el modelo de programación en un ambiente de interfase de usuario. De esta forma, se notará que las propiedades dentro del objeto base 312 y los objetos hijo 340-350 se relacionan con el comportamiento de los objetos. El objeto base 312 representa un objeto de diálogo, el objeto hijo 340 representa un objeto de botón, el objeto hijo 350 representa una caja de edición, y el objeto hijo 342 representa un objeto selector que tiene dos objetos hijo 346 y 348, que representan una caja de lista y un árbol, respectivamente. Cada uno de estos objetos hijo 340-350 incluyen una o más propiedades hijo (por ejemplo la propiedad hijo 341), tal como una propiedad presionada por un botón objeto y la propiedad expandida para el objeto árbol.

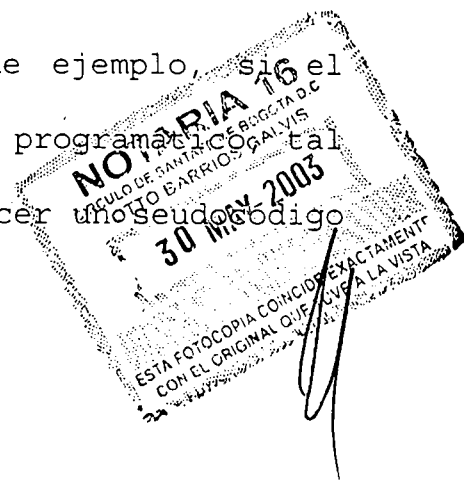
Como contraste, las propiedades dentro de la clase anexada 322 se relacionan con la apariencia del objeto. Por ejemplo, el segundo conjunto de propiedades puede incluir una propiedad y un valor por defecto para el tipo de letra, color, y similar. Este modelo de programación



se configura para establecer una propiedad al momento al tiempo de la ejecución sobre los objetos hijo. En ese alcance, la propiedad anexada (por ejemplo la propiedad anexada 352) en cada uno de los objetos hijo se configura para recibir el valor de la clase anexada.

La figura 4 ilustra varias sintaxis de ejemplo para varios ambientes operativos en los cuales el modelo de programación mostrado en la figura 3 se puede implementar. El uso del término "suministrador" dentro de cada una de las sintaxis de ejemplo denota que los métodos estáticos descritos en la sintaxis se asocian con la clase anexada que tiene un nombre de "suministrador". De esta forma, en razón a que cada una de estas sintaxis le permite a la clase anexada ser nombrada, las propiedades que son nombradas idénticamente dentro de dos diferentes clases anexadas no entrarán en conflicto una con la otra. Esto le permite a los desarrolladores desarrollar las clases de propiedad anexada sin tener que ser prudente de los potenciales conflictos del hombre.

Regresando ahora a la sintaxis de ejemplo, si el ambiente de programación es un lenguaje programático tal como el VisualBasic o el C# puede aparecer un pseudocódigo

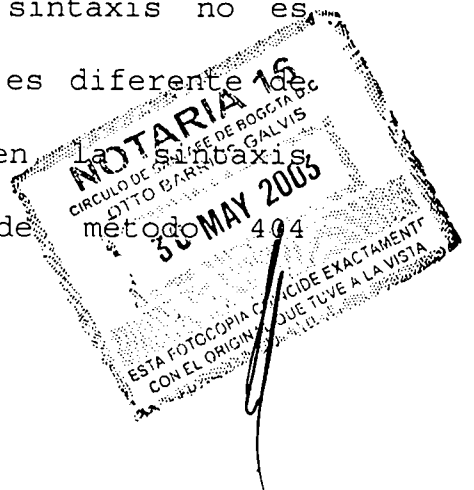


27

como se muestra en la sintaxis programática 401. la sintaxis programática 401 incluye un identificador de clase 402 que identifica la clase como conteniendo una propiedad anexada. La sintaxis programática 401 incluye además un identificador de método 404, una lista de parámetro que tiene un primer identificador 406 para especificar un nombre para una propiedad y un segundo identificador 408 para especificar un valor asociado con la propiedad especificada en el primer identificador 406. la lista de parámetro está incluida dentro del paréntesis y un punto separa el identificador de clase 402 y el identificador de método 404. en relación a la clase anexada 322 mostrada en la figura 3, para la propiedad anexada "FONT", la sintaxis programática 401 es como sigue:

Attached Class.SetFont(PopertyID, value).

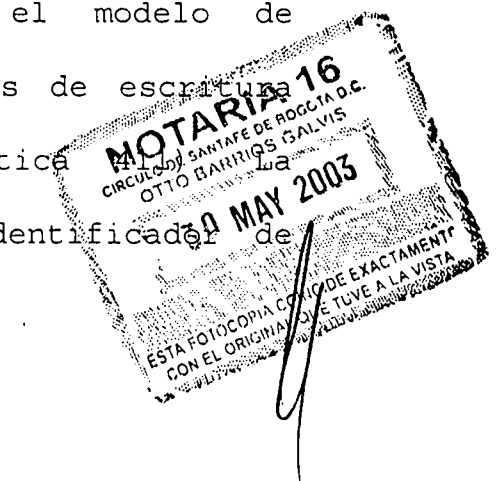
De manera interesante, la sintaxis parece lucir de manera similar a la sintaxis en los modelos de programación previos. Sin embargo, la sintaxis no es idéntica y la sintaxis programática 401 es diferente de la sintaxis anterior. Por ejemplo, en la sintaxis programática 401, el identificador de método 404



218

identifica un método estático sobre una clase suministradora (es decir, la clase anexada 322 en la figura 3). En modelos de programación anterior, el identificador de método habría identificado un método de instancia sobre un suministrador de instancia. Debido a que la presente invención utiliza la clase de proveedor (ie, la clase adjunta 322 en la figura 3), el modelo de programación puede soportar propiedades extendedoras en la marcación y las hojas de propiedades debido al tiempo de vida del momento de suministrador que no necesita ser rastreado. Otra característica de la sintaxis programática 401 es que la sintaxis es fuertemente escrita a máquina. Como se mencionó anteriormente, una sintaxis escrita fuertemente permite un soporte de buenas herramientas debido a que los errores potenciales, tal como nombres mal escritos, tipos de datos incorrectos y similares, pueden ser detectados en el tiempo de compilación en vez de en el tiempo de corrida.

Adicionalmente, los ambientes de programación tales como Visual Basic, C# o similares, el modelo de programación también permite una sintaxis de escritura suelta (por ejemplo sintaxis programática 411 incluye un identificador de



219

objeto 412 que identifica un objeto objetivo, un identificador de método 414 que identifica un método estático dentro de una clase de propiedades adjunta, una lista de parámetros que incluye un identificador de propiedades adjunto y un valor 419. el identificador de propiedades adjunto incluye un identificador de clase 416 y un identificador de propiedades 418 que identifica las propiedades adjuntas dentro de la clase adjunta asociada.

En otra modalidad, el modelo de programación puede ser implementado usando sintaxis de lenguaje de marcación, tal como la sintaxis de marcación 421. La sintaxis de marcación 421 incluye una bandera 422, una propiedad adjunta que tiene un designador de suministrador 424 y un identificador de propiedades 426, y un valor 428. En una modalidad, la bandera 422 es el nombre del objeto objetivo (por ejemplo, el objeto botón 340 en la figura 3). Típicamente, la sintaxis de marcación 421 inicia y finaliza con un símbolo abierto y cerrado "<" y ">", respectivamente. Nuevamente, un analizador lleva a cabo el procesamiento para localizar en donde las propiedades adjuntas existentes debido a que las propiedades adjuntas no están directamente sobre la clase.

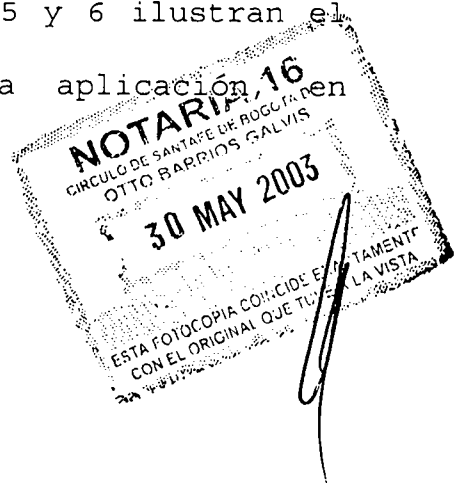
NOTA: ARIA
CIRCULO DE SALUD DE BOGOTA D.C.
OTTO BARROS SALVIS
30 MAY 2003
ESTA FOTOCOPIA CUMPLE EXACTAMENTE
CON EL ORIGINAL. TUVE A LA VISTA

220

En todavía otra modalidad, el modelo de programación puede ser implementado usando un lenguaje de marcación extensible (XML) con hojas de estilo, tales como sintaxis de hoja de estilo 431, mostrada en la figura 4. Para esta modalidad, la sintaxis de hoja de estilo 431 incluye un atributo de "bandera" 432. El atributo de bandera 432 corresponde a la clase que requiere el procesamiento. Un ejemplo ilustrativo para las propiedades adjuntas en XML es como sigue:

```
<Button xmlns:ap="expandoNamespace"Pressed="false"
ap:Expandos.String="string"/>
```

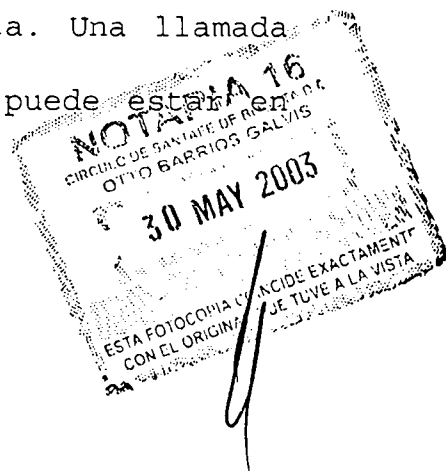
Cuando una aplicación que implementa las propiedades adjuntas de acuerdo con la presente invención se ejecuta, la aplicación muestra inmediatamente los objetos en el árbol de objeto. Entonces, la aplicación, en conjunto con el motor de propiedades, maneja las operaciones de tiempo de corrida, lo que incluye recuperar y establecer los valores para los objetos. Las figuras 5 y 6 ilustran el procesamiento llevado a cabo por la aplicación en conjunto con el motor de propiedades.



221

Sin embargo, antes de que cada proceso SetValue() o tGetValue(), mostrado en la figura 5 y 6, sean llevado a cabo, la propiedad adjunta que es especificada dentro de cualquiera de estos dos procesos debe ser registrada. Como se mencionó anteriormente, una modalidad llevara cabo una verificación para determinar si la propiedad ha sido registrada. Esta verificación (por ejemplo, GetFontID() 328) puede ser llevada a cabo dentro de cualquiera de los métodos estáticos que corresponden a la propiedad adjunta (por ejemplo SetFont() 326 y GetFont() 330). Debido a que esta verificación debe ser llevada a cabo cada vez que cualquiera de los métodos estáticos sea llamado, esta modalidad incurre en una falta de comportamiento. Aquellos versados en la técnica podrán apreciar que otras modalidades que optimizan el registro de la propiedad pueden ser implementadas sin salirse de la presente invención, tal como requerir que la aplicación lleve a cabo el registro.

El registro de la propiedad adjunta regresa un PropertyID único para la propiedad adjunta. Una llamada ilustrativa para registrar una propiedad puede estar en la forma que sigue:

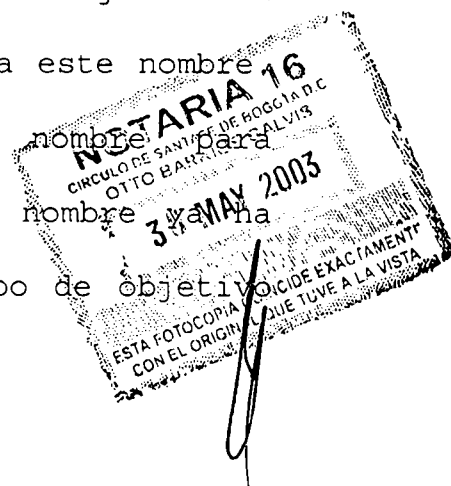


222

```
BarDP =  
RegisterProperty("Bar",0,typeof(string),typeof(Button),0x  
3);.
```

En esta modalidad, la variable BarDP podrá contener el identificador único para la propiedad adjunta. Este identificador único es entonces usado en las funciones SetValue() y GetValue() de llamadas. En una modalidad, la propiedad de registro es llamada, directa o indirectamente desde el constructor estático del propietario. El Propietario se refiere a la clase que adjunta la propiedad adjunta. Una vez que la propiedad adjunta es registrada, cualquier momento que case con el tipo de objetivo para la propiedad adjunta puede adjuntar "implícitamente" la propiedad llamando los métodos estáticos Get y Set asociados con la propiedad adjunta.

Como se mostró anteriormente en la llamada ilustrativa para el registro de una propiedad, "Bar" indica o designa un nombre para la propiedad adjunta. El motor de propiedades típicamente no utiliza este nombre. Sin embargo, un analizador usa este nombre para determinar si una propiedad con el mismo nombre ha sido registrada por el propietario. El tipo de objetivo



223

es botón. Para las propiedades adjuntas, el propietario y el objetivo no será el mismo. El valor por defecto es "0". Como será descrito en detalle en conjunto con la figura 6, el motor de propiedades determina si se recupera un valor para la propiedad adjunta. Por lo tanto, la llamada RegisterProperty incluye bits de comportamiento que identifican etapas en las cuales un valor para la propiedad se busca, tal como herencia, hojas de propiedades. En la llamada ilustrativa anterior, los bits de comportamiento "0x3" indica que un valor local, una hoja de propiedades, y de herencia son buscadas para poder determinar el valor para la propiedad adjunta. Así, en una modalidad, la interpretación de estos bits de comportamiento pueden ser codificadas dentro del método estático GetValue (). Alternativamente, la determinación de estas etapas será provista por una expresión que modela las relaciones entre las propiedades de una manera formal como se describió en la solicitud copendiente asignada generalmente titulada "Extensible On-Demand Property System", Solicitud de patente No. _____, presentada en _____, que se incorpora aquí como referencia.



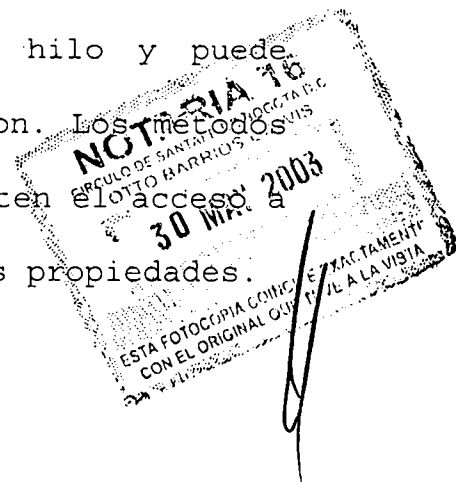
224

En una modalidad, el código para la clase adjunta puede tomar la forma mostrada en la tabla 1.

```
Class BarProvider: Object {  
    public static DynamicProperty BarDP =  
Registerproperty("Bar", 0, typeof(string),  
typeof(Button)):  
  
    public static string GetBar(Button boton){  
        return (string) boton.GetValue(BarDP);  
    }  
    public static void SetBar(Button Button, string  
value){  
        boton.SetValue(BarDP, value);  
    }  
}
```

Tabla 1.

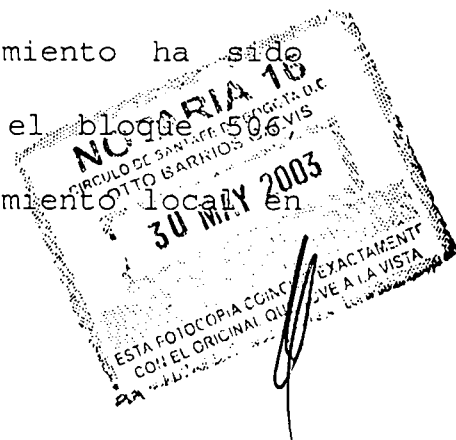
En el anterior código, la propiedad adjunta llamada "Bar" está definida dentro de la clase adjunta "BarProvider". La Propiedad adjunta llamada "Bar" está definida como un tipo de dato como un hilo y puede solamente estar ligada a los objetos Button. Los métodos estáticos son fuertemente escritos y permiten el acceso a las propiedades y al establecimiento de las propiedades.



225

Una vez que la propiedad adjunta ha sido registrada, el SetValue() y el GetValue() como funciones pueden ser llevados a cabo. La figura 5 es un diagrama de flujo lógico que ilustra un proceso para establecer un valor de acuerdo con la presente invención. El procesamiento se inicia en el bloque 501, en donde la función establece ha sido iniciada para cambiar un valor asociado con alguna propiedad. En general, el proceso SetValue 500 almacenará localmente el valor para una propiedad de interés (de aquí en adelante se refiere a la propiedad interesada). El procesamiento continúa con el bloque de decisión 502.

En el bloque de decisión 502, se hace una determinación de si el almacenamiento local existe para la propiedad interesada. Debido a que la presente invención no tiene de manera automática almacenamiento para cada propiedad en cada instante de un objeto, si la propiedad interesada no se le ha dado previamente un valor almacenado para esta, el almacenamiento es asignado (bloque 504). Una vez que el almacenamiento ha sido asignado, el procesamiento continúa en el bloque 506 como si se hubiera detectado un almacenamiento local en el bloque de decisión 502.



226

El bloque 506, el valor es copiado dentro del almacenamiento asociado con la propiedad interesada. Una vez que la propiedad interesada ha cambiado de sus estados, envían notificaciones del cambio en el estado (bloque 508). En una modalidad, esta notificación puede involucrar establecer un bit sucio para indicar a las propiedades dependientes que su estado puede no ser válido ahora. Alternativamente, la notificación puede involucrar reportar a cada dependiente que una fuente ha cambiado como se describió en la solicitud de patente anteriormente mencionada. El procesamiento continúa en el bloque de decisión 510.

En el bloque de decisión 510, se hace una determinación de si el valor fue almacenado en el caché para la propiedad interesada. Si no hay un valor en el caché para esta propiedad interesada, el proceso finaliza. Sin embargo, si hay un valor en el caché, el caché es limpiado (bloque 512) debido a que este valor no es ya válido. El proceso entonces finaliza.

La figura 6 es un diagrama de flujo lógico que ilustra un proceso para recuperar un valor de acuerdo con

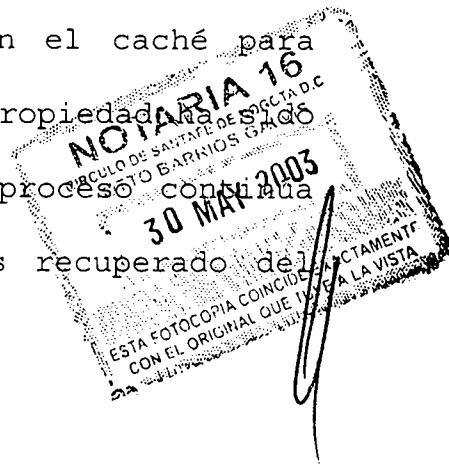


227

la presente invención. El proceso se inicia en el bloque 601, en donde se inicia una pregunta. El proceso continúa en el bloque 602.

En el bloque 602, la aplicación indica que una de las propiedades de interés (de aquí en adelante será referida como propiedad interesada) requiere ser actualizada. Típicamente, esto puede ocurrir cuando un objeto está buscando propiedades para su estado actual. El proceso 600 determina un valor para las propiedades por medio de verificar diversas etapas. Estas etapas son definidas cuando la propiedad es registrada. Los bloques de decisión 606 a 610 representan etapas ilustrativas, pero otras etapas pueden ser agregadas sin salirse de la presente invención. El proceso continúa en el bloque de decisión 604.

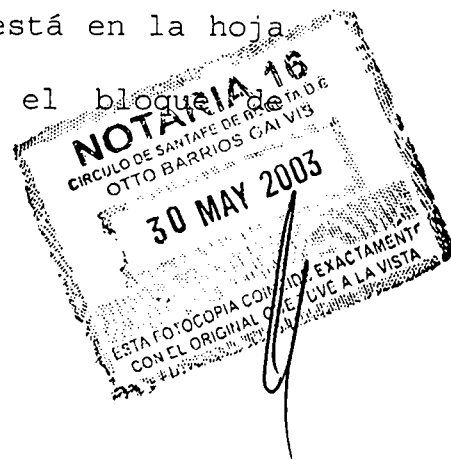
En el bloque de decisión 604, el proceso verifica el caché para determinar si la propiedad interesada ha sido guardada en el caché anteriormente. Típicamente, las propiedades interesadas son guardadas en el caché para poder optimizar su recuperación. Si la propiedad ha sido guardada anteriormente en el caché, el proceso continúa en el bloque 612, en donde el valor es recuperado de



228

caché. Alternativamente, el proceso continúa en el bloque de decisión 606. en el bloque de decisión 606, se hace una determinación de si el valor para la propiedad interesada es local. El valor será local si un SetValue() ha sido llevado a cabo para establecer un valor localmente para la propiedad, como se describió en la figura 5. Si el valor es local, el proceso continúa en el bloque 612, en donde el valor es recuperado del almacenamiento local. Si el valor no es local, el proceso continúa en el bloque de decisión 608.

En el bloque de decisión 608, se hace una determinación de si el valor para la propiedad interesada está disponible en una hoja de propiedades. Cuando una clase de base tiene una hoja de propiedades asociada, la hoja de propiedades especifica cómo cada uno de los hijos de la clase de base se convertirá una vez que se inicie. Si las propiedades interesadas están disponibles en una hoja de propiedades, el proceso continúa en el bloque 612, en donde el valor es recuperado de la hoja de propiedades. Sin embargo, si el valor no está en la hoja de propiedades, el proceso continúa en el bloque de decisión 610.

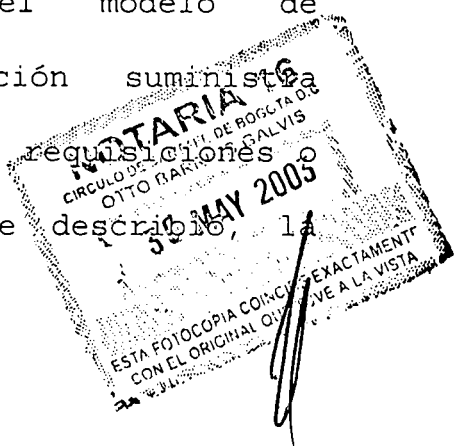


209

El bloque de decisión 610, se hace una determinación de si el valor para la propiedad interesada puede ser obtenido de una herencia (por ejemplo un valor heredado). Los valores heredados son derivados de un objeto pariente del objeto hijo. Si uno de los objetos parientes tiene una propiedad interesada, el proceso continúa en el bloque 612, en donde el valor es recuperado del objeto pariente. Sin embargo, si el valor no es heredado, un valor por defecto de la clase asociada con la propiedad adjunta es recuperada (bloque 614). El proceso continúa en el bloque 616.

En el bloque 616, la aplicación calcula un peso métrico basado en la etapa en la cual el valor fue recibido. El peso métrico es almacenado y es usado para efectuar cuales propiedades interesadas para poder optimizar recuperaciones futuras. El proceso entonces finaliza.

Adicionalmente, para los procesos SetValue y GetValue descritos anteriormente, el modelo de programación de la presente invención suministra funciones mejoradas tales como grupos de requisiciones o grupos de notificaciones. Así, como se describe, la



750

presente invención suministra un modelo de programación que suministra un sistema de propiedades con base en la dependencia, que soporta relaciones complejas entre las propiedades por medio del uso de clases adjuntas. Las aplicaciones construidas usando este modelo de programación se dimensionan muy bien debido a que la técnica de administración de propiedades no requiere un almacenamiento con base en perinstantia local excesiva. En su lugar, el modelo de programación promueve la reutilización de los valores de propiedades, de modo que a través del uso de hojas de propiedad y herencia al nivel de clase de base. Como se describió anteriormente, la presente invención suministra un mecanismo de administración de propiedades en donde los objetos en la jerarquía de objetos almacena los valores de propiedades por medio de adjuntarlos.

La anterior especificación, ejemplos y datos suministran una descripción completa de la fabricación y uso de la composición de la invención. Ya que muchas modalidades de la invención pueden ser hechas sin salirse del espíritu y alcance la invención, la invención reside en las reivindicaciones adjuntas aquí.



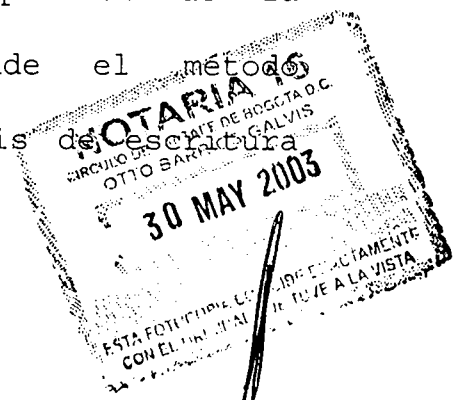
231

REIVINDICACIONES

1. Un medio legible por computador que incluye un objeto que tiene una propiedad de un primer conjunto de propiedades, el medio legible por computador comprende adicionalmente:

Una estructura de datos que incluye definiciones para cada uno de un segundo conjunto de propiedades e incluye por lo menos un método estático asociado con una propiedad de sacada del segundo conjunto de propiedades, por lo menos un método estático tiene un primer parámetro que identifica de manera única aquella propiedad, el método estático es operativo para asociar esa una propiedad con el objeto sin especificar una referencia explícita a esa propiedad en el objeto.

2. El medio legible por computador de la reivindicación 1, en donde el método estático soporta una sintaxis de descripción fuerte.



- 232
3. El método legible por computador de la reivindicación 1, en donde el método estático es operativo de manera que asocia a aquella una propiedad con el objeto que incluye recuperar un valor para el objeto son tener el valor almacenado localmente en el objeto.
 4. El medio legible por computador de la reivindicación 3, en donde recuperar el valor comprende de terminar cual entre una pluralidad de etapas tiene el valor asociado con esa una propiedad.
 5. El medio legible por computador de la reivindicación 4, en donde una etapa incluye un objeto pariente del objeto.
 6. El medio legible por computador de la reivindicación 4, en donde una etapa incluye una hoja de propiedades.



233

7. El medio legible por computador de la reivindicación 1, en donde el método estático comprende incluye adicionalmente el segundo parámetro, el segundo parámetro corresponde a un valor dado para aquella propiedad, y en donde el método estático es operativo para asociar dicha una propiedad con el objeto que comprende establecer el valor dado en un almacenamiento local para el objeto.

8. El medio legible por computador de la reivindicación 1, en donde el primer conjunto de propiedades corresponde a comportamientos del objeto, y el segundo conjunto de propiedades corresponde a una apariencia del objeto.

9. Un medio legible por computador que tiene componentes ejecutables por computador, comprende:

Una clase de base que incluye una pluralidad de propiedades; y

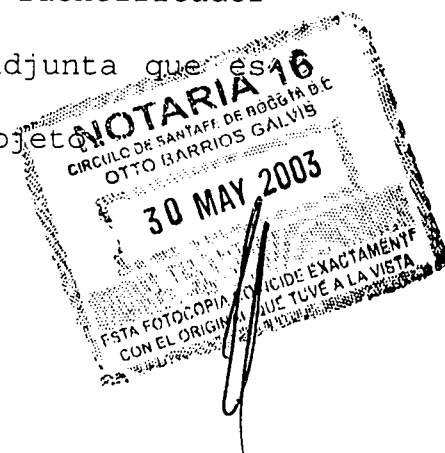


284

Una clase adjunta que incluye una pluralidad de propiedades adjuntas, cada una de las propiedades adjuntas es capaz de estar asociada con una instante de un objeto derivado de la clase de base, la clase adjunta incluye adicionalmente un método estático para asociar una entre la pluralidad de propiedades adjuntas con el instante del objeto.

10. El medio legible por computador de la reivindicación 9, en donde el objeto derivado de la clase de base hereda en por lo menos un método de una clase de nodo, el un método es llamado por el método estático cuando se asocia con una propiedad adjunta con la instancia del objeto.

11. El medio legible por computador de la reivindicación 10, en donde el método estático y el un método cada uno incluye un primer parámetro para pasar un identificador único para la una propiedad adjunta que es asociada con el instante del objeto.



235

- 12. El medio legible por computador de la reivindicación 11, en donde el identificador único está disponible después de registrar aquella una propiedad adjunta en el tiempo de corrido.
- 13. El medio legible por computador de la reivindicación 10, en donde el un método soporta una sintaxis de tipo de escritura suelta.
- 14. El medio legible por computador de la reivindicación 9, en donde el método estático soporta una sintaxis de escritura fuerte.
- 15. El medio legible por computador de la reivindicación 9, en donde asociar aquella una propiedad adjunta comprende recuperar un valor para el objeto de un objeto pariente del objeto.
- 16. El medio legible por computador de la reivindicación 9, en donde asociar aquella



236

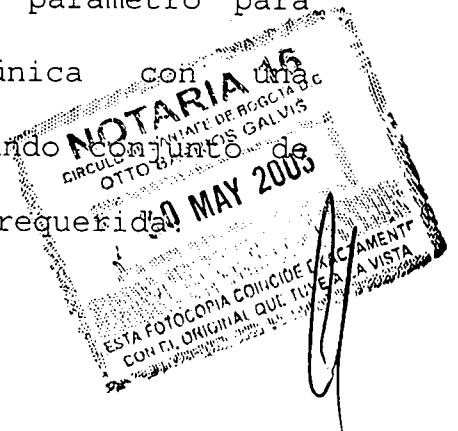
una propiedad adjunta comprende recuperar un valor para el objeto de una hoja de propiedades.

17. Un sistema de computador que utiliza objetos, comprende:

Un primer objeto derivado de una primera clase, el primer objeto incluye un prime conjunto de propiedades;

Una segunda clase que suministra un segundo conjunto de propiedades al primer objeto, la segunda clase incluye un método estático operativo para adjuntar una entre el segundo conjunto de propiedades al primer objeto en respuesta a una operación asociada con el primer objeto.

18. El sistema de computador de la reivindicación 17, en donde el método estático incluye un primer parámetro para identificar de manera única con una propiedad extraída del segundo conjunto de propiedades la operación es requerida.



19. El sistema de computador de la reivindicación 18, en donde aquella una propiedad es registrada antes de que el primer parámetro pueda identificar de manera única aquella una propiedad.
20. Un método implementado por computador para establecer un estado para una propiedad, comprende:

Declarar un primer objeto que tiene un primer conjunto de propiedades;

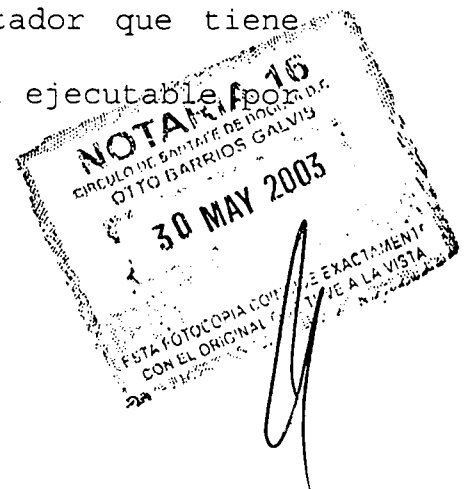
Registrar una o más propiedades dinámicas que están asociadas con el primer objeto, las propiedades dinámicas suministran un segundo conjunto de propiedades al primer objeto, en donde el almacenamiento para el segundo conjunto de propiedades no está dentro del primer objeto.

21. El método implementado por computador de la reivindicación 20, en donde registrar la una o más propiedades dinámicas incluye asignar



un identificador único a cada una de la una o más propiedades dinámicas.

22. El método implementado por computador de la reivindicación 21, que comprende adicionalmente llamar un método estático que tiene un primer parámetro, el primer parámetro se refiere al identificador único.
23. El método implementado por computador de la reivindicación 22, en donde el método estático es escrito fuertemente.
24. El método implementado por computador de la reivindicación 20, en donde el primer objeto tiene uno o más objetos hijo, cada objeto hijo está por lo tanto asociado con el segundo conjunto de propiedades.
25. El medio legible por computador que tiene por lo menos una instrucción ejecutable por computador, que comprende:





-
- computador de la
- nde el medio de
- NOTARIA 16**
- CIRCULO DE SANTAFE ROSITA D.C.
- SANTO PARRIO ALVIS
- 30 MAR 2003**
- ESTA FOTOCOPIA COINCIDE EXACTAMENTE
CON EL ORIGINAL QUE TIENE LA VISTA

240

instrucción comprende una formación en una hoja de propiedades.

30. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Un identificador de clase que identifica una clase de propiedades adjunta;

Un identificador de método que identifica un método en la clase de propiedades adjuntas para afectar una propiedad adjunta asociada con un objeto objetivo; una lista de parámetros, que comprende:

Un identificador de propiedades que identifica la propiedad adjunta.

31. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Un identificador de objeto que identifica un objeto objetivo;



241

Un identificador de método que identifica un método en una clase de propiedades adjunta para afectar una propiedad adjunta asociada con el objeto objetivo;

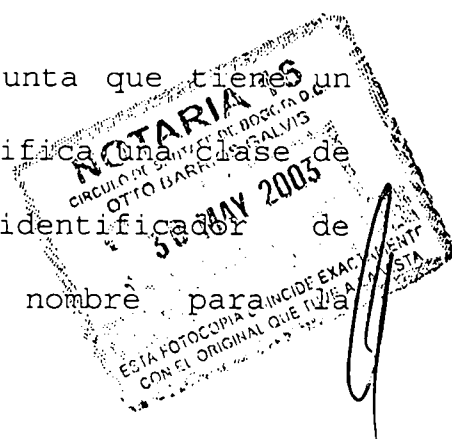
Una lista de parámetros, comprende:

Un identificador de propiedad adjunta que incluye un identificador de clase que identifica la clase de propiedades adjunta y un identificador de propiedades que identifica un nombre para una propiedad adjunta dentro de la clase de propiedades adjunta.

32. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Una bandera que representa un objeto objetivo;

Un identificador de propiedad adjunta que tiene un designador suministrador que identifica una clase de propiedades adjunta y un identificador de propiedades que identifica un nombre para



242

propiedad adjunta dentro de las clases de propiedades adjunta.

33. Un medio legible por computador que tiene por lo menos una instrucción codificada por computador, la instrucción comprende:

Un esquema XML para definir un objeto objetivo, una clase de propiedades adjunta, y una propiedad adjunta dentro de la clase de propiedades adjunta, de tal manera que la propiedad adjunta se asocia con el objeto objetivo sin una referencia explícita a la propiedad adjunta en el objeto objetivo.



243

RESUMEN

Se describe aquí un mecanismo para permitir una nueva funcionalidad para un objeto de ser expresado como una propiedad que no está construida dentro de la clase de la cual el objeto se deriva. Más específicamente, el mecanismo asocia propiedades en una clase con otra clase. Un medio legible por computador, que incluye un objeto que tiene una propiedad en un primer conjunto de propiedades, y que incluye adicionalmente una estructura de datos. La estructura de datos incluye definiciones para cada uno de un segundo conjunto de propiedades e incluye por lo menos un método estático. El método estático está asociado con una propiedad sacada del segundo conjunto de propiedades e incluye un primer parámetro. El primer parámetro identifica de manera única aquella una propiedad. El método estático es operativo para asociar dicha una propiedad con el objeto sin especificar una referencia explícita a dicha una propiedad en el objeto. La propiedad es



244

registrada durante el tiempo de corrida para poder recibir el identificador único.



245

SOLICITUD DE PATENTE

DECLARACIÓN Y PODER DE ABOGADO

REGISTRO DE ABOGADO No. 50037.96US01 REGISTRO MS No.196295.1

Como inventor nombrado adelante, Declaro que:

Mi residencia/dirección postal y ciudadanía son como se establece adelante junto a mi nombre:

Soy el primero, único y original inventor (si solo un nombre es listado adelante) o primeros, únicos y originales inventores (si se lista pluralidad de nombres adelante) de la materia de asunto que se reivindica y para el cual una patente se pretende sobre la invención titulada: SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

la descripción de la cual se adjunta aquí a menos que la siguiente casilla sea marcada:



246

9 que fue presentada en _____ como solicitud
Estadounidense serie No. o Solicitud Internacional Número
_____ y fue corregido en _____ (si aplica).

Declaro por medio de la presente que he revisado y
entendido los contenidos de la descripción identificada
anteriormente, incluyendo las reivindicaciones, como se
corrigió mediante cualquier corrección(es) referidas
anteriormente. Reconozco la obligación de describir toda
la información que es material patentable como se definió
en el CFR 37 1.56.

Solicitud(es) Extranjera(s) y/o Reivindicación de
Prioridad Extranjera

Por medio de la presente reivindico los beneficios de la
prioridad extranjera de acuerdo con el título 35, sección
119 del código de Estados Unidos de una solicitud(es)
extranjera(s) para certificado de patente o inventor(es)
listado adelante y tiene también identificado adelante
cualquier solicitud extranjera para certificado
patente o inventor(es) que tiene una



257

presentación antes de la solicitud sobre la que la prioridad es reivindicada:

PAÍS	NÚMERO DE SOLICITUD	FECHA DE PRESENTACIÓN	PRIORIDAD REIVINDICADA DE ACUERDO CON 35 U.S.C.
			Si: 9 No: 9

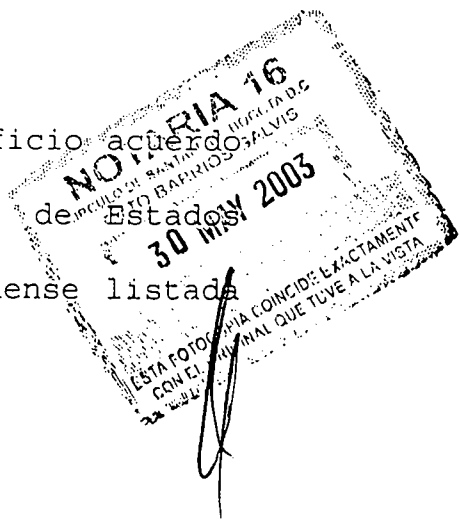
Solicitud Provisional

Por medio de la presente reivindico el beneficio de acuerdo con el título 35, sección 119(e) del código de Estados Unidos de cualquier solicitud(es) provisional(es) Estadounidenses listado adelante:

NÚMERO DE SOLICITUD	FECHA DE PRESENTACIÓN

Reivindicación de Prioridad Estadounidense

Por medio de la presente reivindico el beneficio de acuerdo con el título 35, sección 120 del código de Estados Unidos de cualquier solicitud(es) Estadounidense listada



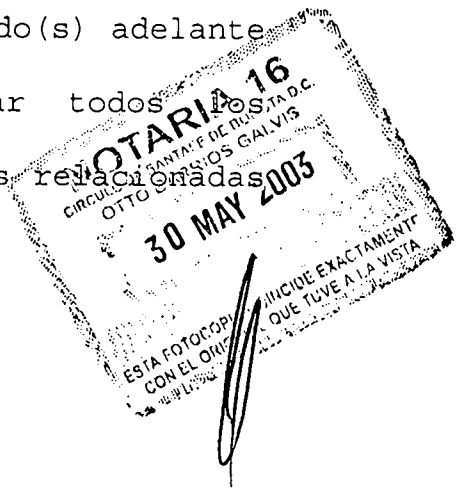
248

adelante y, en la medida en que la materia asunto de cada una de las reivindicaciones de esta solicitud no está descrita en la solicitud Estadounidense en la forma dada en el primer párrafo del título 35, del código de Estados Unidos sección 112, Reconozco la obligación de describir la información material como se definió en el título 37, Código de Regulaciones Federal, Sección 1.56(a) que ocurrió entre la fecha de presentación de la solicitud anterior y la nacional o fecha de presentación internacional PCT de esta solicitud:

NÚMERO DE SOLICITUD	FECHA DE PRESENTACIÓN	ESTADO (patentado/pendiente/abandonado)

PODER DE ABOGADO

Como inventor, por medio de la presente nombro al siguiente(s) abogado(s) y/o agente(s) listado(s) adelante para procesar esta solicitud y tramitar todos los negocios en la Oficina de Marcas y Patentes, revuacionadas este:



27488

Cliente No. 27488

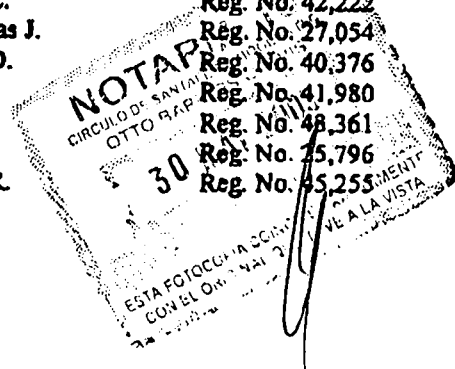


Albrecht, John W.
Ali, M. Jeffer
Ahera, Allan G.
Anderson, Gregg L.
Batzli, Brian H.
Beard, John L.
Berna, John M.
Branch, John W.
Bremer, Dennis C.
Brown, Jeffrey C.
Bruess, Steven C.
Byrne, Linda M.
Campbell, Keith
Carlson, Alan G.
Caspers, Philip P.
Clifford, John A.
Daignault, Ronald A.
Daley, Dennis R.
Dalglish, Leslie E.
Daulton, Julie R.
DeVries Smith, Katherine M.
DiPietro, Mark J.
Doscotch, Matthew A.
Edell, Robert T.
Epp Ryan, Sandra
Glance, Robert J.
Goggin, Matthew J.
Golla, Charles E.
Gorman, Alan G.
Gould, John D.
Gregson, Richard
Gresens, John J.
Harner, Samuel A.
Hamre, Curtis B.
Harrison, Kevin C.
Hertzberg, Brett A.
Hillson, Randall A.
Holzer, Jr., Richard J.
Hope, Leonard J.
Jardine, John S.
Johnston, Scott W.
Kadievitch, Natalie D.
Kaseburg, Frederick A.
Kettelberger, Denise
Keys, Jeramie J.
Knearl, Homer L.
Kowalchyk, Alan W.
Kowalchyk, Katherine M.
Lacy, Paul E.

Reg. No. 40,481
Reg. No. 46,359
Reg. No. 40,274
Reg. No. 28,828
Reg. No. 32,960
Reg. No. 27,612
Reg. No. 43,496
Reg. No. 41,633
Reg. No. 40,528
Reg. No. 41,643
Reg. No. 34,130
Reg. No. 32,404
Reg. No. 46,597
Reg. No. 25,959
Reg. No. 33,227
Reg. No. 30,247
Reg. No. 25,968
Reg. No. 34,994
Reg. No. 40,579
Reg. No. 36,414
Reg. No. 42,157
Reg. No. 28,707
Reg. No. 48,957
Reg. No. 20,187
Reg. No. 39,667
Reg. No. 40,620
Reg. No. 44,125
Reg. No. 26,896
Reg. No. 38,472
Reg. No. 18,223
Reg. No. 41,804
Reg. No. 33,112
Reg. No. 46,754
Reg. No. 29,165
Reg. No. 46,759
Reg. No. 42,660
Reg. No. 31,838
Reg. No. 42,668
Reg. No. 44,774
Reg. No. 48,835
Reg. No. 39,721
Reg. No. 34,196
Reg. No. 47,695
Reg. No. 33,924
Reg. No. 42,724
Reg. No. 21,197
Reg. No. 31,535
Reg. No. 36,848
Reg. No. 38,946

Larson, James A.
Leonard, Christopher J.
Liepa, Mara E.
Lindquist, Timothy A.
Lown, Jean A.
Mayfield, Denise L.
McDonald, Daniel W.
McIntyre, Jr., William F.
Mitchem, M. Todd
Mueller, Douglas P.
Nelson, Anna M.
Parsons, Nancy J.
Pauly, Daniel M.
Phillips, John B.
Pino, Mark J.
Prendergast, Paul
Pytel, Melissa J.
Qualey, Terry
Reich, John C.
Reiland, Earl D.
Roberts, Fred
Samuels, Lisa A.
Schmaltz, David G.
Schuman, Mark D.
Schumann, Michael D.
Soull, Timothy B.
Sebald, Gregory A.
Skoog, Mark T.
Spellman, Steven J.
Stoll-DeBell, Kirstin L.
Sullivan, Timothy
Sumner, John P.
Swenson, Erik G.
Tellekson, David K.
Trembath, Jon R.
Tunheim, Marcia A.
Underhill, Albert L.
Vandenburgh, J. Derek
Wahl, John R.
Weaver, Karrie G.
Welter, Paul A.
Whipps, Brian
Whitaker, John E.
Williams, Douglas J.
Withers, James D.
Witt, Janelle
Wu, Tong
Young, Thomas
Zeuli, Anthony R.

Reg. No. 40,443
Reg. No. 41,940
Reg. No. 40,066
Reg. No. 40,701
Reg. No. 48,428
Reg. No. 33,732
Reg. No. 32,044
Reg. No. 44,921
Reg. No. 40,731
Reg. No. 30,300
Reg. No. 48,935
Reg. No. 40,364
Reg. No. 40,123
Reg. No. 37,206
Reg. No. 43,858
Reg. No. 46,068
Reg. No. 41,512
Reg. No. 25,148
Reg. No. 37,703
Reg. No. 25,767
Reg. No. 34,707
Reg. No. 43,080
Reg. No. 39,828
Reg. No. 31,197
Reg. No. 30,422
Reg. No. 42,137
Reg. No. 33,280
Reg. No. 40,178
Reg. No. 45,124
Reg. No. 43,164
Reg. No. 47,981
Reg. No. 29,114
Reg. No. 45,147
Reg. No. 32,314
Reg. No. 38,344
Reg. No. 42,189
Reg. No. 27,403
Reg. No. 32,179
Reg. No. 33,044
Reg. No. 43,245
Reg. No. 20,890
Reg. No. 43,261
Reg. No. 42,222
Reg. No. 27,054
Reg. No. 40,376
Reg. No. 41,980
Reg. No. 48,361
Reg. No. 25,796
Reg. No. 35,255



254

Adicionalmente, por medio de la presente también nombro a los siguientes abogados para procesar esta solicitud y para tramitar todos los negocios en la Oficina de Marcas y Patentes relacionadas este:

Kate E. Sako, Reg, No. 32,628

Daniel D. Crouse, Reg. 32,022

Enviar Correspondencia a: Marcia A. Tunheim; Reg. No. 42,189 Merchant & Gould P.C. P.O. Box 2903 Minneapolis, MN 55402-0903	Llamadas a teléfono directo a: Contacto: Marcia A. Tunheim (206) 342.6259
---	---

Por medio de la presente declaro que todos las afirmaciones hechas aquí de mi propio conocimiento son verdad y que toas las afirmaciones hechas sobre información y opinión son creídas para ser verdad; y adicionalmente que estas afirmaciones fueron hechas con el conocimiento de las informaciones falsas intencionadas y similares pueden ser hechas punibles por multa o encarcelamiento, o ambos, de acuerdo con la Sección 1001 del título 18 del Código de Estados Unidos. Las afirmaciones falsas intencionadas pueden arriesgar la

NOTARIO
CIRCUITO DE LA CAPITAL DE MICHIGAN
OTTO BACALIS GALVIS
MAY 2003
EXACTAMENTE
A LA VISTA
ESTA FOTOCOPIA DEBE
CON EL ORIGINAL

252

validez de la solicitud o cualquier patente publicada por consiguiente.

Nombre completo del Inventor: Jeffrey L. Bogdan
Ciudadanía: US
Residencia: Redmond, WA
Dirección de Residencia: 25936 NE 29th Place,
Redmond, WA 98052

(aparece firma)

Firma del Inventor

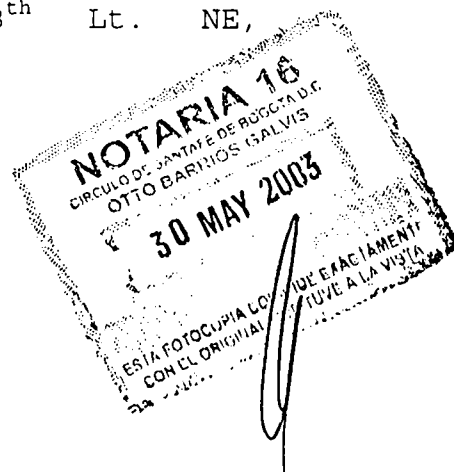
6-28-2002

Fecha

Nombre completo del Inventor: Mark J. Finocchio
Ciudadanía: US
Residencia: Redmond, WA
Dirección de Residencia: 4631 168th Lt. NE,
Redmond, WA 98052

(aparece firma)

Firma del Inventor



253

6-28-02

Fecha

Nombre completo del Inventor: Nicholas M. Kramer
Ciudadanía: US
Residencia: Seattle, WA 98103
Dirección de Residencia: 3728, Wallingford Ave.,
Seattle, WA 98103

(aparece firma)

Firma del Inventor

6/28/2002

Fecha

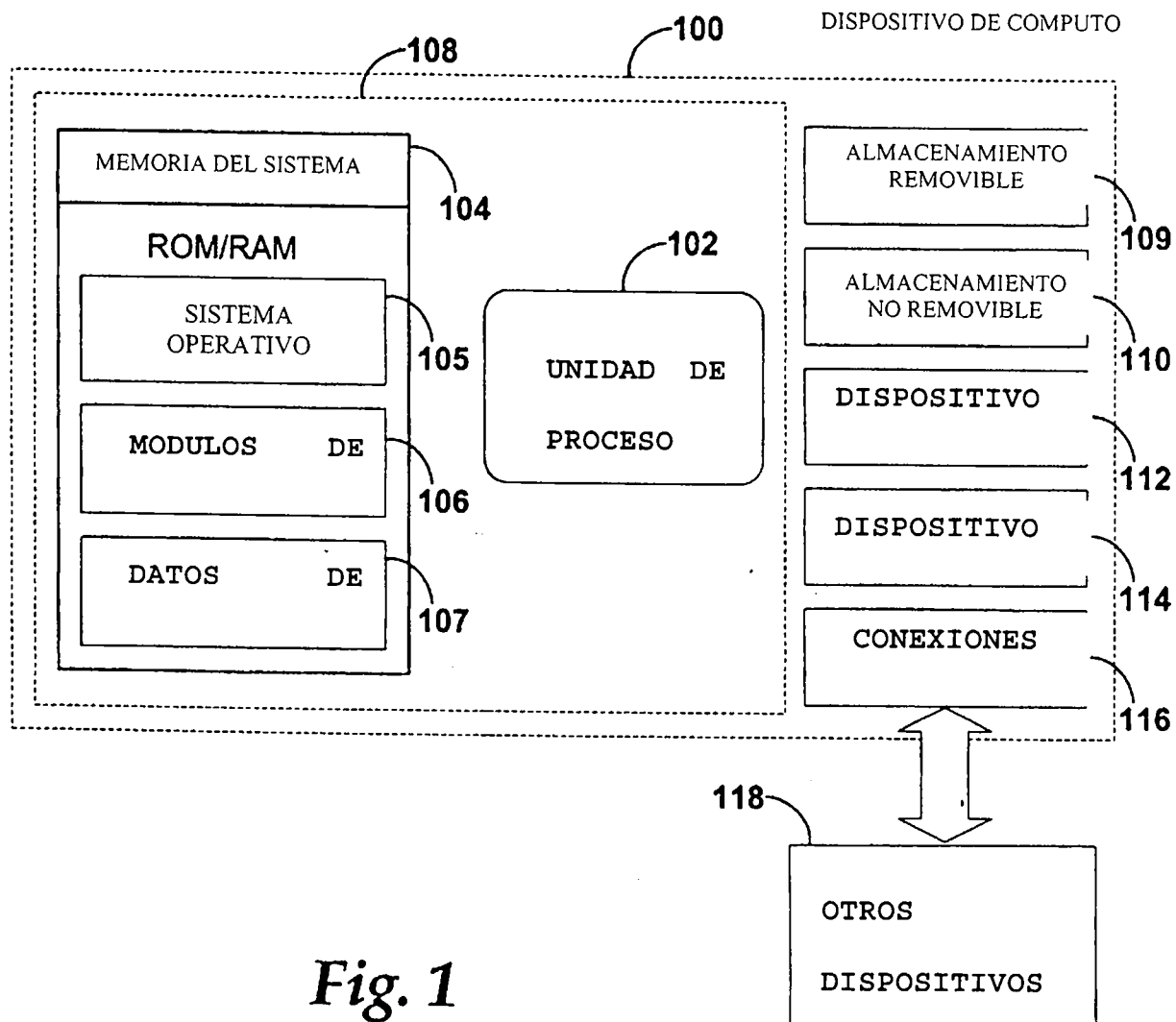


Título: SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

Inventores: Bodgan et al.

Registro No: M&G 50037.0096US01

1/6



255

Título: SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

Inventores: Bodgan et al.

Registro No: M&G 50037.0096US01

2/6

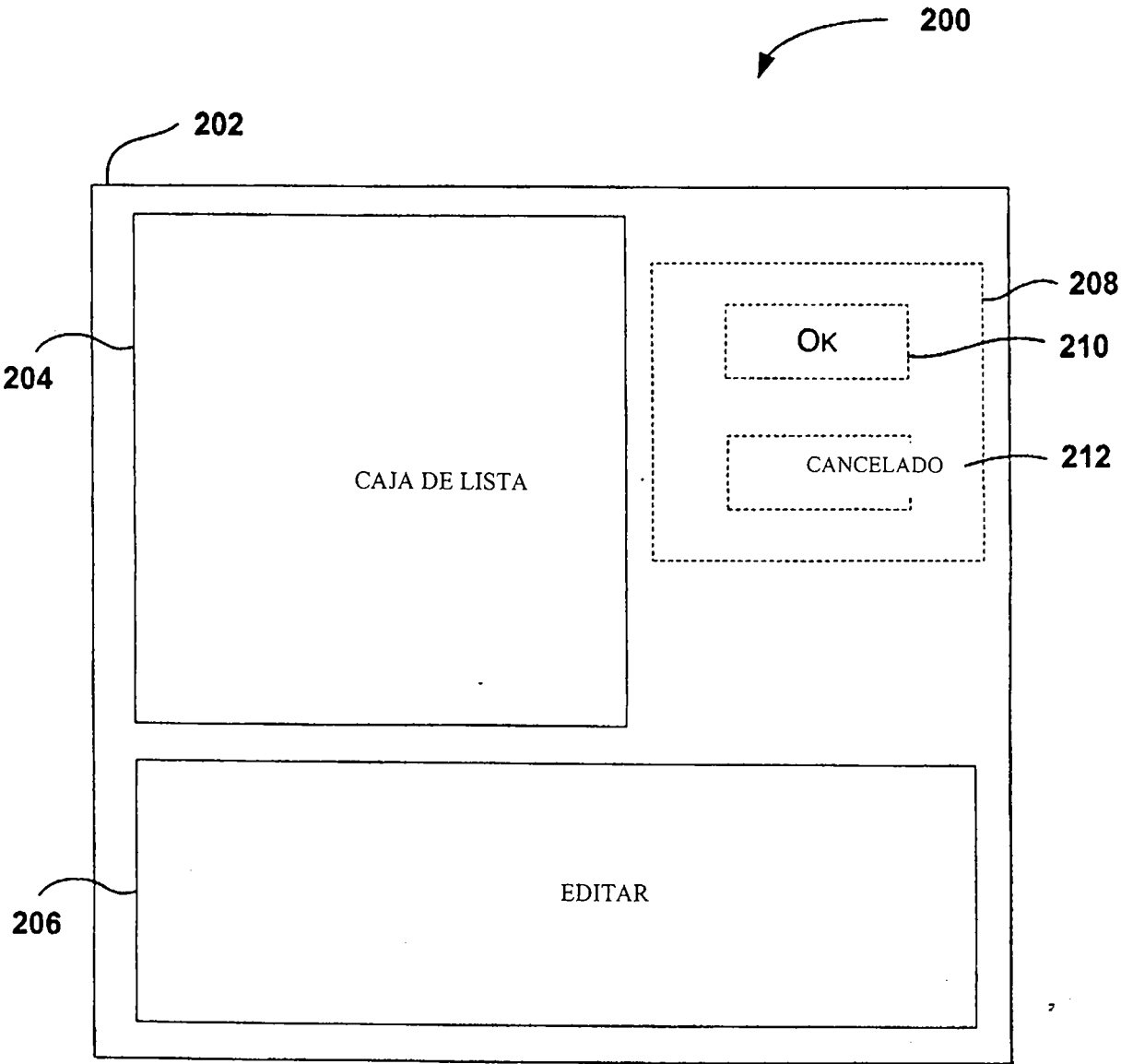
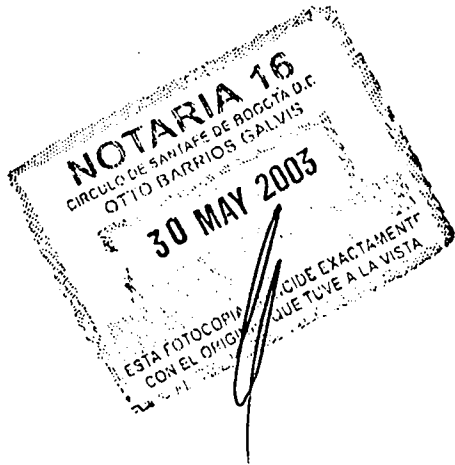


Fig. 2



3/6

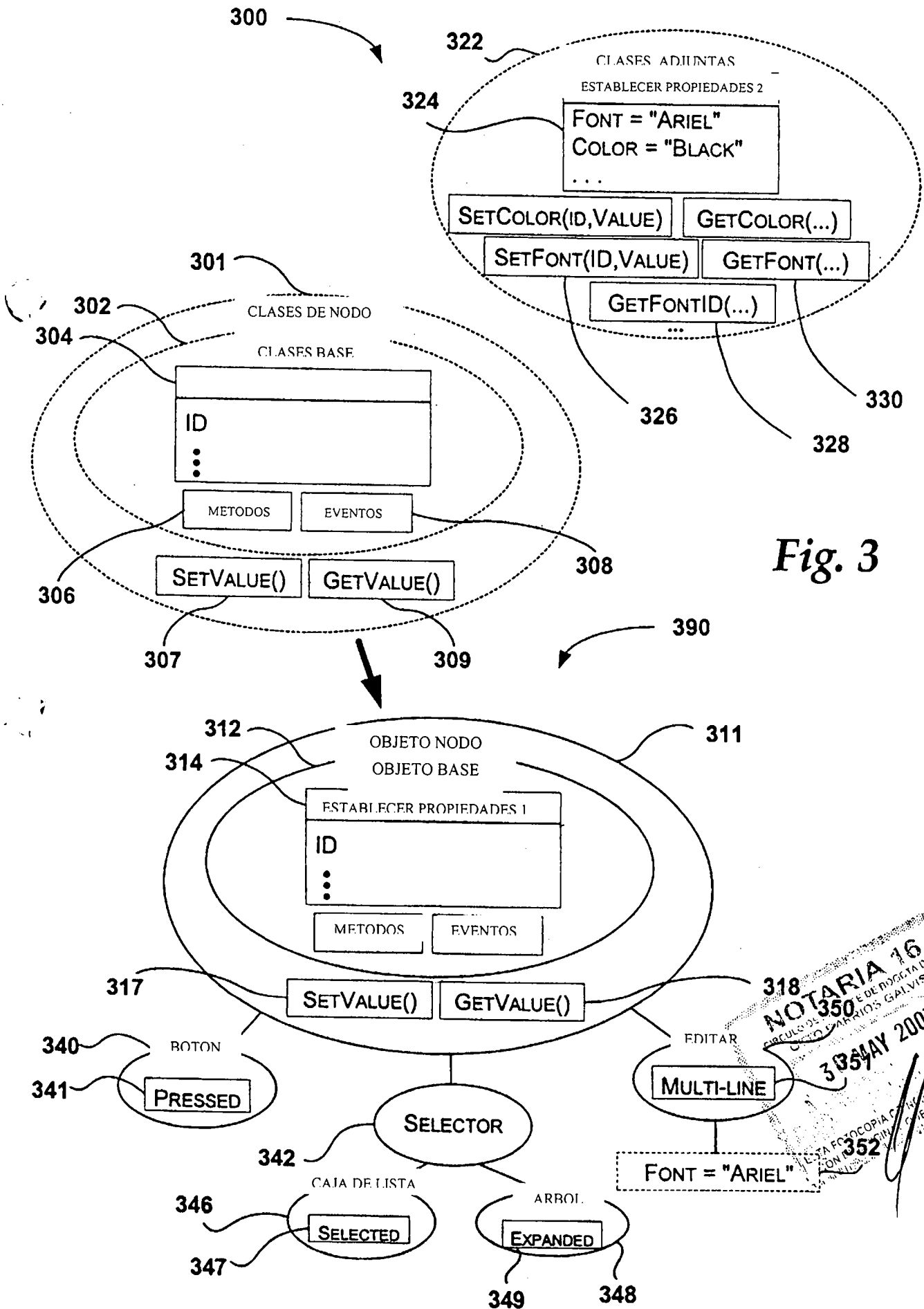


Fig. 3

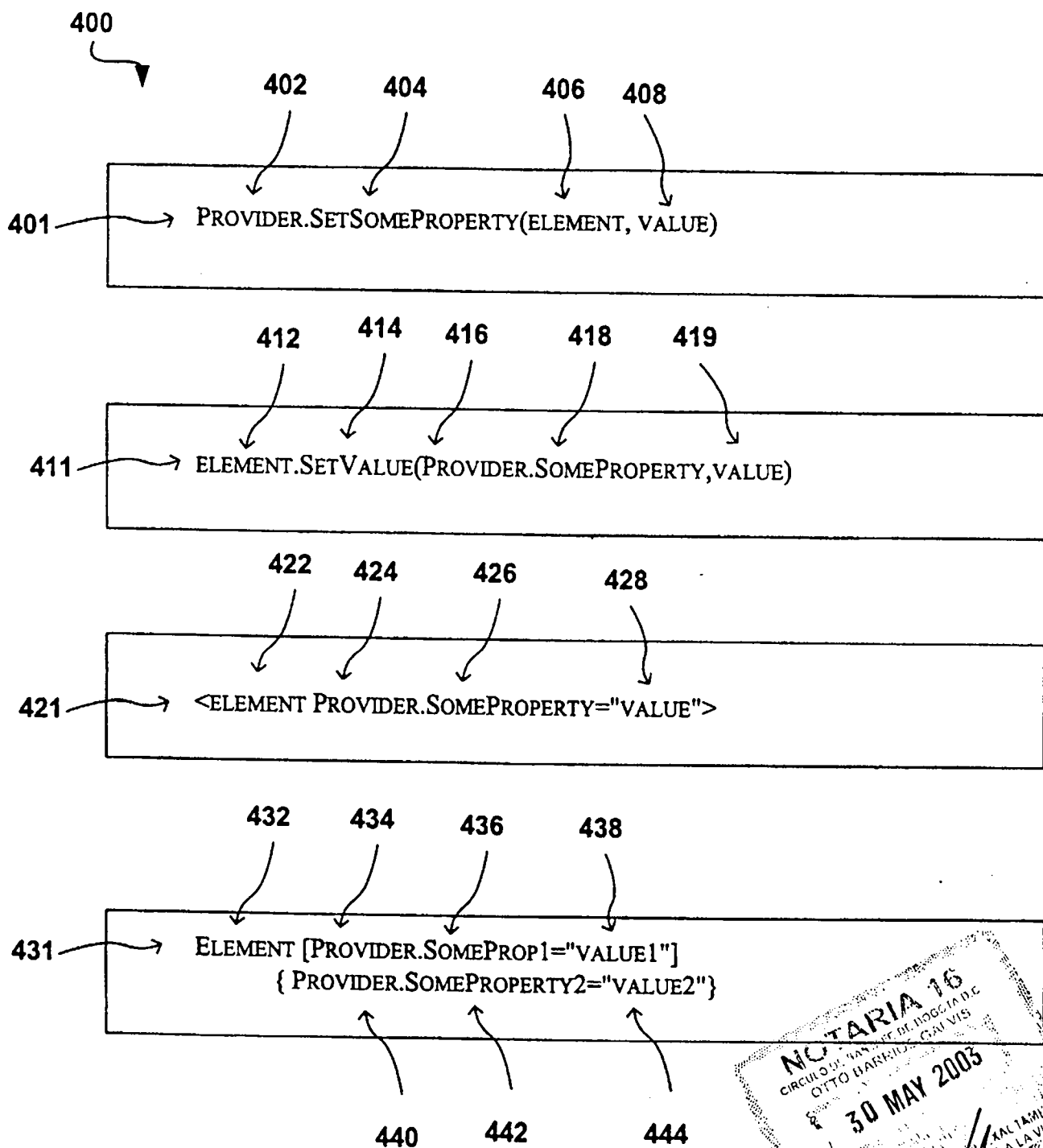


Fig. 4

258

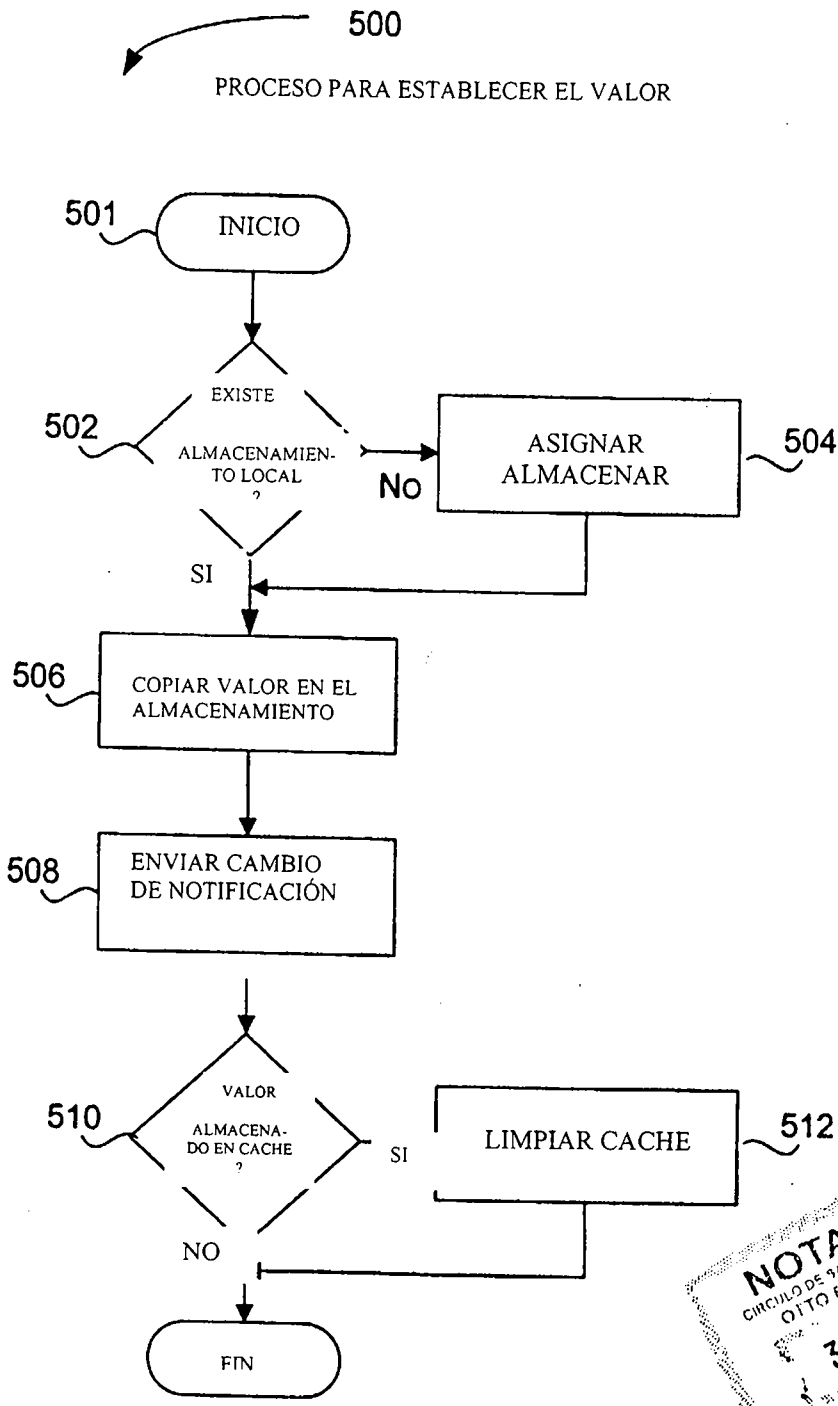
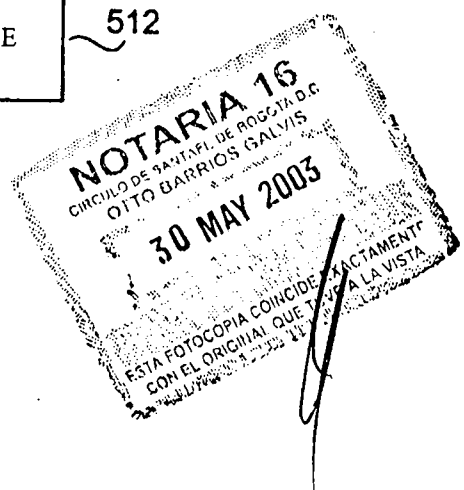


Fig. 5



Título: SISTEMA Y MÉTODO PARA ASOCIAR PROPIEDADES CON OBJETOS

Inventores: Bodgan et al.

Registro No: M&G 50037.0096US01

6/6

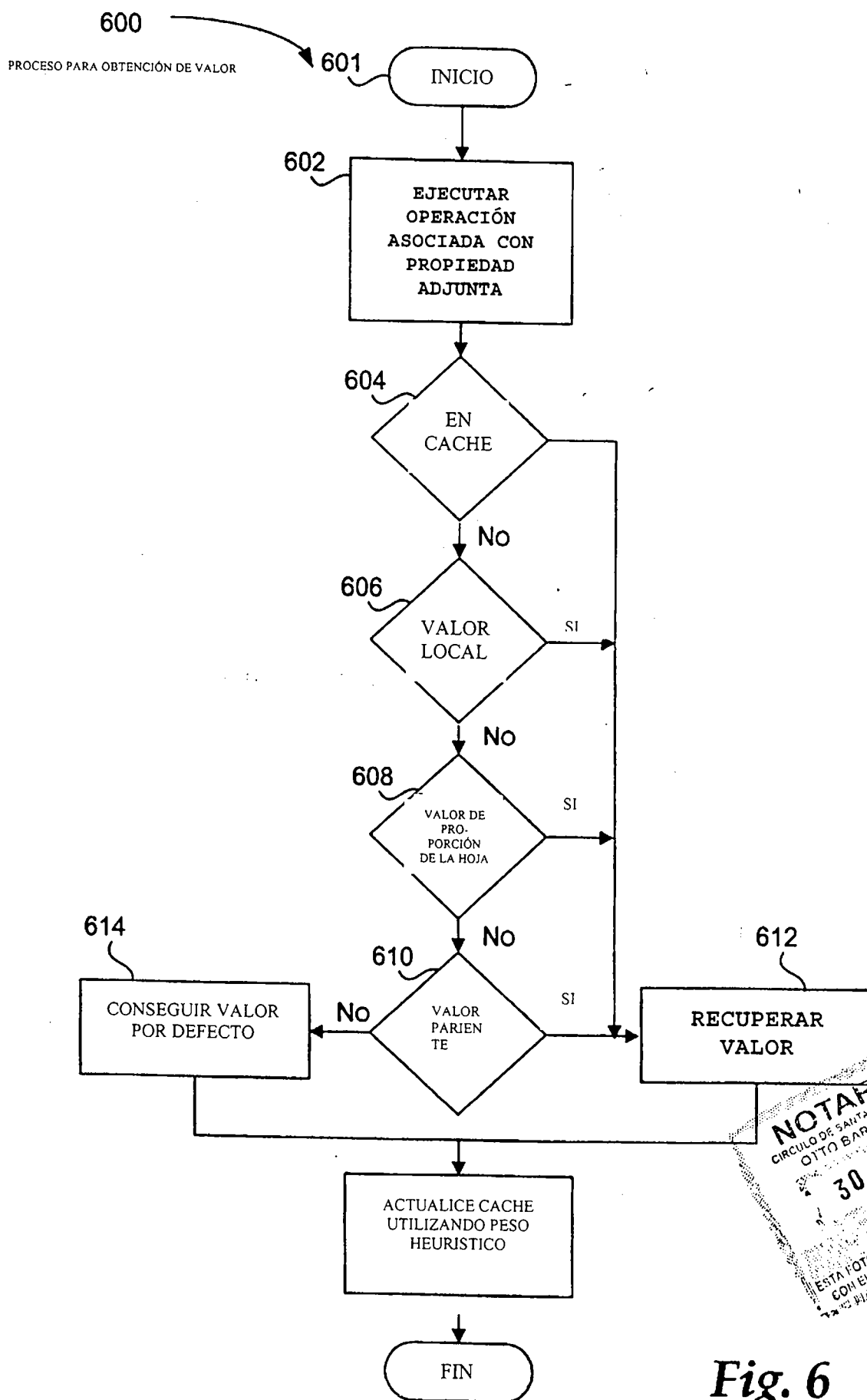


Fig. 6

Esta es una traducción fiel y completa, de la cual se deja copia para su confrontación

260

BEATRIZ EUGENIA ALARCÓN GUTIÉRREZ
TRADUCTORA E INTÉRPRETE OFICIAL
RESOLUCIÓN 2512 DE SEPT. 15/93 MINJUSTICIA

30/15/5.

BEATRIZ EUGENIA ALARCÓN GUTIÉRREZ

C.C. 39.692.345 de Usaquén

Traductora e Intérprete Oficial

Res. 2512 del 15 de septiembre de 1993 de Minjusticia

NOTARIA 16 DEL CIRCULO DE
SANTAFE DE BOGOTA, D.C.
DILIGENCIA DE PRESENTACION PERSONAL
Y RECONOCIMIENTO

El Notario ha verificado que el escrito que
se le presenta es autografo y es
de Beatriz Eugenia Alarcón Gutiérrez
identificada con C.C. 39.692.345

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
30 MAY 2003

ESTA FOTOCOPIA COINCIDE EXACTAMENTE
CON EL ORIGINAL QUE SE LE PRESENTA A LA VISTA

30/15/5.



261

Industria y Comercio
SUPERINTENDENCIA

PROTOCOLO

SUPERINDUSTRIA Y COMERCIO SISTEMA DE REGISTRO

Fecha 02/07/2003

MANTENIMIENTO DE PROTOCOLOS

Numero : 16069

Fecha : 04/08/1995

Conferido: MICROSOFT CORPORATION

Pais : US ESTADOS UNIDOS DE AMERICA

Ciudad : 234 REDMOND, WASHINGTON

Represent: 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

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

REQUISITOS NECESARIOS PARA PR

TRAMITE

USUARIO RADICADOR

PATENTE DE INVENCION ✓

MODELO DE UTILIDAD

- 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 PORSC

Fecha:



263

Industria y Comercio

SUPERINTENDENCIA

~~Folio 257~~

2020
Bogotá DC

Doctor(a)
DARIO CARDENAS NAVAS
Apoderado(a)

REFERENCIA:	Radicación No.	03-46839
	Trámite	002
	Evento	001
	Actuación	416
	Oficio No.	991
	Folios	001

Teniendo en cuenta que la solicitud de privilegio de patente que se tramita bajo el expediente indicado en la referencia, reúne los requisitos de forma establecidos en los artículos 26 y 27 de la Decisión 486 de la Comisión de la Comunidad Andina, y en las demás disposiciones vigentes sobre la materia, se envía el extracto de la solicitud a la Oficina de Comunicaciones para efecto de su publicación en la Gaceta de Propiedad Industrial.

De acuerdo con lo previsto por el artículo 40 de la Decisión 486/00, la publicación se hará a partir del **día 4 de diciembre de 2004.**

Cúmplase

Dado en Bogotá DC, a los

24 JUL. 2003

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

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