J.E

**Industria y Comercio**

SUPERINTENDENCIA

# SOLICITUD
# PATENTE DE INVENCIÓN

EXPEDIENTE N° 04-2656

TITULO INTERFACE DE PROGRAMACION PARA UNA PLATAFORMA DE COMPUTO

CLASIFICACION INTERNACIONAL G06F 13/42

SOLICITANTE MICROSOFT CORPORATION

DOMICILIO REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA

APODERADO DARIO CARDENAS NAVAS

BOGOTA, D C _____

# PETITORIO

Industria y Comercio
SUPERINTENDENCIA

SUPERINDUSTRIA Y COMERCIO
Radicacion : 04000586 00000000 Folios: 178
Fecha (AMD): 2004-09-08 12:28:39
Tramite : 002 PATENTES D 1 REGISTRO/ 411 PRESENTA
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

| | | | |
|---|---|---|---|
| **SOLICITANTE (71)** | Nombre: | **MICROSOFT CORPORATION** | **IDENTIFICACION** C.C. [ ] NIT [ ] C.E. [ ] Otro [X] Número |
| | Dirección: | One Microsoft Way Redmond, Washington 98052, Estados Unidos de America | |
| | Teléfono: | 3123600 Fax: 3122410 | |
| | E-Mail: | [illegible] | |
| | Lugar de Constitución | Estado de Washington, Estados Unidos de América | |
| | Domicilio: | REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA. | |
| **REPRESENTANTE O APODERADO (74)** | Nombre: | **DARIO CARDENAS NAVAS** | **IDENTIFICACION** C.C. [X] NIT [ ] C.E. [ ] Otro [ ] Número 17.088.629 BTA TP 1.250 C.S.J. |
| | Dirección: | Carrera 7 No 71-52 Torre B Piso 9 | |
| | Teléfono: | 3123600 Fax: 3122410 | |
| | E-Mail: | general@cardenasycardenas.com | |
| **4 INVENTOR(ES) (72)** | Nombre: | 1. Jeffrey L. Bogdan<br>2. Robert A. Relyea | **IDENTIFICACION** C.C. [ ] NIT [ ] C.E. [ ] Otro [X] Número |
| | Dirección: | 1. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America<br>2. One Microsoft Way Redmond, Washington 98052, Estados Unidos de America | |
| | Teléfono: | 3123600 Fax: 3123600 | |
| | E-Mail: | general@cardenasycardenas.com | |
| | Domicilio: | 1. WASHINGTON, ESTADOS UNIDOS DE AMERICA<br>2. WASHINGTON, ESTADOS UNIDOS DE AMERICA | |

**5 Título(54):**

**INTERFACE DE PROGRAMACION PARA UNA PLATAFORMA DE COMPUTO**

**6 Prioridad** SI [X] NO [ ]

| (33)País de Origen | (32)Fecha | (31)N° de Solicitud |
|---|---|---|
| U.S.A. | Octubre 24 2003 | 10/693.854 |

**7 Para publicar á partir de la fecha de la presente solicitud a los:**

[ ] 6 meses [ ] 12 meses [ ] 18 meses [ ] Otro

AMR/JAG/amg
No P-280

| Comprobante de Pago No.: | 04-69,416 / 04-69,417 | Fecha | Sept. 2, 2004 |
|---|---|---|---|

**ANEXOS**

- [X] Comprobante de pago de la tasa de presentación de la solicitud. (No. 04-69,416)
- [ ] Comprobante de pago de la tasa por concepto de excedente de palabras en la publicación.
- [X] Comprobante de pago de la tasa por concepto de reivindicación de prioridad. (No 04-69,417)
- [X] Poderes, si fuere el caso. (Protocolo No 16.069)
- [X] 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.
- [X] Descripción de la invención.
- [X] Una o más reivindicaciones.
- [X] 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.
- [X] Arte final 12x12 y 6x6 cm.
- [X] Resumen y extracto para publicación.



NOMBRE: DARIO CARDENAS NAVAS

FIRMA:

C.C. 17.066.629 BTA TP. 1.250 C.S.J.

3

SUPERINDUSTRIA Y COMERCIO

NIT : 800176089-2

Radicacion : 04086586 00000000 Folios: 178
Fecha (AMD): 2004-09-08 12:28:39
Tramite : 1002 PATENTES D 1 REGISTRO/ 411 PRESENTAC
Dependencia: 0605 DIVISION DE NUEVAS CREACIONES

RECIBO OFICIAL DE CAJA : 04 - 69,416
FECHA : SEPTIEMBRE 2 DE 2004

***** C O N S I G N A C I O N *****

| DEPOSITANTE | TIPO PAGO | BANCO | CUENTA | NO. PAGO | FECHA PGO | Vr. PAGO |
|---|---|---|---|---|---|---|
| CARDENAS Y CARDENAS P.I. CONSIGNACION | BANCO POPULAR | 050-00120-6 | | 23249134 | 02/09/2004 | 6,887,600.00 |

***** C O N C E P T O *****

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

SON: CUATROCIENTOS VEINTIDOS MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

INTERFACE DE PROGRAMACION PARA UNA PLATAFORMA DE COMPUTO

CARDENAS Y CARDENAS ABOGADOS

4

SUPERINDUSTRIA Y COMERCIO

NIT : 800176089-2 Radicacion : 04086586 00000000 Folios: 178

Fecha (DMA): 2004-09-06 12:28:29

Tramite : 002 PATENTES D 11 REGISTRO/ 411 PRESENTAC

Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

RECIBO OFICIAL DE CAJA : 04 - 69,417

FECHA : SEPTIEMBRE 2 DE 2004

***** C O N S I G N A C I O N *****

| DEPOSITANTE | TIPO PAGO | BANCO | CUENTA | No. PAGO | FECHA PGO | Vr. PAGO |
|---|---|---|---|---|---|---|
| CARDENAS Y CARDENAS P.I. CONSIGNACION | | BANCO POPULAR | 050-00110-6 | 23249134 | 02/09/2004 | 6,887,600.00 |

***** C O N C E P T O *****

| CANT. | RENTISTICO | CONCEPTO | | TOTAL CONCEPTO |
|---|---|---|---|---|
| 1 | 50005-01-01 SOLICITUDES | 92 | INVOCACION DE UNA PRIORIDAD EN NU | 118,000.00 |
| | | | TOTAL : $ | 118,000.00 |

SON: CIENTO DIEZ Y OCHO MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No._____

INTERFACE DE PROGRAMACION PARA UNA PLATAFORMA DE COMPUTO

CARDENAS Y CARDENAS ABOGADOS

4

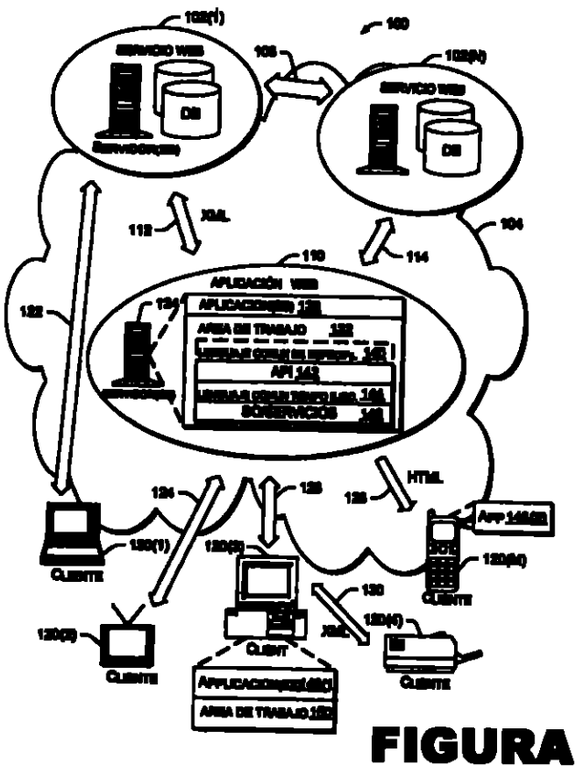



**FIGURA 1**

6

Industria y Comercio

# FORMATO DE DIGITALIZACION
## RELACION DE ANEXOS

Roypl Technologies S.A.

| Item | Expediente No | | No de Folio |
|---|---|---|---|
| | | No de Unidades | |
| 1 | CD X | 2 | 2 |
| 2 | DVD | | |
| 3 | REVISTA . | | |
| 4 | LIBRO | | |
| 5 | PERIODICO | | |
| 6 | DISQUETES | | |
| 7 | SOBRE DE MANILA | | |
| 8 | SOBRE BLANCO TAMAÑO CARTA | | |
| 9 | SOBRE BLANCO TAMAÑO OFICIO | | |
| 10 | OTROS: | | |

**Observaciones:** CD - Contienen Art Final

## TECHNICAL FIELD

This invention relates to software and to development of such software. More particularly, this invention relates to a programming interface that facilitates use of a software platform by application programs and computer hardware.

## BRIEF DESCRIPTION OF ACCOMPANYING COMPACT DISCS

Accompanying this specification is a set of three compact discs that stores a Software Development Kit (SDK) for the Microsoft® Windows® Code-Named "Longhorn" operating system. The SDK contains documentation for the Microsoft® Windows® Code-Named "Longhorn" operating system. Duplicate copies of each of these three compact discs also accompany this specification.

The first compact disc in the set of three compact discs (CD 1 of 3) includes a file folder named "lhsdk" that was created on October 22, 2003; it is 586 Mbytes in size, contains 9,692 sub-folders, and contains 44,292 sub-files. The second compact disc in the set of three compact discs (CD 2 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 605 Mbytes in size, contains 12,628 sub-folders, and contains 44,934 sub-files. The third compact disc in the set of three compact discs (CD 3 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 575 Mbytes in size, contains 9,881 sub-folders, and contains 43,630 sub-files. The files on each of these three compact discs can be executed on a Windows®-based computing device (e.g., IBM-PC, or equivalent) that executes a Windows®-brand operating system (e.g., Windows NT, Windows® 98, Windows® 2000, Windows® XP, etc.). The files on each

compact disc in this set of three compact discs are hereby incorporated by reference.

Each compact disc in the set of three compact discs itself is a CD-R, and conforms to the ISO 9660 standard. The contents of each compact disc in the set of three compact discs is in compliance with the American Standard Code for Information Interchange (ASCII).

## BACKGROUND

Very early on, computer software came to be categorized as "operating system" software or "application" software. Broadly speaking, an application is software meant to perform a specific task for the computer user such as solving a mathematical equation or supporting word processing. The operating system is the software that manages and controls the computer hardware. The goal of the operating system is to make the computer resources available to the application programmer while at the same time, hiding the complexity necessary to actually control the hardware.

The operating system makes the resources available via functions that are collectively known as the Application Program Interface or API. The term API is also used in reference to a single one of these functions. The functions are often grouped in terms of what resource or service they provide to the application programmer. Application software requests resources by calling individual API functions. API functions also serve as the means by which messages and information provided by the operating system are relayed back to the application software.

In addition to changes in hardware, another factor driving the evolution of operating system software has been the desire to simplify and speed application software development. Application software development can be a daunting task, sometimes requiring years of developer time to create a sophisticated program with millions of lines of code. For a popular operating system such as various versions of the Microsoft Windows® operating system, application software developers write thousands of different applications each year that utilize the operating system. A coherent and usable operating system base is required to support so many diverse application developers.

Often, development of application software can be made simpler by making the operating system more complex. That is, if a function may be useful to several different application programs, it may be better to write it once for inclusion in the operating system, than requiring dozens of software developers to write it dozens of times for inclusion in dozens of different applications. In this manner, if the operating system supports a wide range of common functionality required by a number of applications, significant savings in applications software development costs and time can be achieved.

Regardless of where the line between operating system and application software is drawn, it is clear that for a useful operating system, the API between the operating system and the computer hardware and application software is as important as efficient internal operation of the operating system itself.

When developing applications, developers use a variety of tools to generate graphical items and other content. Additional tools are available to arrange graphical items and other data to be displayed or rendered. These tools are typically created by different entities or different tool developers. As a result, the



10

tools do not provide a consistent programming environment. Thus, a developer using these different tools needs to learn how to utilize each of the tools and attempt to make them communicate with one another. These activities can be tedious and time consuming, taking time away from the actual development task at hand.

Over the past few years, the universal adoption of the Internet, and networking technology in general, has changed the landscape for computer software developers. Traditionally, software developers focused on single-site software applications for standalone desktop computers, or LAN-based computers that were connected to a limited number of other computers via a local area network (LAN). Such software applications were typically referred to as "shrink wrapped" products because the software was marketed and sold in a shrink-wrapped package. The applications utilized well-defined APIs to access the underlying operating system of the computer.

As the Internet evolved and gained widespread acceptance, the industry began to recognize the power of hosting applications at various sites on the World Wide Web (or simply the "Web"). In the networked world, clients from anywhere could submit requests to server-based applications hosted at diverse locations and receive responses back in fractions of a second. These Web applications, however, were typically developed using the same operating system platform that was originally developed for standalone computing machines or locally networked computers. Unfortunately, in some instances, these applications do not adequately transfer to the distributed computing regime. The underlying platform was simply not constructed with the idea of supporting limitless numbers of interconnected computers.

To accommodate the shift to the distributed computing environment being ushered in by the Internet, Microsoft Corporation developed a network software platform known as the ".NET" Framework (read as "Dot Net"). Microsoft® .NET is software for connecting people, information, systems, and devices. The platform allows developers to create Web services that will execute over the Internet. This dynamic shift was accompanied by a set of API functions for Microsoft's .NET™ Framework.

As use of the .NET™ Framework has become increasingly common, ways to increase the efficiency and/or performance of the platform have been identified. The inventors have developed a unique set of API functions to allow for such increased efficiency and/or performance.

## SUMMARY

A programming interface, such as an API, provides functions for generating applications, documents, media presentations and other content. These functions allow developers to obtain services from an operating system, object model service, or other system or service. In one embodiment, the functions allow a developer to generate a graphical user interface.

## BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like features.

Fig. 1 illustrates a network architecture in which clients access Web services over the Internet using conventional protocols.

Fig. 2 is a block diagram of a software architecture for a network platform, which includes an application program interface (API).

Fig. 3 is a block diagram of the presentation subsystem supported by the API, as well as function classes of the various API functions.

Fig. 4 is a block diagram of an exemplary computer that may execute all or part of the software architecture.

Figs. 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 illustrate various example implementations of a programming interface.

## DETAILED DESCRIPTION

This disclosure addresses an application program interface (API) for a network platform upon which developers can build Web applications and services. More particularly, an exemplary API is described for operating systems that make use of a network platform, such as the .NET™ Framework created by Microsoft Corporation. The .NET™ Framework is a software platform for Web services and Web applications implemented in the distributed computing environment. It represents the next generation of Internet computing, using open communication standards to communicate among loosely coupled Web services that are collaborating to perform a particular task.

In the described implementation, the network platform utilizes XML (extensible markup language), an open standard for describing data. XML is managed by the World Wide Web Consortium (W3C). XML is used for defining data elements on a Web page and business-to-business documents. XML uses a similar tag structure as HTML; however, whereas HTML defines how elements are displayed, XML defines what those elements contain. HTML uses predefined

tags, but XML allows tags to be defined by the developer of the page. Thus, virtually any data items can be identified, allowing Web pages to function like database records. Through the use of XML and other open protocols, such as Simple Object Access Protocol (SOAP), the network platform allows integration of a wide range of services that can be tailored to the needs of the user. Although the embodiments described herein are described in conjunction with XML and other open standards, such are not required for the operation of the claimed invention. Other equally viable technologies will suffice to implement the inventions described herein.

As used herein, the phrase application program interface or API includes traditional interfaces that employ method or function calls, as well as remote calls (e.g., a proxy, stub relationship) and SOAP/XML invocations.

It should be appreciated that in some of namespace descriptions below, descriptions of certain classes, interfaces, enumerations and delegates are left blank. More complete descriptions of these classes, interfaces, enumerations and delegates can be found in the subject matter of the compact discs that store the SDK referenced above.

## EXEMPLARY NETWORK ENVIRONMENT

Fig. 1 shows a network environment 100 in which a network platform, such as the .NET™ Framework, may be implemented. The network environment 100 includes representative Web services 102(1), ..., 102(N), which provide services that can be accessed over a network 104 (e.g., Internet). The Web services, referenced generally as number 102, are programmable application components that are reusable and interact programmatically over the network 104, typically

through industry standard Web protocols, such as XML, SOAP, WAP (wireless application protocol), HTTP (hypertext transport protocol), and SMTP (simple mail transfer protocol) although other means of interacting with the Web services over the network may also be used, such as Remote Procedure Call (RPC) or object broker type technology. A Web service can be self-describing and is often defined in terms of formats and ordering of messages.

Web services 102 are accessible directly by other services (as represented by communication link 106) or a software application, such as Web application 110 (as represented by communication links 112 and 114). Each Web service 102 is illustrated as including one or more servers that execute software to handle requests for particular services. Such services often maintain databases that store information to be served back to requesters. Web services may be configured to perform any one of a variety of different services. Examples of Web services include login verification, notification, database storage, stock quoting, location directories, mapping, music, electronic wallet, calendar/scheduler, telephone listings, news and information, games, ticketing, and so on. The Web services can be combined with each other and with other applications to build intelligent interactive experiences.

The network environment 100 also includes representative client devices 120(1), 120(2), 120(3), 120(4), ..., 120(M) that utilize the Web services 102 (as represented by communication link 122) and/or the Web application 110 (as represented by communication links 124, 126, and 128). The clients may communicate with one another using standard protocols as well, as represented by an exemplary XML link 130 between clients 120(3) and 120(4).

The client devices, referenced generally as number 120, can be implemented many different ways. Examples of possible client implementations include, without limitation, portable computers, stationary computers, tablet PCs, televisions/set-top boxes, wireless communication devices, personal digital assistants, gaming consoles, printers, photocopiers, and other smart devices.

The Web application 110 is an application designed to run on the network platform and may utilize the Web services 102 when handling and servicing requests from clients 120. The Web application 110 is composed of one or more software applications 130 that run atop a programming framework 132, which are executing on one or more servers 134 or other computer systems. Note that a portion of Web application 110 may actually reside on one or more of clients 120. Alternatively, Web application 110 may coordinate with other software on clients 120 to actually accomplish its tasks.

The programming framework 132 is the structure that supports the applications and services developed by application developers. It permits multi-language development and seamless integration by supporting multiple languages. It supports open protocols, such as SOAP, and encapsulates the underlying operating system and object model services. The framework provides a robust and secure execution environment for the multiple programming languages and offers secure, integrated class libraries.

The framework 132 is a multi-tiered architecture that includes an application program interface (API) layer 142, a common language runtime (CLR) layer 144, and an operating system/services layer 146. This layered architecture allows updates and modifications to various layers without impacting other portions of the framework. A common language specification (CLS) 140 allows

designers of various languages to write code that is able to access underlying library functionality. The specification 140 functions as a contract between language designers and library designers that can be used to promote language interoperability. By adhering to the CLS, libraries written in one language can be directly accessible to code modules written in other languages to achieve seamless integration between code modules written in one language and code modules written in another language. One exemplary detailed implementation of a CLS is described in an ECMA standard created by participants in ECMA TC39/TG3. The reader is directed to the ECMA web site at www.ecma.ch.

The API layer 142 presents groups of functions that the applications 130 can call to access the resources and services provided by layer 146. By exposing the API functions for a network platform, application developers can create Web applications for distributed computing systems that make full use of the network resources and other Web services, without needing to understand the complex interworkings of how those network resources actually operate or are made available. Moreover, the Web applications can be written in any number of programming languages, and translated into an intermediate language supported by the common language runtime 144 and included as part of the common language specification 140. In this way, the API layer 142 can provide methods for a wide and diverse variety of applications.

Additionally, the framework 132 can be configured to support API calls placed by remote applications executing remotely from the servers 134 that host the framework. Representative applications 148(1) and 148(2) residing on clients 120(3) and 120(M), respectively, can use the API functions by making calls directly, or indirectly, to the API layer 142 over the network 104.

The framework may also be implemented at the clients. Client 120(3) represents the situation where a framework 150 is implemented at the client. This framework may be identical to server-based framework 132, or modified for client purposes. Alternatively, the client-based framework may be condensed in the event that the client is a limited or dedicated function device, such as a cellular phone, personal digital assistant, handheld computer, or other communication/computing device.

DEVELOPERS' PROGRAMMING FRAMEWORK

Fig. 2 shows the programming framework 132 in more detail. The common language specification (CLS) layer 140 supports applications written in a variety of languages 130(1), 130(2), 130(3), 130(4), ..., 130(K). Such application languages include Visual Basic, C++, C#, COBOL, Jscript, Perl, Eiffel, Python, and so on. The common language specification 140 specifies a subset of features or rules about features that, if followed, allow the various languages to communicate. For example, some languages do not support a given type (e.g., an "int*" type) that might otherwise be supported by the common language runtime 144. In this case, the common language specification 140 does not include the type. On the other hand, types that are supported by all or most languages (e.g., the "int[]" type) is included in common language specification 140 so library developers are free to use it and are assured that the languages can handle it. This ability to communicate results in seamless integration between code modules written in one language and code modules written in another language. Since different languages are particularly well suited to particular tasks, the seamless integration between languages allows a developer to select a particular language

for a particular code module with the ability to use that code module with modules written in different languages. The common language runtime 144 allow seamless multi-language development, with cross language inheritance, and provide a robust and secure execution environment for the multiple programming languages. For more information on the common language specification 140 and the common language runtime 144, the reader is directed to co-pending applications entitled "Method and System for Compiling Multiple Languages", filed 6/21/2000 (serial number 09/598.105) and "Unified Data Type System and Method" filed 7/10/2000 (serial number 09/613,289), which are incorporated by reference.

The framework 132 encapsulates the operating system 146(1) (e.g., Windows®-brand operating systems) and object model services 146(2) (e.g., Component Object Model (COM) or Distributed COM). The operating system 146(1) provides conventional functions, such as file management, notification, event handling, user interfaces (e.g., windowing, menus, dialogs, etc.), security, authentication, verification, processes and threads, memory management, and so on. The object model services 146(2) provide interfacing with other objects to perform various tasks. Calls made to the API layer 142 are handed to the common language runtime layer 144 for local execution by the operating system 146(1) and/or object model services 146(2).

The API 142 groups API functions into multiple namespaces. Namespaces essentially define a collection of classes, interfaces, delegates, enumerations, and structures, which are collectively called "types", that provide a specific set of related functionality. A class represents managed heap allocated data that has reference assignment semantics. A delegate is an object oriented function pointer. An enumeration is a special kind of value type that represents named constants. A

structure represents static allocated data that has value assignment semantics. An interface defines a contract that other types can implement.

By using namespaces, a designer can organize a set of types into a hierarchical namespace. The designer is able to create multiple groups from the set of types, with each group containing at least one type that exposes logically related functionality. In the exemplary implementation, the API 142 is organized to include three root namespaces. It should be noted that although only three root namespaces are illustrated in Fig. 2, additional root namespaces may also be included in API 142. The three root namespaces illustrated in API 142 are: a first namespace 200 for a presentation subsystem (which includes a namespace 202 for a user interface shell), a second namespace 204 for web services, and a third namespace 206 for a file system. Each group can then be assigned a name. For instance, types in the presentation subsystem namespace 200 can be assigned the name "Windows", and types in the file system namespace 206 can be assigned names "Storage". The named groups can be organized under a single "global root" namespace for system level APIs, such as an overall System namespace. By selecting and prefixing a top level identifier, the types in each group can be easily referenced by a hierarchical name that includes the selected top level identifier prefixed to the name of the group containing the type. For instance, types in the file system namespace 206 can be referenced using the hierarchical name "System.Storage". In this way, the individual namespaces 200, 204, and 206 become major branches off of the System namespace and can carry a designation where the individual namespaces are prefixed with a designator, such as a "System." prefix.

The presentation subsystem namespace 200 pertains to programming and content development. It supplies types that allow for the generation of applications, documents, media presentations and other content. For example, presentation subsystem namespace 200 provides a programming model that allows developers to obtain services from the operating system 146(1) and/or object model services 146(2).

The shell namespace 202 pertains to user interface functionality. It supplies types that allow developers to embed user interface functionality in their applications, and further allows developers to extend the user interface functionality.

The web services namespace 204 pertains to an infrastructure for enabling creation of a wide variety of applications, e.g. applications as simple as a chat application that operates between two peers on an intranet, and/or as complex as a scalable Web service for millions of users. The described infrastructure is advantageously highly variable in that one need only use those parts that are appropriate to the complexity of a particular solution. The infrastructure provides a foundation for building message-based applications of various scale and complexity. The infrastructure or framework provides APIs for basic messaging, secure messaging, reliable messaging and transacted messaging. In the embodiment described below, the associated APIs have been factored into a hierarchy of namespaces in a manner that has been carefully crafted to balance utility, usability, extensibility and versionability.

The file system namespace 206 pertains to storage. It supplies types that allow for information storage and retrieval.

In addition to the framework 132, programming tools 210 are provided to assist the developer in building Web services and/or applications. One example of the programming tools 210 is Visual Studio™, a multi-language suite of programming tools offered by Microsoft Corporation.

ROOT API NAMESPACES

Fig. 3 shows a portion of the presentation subsystem 200 in more detail. In one embodiment, the namespaces are identified according to a hierarchical naming convention in which strings of names are concatenated with periods. For instance, the presentation subsystem namespace 200 is identified by the root name "System.Windows". Within the "Sytem.Windows" namespace is another namespace for various controls, identified as "System.Windows.Controls", which further identifies another namespace for primitives (not shown) known as "System.Windows.Controls.Primitives". With this naming convention in mind, the following provides a general overview of selected namespaces of the API 142, although other naming conventions could be used with equal effect.

As shown in Fig. 3, the presentation subsystem 200 includes multiple namespaces. The namespaces shown in Fig. 3 represent a particular embodiment of the presentation subsystem 200. Other embodiments of the presentation subsystem 200 may include one or more additional namespaces or may omit one or more of the namespaces shown in Fig. 3.

The presentation subsystem 200 is the root namespace for much of the presentation functionality of the API 142. A controls namespace 310 includes controls used to build a display of information, such as a user interface, and classes that allow a user to interact with an application. Example controls include

"Button" that creates a button on the display, "RadioButton" that generates a radio-style button on the display, "Menu" that creates a menu on the display, "ToolBar" that creates a toolbar on the display, "Image" that generates an image on the display and "TreeView" that creates a hierarchical view of information.

Certain controls are created by nesting and arranging multiple elements. The controls have a logical model that hides the elements used to create the controls, thereby simplifying the programming model. The controls can be styled and themed by a developer or a user (e.g., by customizing the appearance and behavior of user interface buttons). Some controls have addressable components that allow an individual to adjust the style of individual controls. Additionally, the controls can be sub-classed and extended by application developers and component developers. The controls are rendered using vector graphics such that they can be resized to fit the requirements of a particular interface or other display. The controls are capable of utilizing animation to enhance, for example, the interactive feel of a user interface and to show actions and reactions.

The controls namespace 310 includes one or more panels, which are controls that measure and arrange their children (e.g., nested elements). For example, a "DockPanel" panel arranges children by docking each child to the top, left, bottom or right side of the display, and fills-in the remaining space with other data. A particular panel may dock menus and toolbars to the top of the display, a status bar to the bottom of the display, a folder list to the left side of the display, and fills the rest of the space with a list of messages.

As mentioned above, System.Windows.Controls.Primitives is a namespace that includes multiple controls that are components typically used by developers of the controls in the System.Windows.Controls namespace and by developers

creating their own controls. Examples of these components include "Thumb and RepeatButton". "ScrollBar", another component, is created using four repeat buttons (one for "line up", one for "line down", one for "page up", and one for "page down") and a "Thumb" for dragging the current view to another location in the document. In another example, "ScrollViewer" is a control created using two "ScrollBars" and one "ScrollArea" to provide a scrollable area.

The following list contains example classes exposed by the System.Windows.Controls namespace. These classes allow a user to interact with, for example, an application through various input and output capabilities as well as additional display capabilities.

- AccessKey – AccessKey is a FrameworkElement element that wraps a character, indicating that it is to receive keyboard cue decorations denoting the character as a keyboard mnemonic. By default, the keyboard cue decoration is an underline.
- Audio – Audio Element.
- Border – Draws a border, background, or both around another element.
- Button – Represents the standard button component that inherently reacts to the Click event.
- Canvas – Defines an area within which a user can explicitly position child elements by coordinates relative to the Canvas area.

- CheckBox – Use a CheckBox to give the user an option, such as true/false. CheckBox allows the user to choose from a list of options. CheckBox controls let the user pick a combination of options.

- CheckedChangedEventArgs – This CheckedChangedEventArgs class contains additional information about the CheckedChangedEvent event.

- CheckStateChangedEventArgs – This CheckStateChangedEventArgs class contains additional information about the CheckStateChangedEvent event.

- ClickEventArgs – Contains information about the Click event.

- ColumnStyle – Represents a changeable ColumnStyle object.

- ColumnStyles – Changeable pattern IList object that is a collection of Changeable elements.

- ComboBox – ComboBox control.

- ComboBoxItem – Control that implements a selectable item inside a ComboBox.

- ContactPickerDialog – Allows a user to select one or more contacts.

- ContactPropertyRequest – Allows an application to request information about a contact property through a ContactPickerDialog. This class cannot be inherited.

- ContactPropertyRequest Collection – Represents a collection of ContactPropertyRequest objects.

- ContactSelection – Information about a selected contact from Microsoft® Windows® File System, code-named "WinFS" or Microsoft Active Directory®.

- ContactSelectionCollection – Represents a collection of ContactSelection objects.

- ContactTextBox – An edit control that supports picking contacts or properties of contacts.

- ContactTextBoxSelectionChangedEventArgs – Arguments for the ContactTextBoxSelectionChanged event.

- ContactTextBoxTextChangedEventArgs – Arguments for the ContactTextBoxTextChanged event.

- ContactTextBoxTextResolvedEventArgs – Arguments for the TextResolvedToContact event.

- ContentChangedEventArgs – The event arguments for ContentChangedEvent.

- ContentControl – The base class for all controls with a single piece of content.

- ContentPresenter – ContentPresenter is used within the style of a content control to denote the place in the control's visual tree (chrome template) where the content is to be added.

- ContextMenu – Control that defines a menu of choices for users to invoke.

- ContextMenuEventArgs – The data sent on a ContextMenuEvent.
- Control – Represents the base class for all user-interactive elements. This class provides a base set of properties for its subclasses.
- Decorator – Base class for elements that apply effects onto or around a single child element, such as Border.
- DockPanel – Defines an area within which you can arrange child elements either horizontally or vertically, relative to each other.
- DragDeltaEventArgs – This DragDeltaEventArgs class contains additional information about the DragDeltaEvent event.
- FixedPanel – FixedPanel is the root element used in fixed-format documents to contain fixed pages for pagination. FixedPanel displays paginated content one page at a time or as a scrollable stack of pages.
- FlowPanel – FlowPanel is used to break, wrap, and align content that exceeds the length of a single line. FlowPanel provides line-breaking and alignment properties that can be used when the flow of the container's content, Text for example, is likely to exceed the length of a single line.
- Frame – An area that can load the contents of another markup tree.
- Generator – Generator is the object that generates a UI on behalf of an ItemsControl, working under the supervision of a GeneratorFactory.
- GeneratorFactory – A GeneratorFactory is responsible for generating the UI on behalf of an ItemsControl. It maintains the association between the

items in the control's ItemsCollection (flattened view) and the corresponding UIElements. The control's item-container can ask the factory for a Generator, which does the actual generation of UI.

- GridPanel – Defines a grid area consisting of columns and rows.
- HeaderItemsControl – The base class for all controls that contain multiple items and have a header.
- HorizontalScrollBar – The Horizontal ScrollBar class.
- HorizontalSlider – The Horizontal Slider class.
- HyperLink – The HyperLink class implements navigation control. The default presenter is TextPresenter.
- Image – Provides an easy way to include an image in a document or an application.
- IncludeContactEventArgs – Arguments passed to handlers of the ContactPickerDialog.IncludeContact event.
- ItemCollection – Maintains a collection of discrete items within a control. Provides methods and properties that enable changing the collection contents and obtaining data about the contents.
- ItemsChangedEventArgs – The ItemsChanged event is raised by a GeneratorFactory to inform layouts that the items collection has changed.
- ItemsControl – The base class for all controls that have multiple children.

- ItemsView – ItemsView provides a flattened view of an ItemCollection.
- KeyboardNavigation – KeyboardNavigation class provide methods for logical (Tab) and directional (arrow) navigation between focusable controls.
- ListBox – Control that implements a list of selectable items.
- ListItem – Control that implements a selectable item inside a ListBox.
- Menu – Control that defines a menu of choices for users to invoke.
- MenuItem – A child item of Menu. MenuItems can be selected to invoke commands. MenuItems can be separators. MenuItems can be headers for submenus. MenuItems can be checked or unchecked.
- PageViewer – Represents a document-viewing composite control that contains a pagination control, a toolbar, and a page bar control.
- PaginationCompleteEventArgs – The event arguments for the PaginationCompleteEvent.
- PaginationProgressEventArgs – The event arguments for the PaginationProgressEvent.
- Pane – Provides a way to define window properties in a markup language (e.g., "XAML") without launching a new window.
- Panel – Provides a base class for all Panel elements. In order to instantiate a Panel element, use the derived concrete class.

- RadioButton – RadioButton implements an option button with two states: true or false.

- RadioButtonList – This control serves as a grouping control for RadioButtons and is the piece that handles RadioButton mutual exclusivity. The RadioButtonList inherits from Selector. The RadioButtonList is essentially a Single SelectionMode Selector and the concept of Selection (from Selector) is keyed off of the Checked property of the RadioButton it is grouping.

- RowStyle – Changeable pattern Changeable elements.

- RowStyles – Changeable pattern IList object that is a collection of Changeable elements.

- ScrollChangeEventArgs – The ScrollChangeEventsArgs describe a change in scrolling state.

- ScrollViewer –

- SelectedItemsCollection – A container for the selected items in a Selector.

- SelectionChangedEventArgs – The inputs to a selection changed event handler.

- SimpleText – SimpleText is a lightweight, multi-line, single-format text element intended for use in user interface (UI) scenarios. SimpleText exposes several of the same formatting properties as Text and can often be used for a performance gain at the cost of some versatility.

- StyleSelector – StyleSelector allows the app writer to provide custom style selection logic. For example, with a class Bug as the Content, use a particular style for Pri1 bugs and a different style for Pri2 bugs. An application writer can override the SelectStyle method in a derived selector class and assign an instance of this class to the StyleSelector property on ContentPresenter class.

- Text – Represents a Text control that enables rendering of multiple formats of Text. Text is best used within an application UI; more advanced text scenarios benefit from the additional feature set of TextPanel. In most cases where relatively simple text support is required, Text is the preferred element because of its lightweight nature and range of features.

- TextBox – Represents the control that provides an editable region that accepts text input.

- TextChangedEventArgs – The TextChangedEventArgs class represents a type of RoutedEventArgs that are relevant to events raised by TextRange.SetText().

- TextPanel – Formats, sizes, and draws text. TextPanel supports multiple lines of text and multiple text formats.

- ToolTip – A control to display information when the user hovers over a control.

- ToolTipEventArgs – The data sent on a ToolTipEvent.

- TransformDecorator – TransformDecorator contains a child and applies a specified transform to it. TransformDecorator implements logic to measure and arrange the child in its local (pre-transform) coordinates such that after the transform, the child fits tightly within the decorator's space and uses maximal area. The child therefore needs to have no knowledge that a transform has been applied to it.

- UIElementCollection – A UIElementCollection is a ordered collection of UIElements.

- ValueChangedEventArgs – This ValueChangedEventArgs class contains additional information about the ValueChangedEvent event.

- VerticalScrollBar – The Vertical ScrollBar class.

- VerticalSlider – The Vertical Slider class.

- Video – Plays a streaming video or audio file in a specified rectangle within the current user coordinate system.

- VisibleChangedEventArgs – The VisibleChangedEventArgs class contains additional information about the VisibleChangedEvent event.

The System.Windows.Controls namespace also contains various enumerations. The following list contains example enumerations associated with the System.Windows.Controls namespace.

- CharacterCase – Specifies the case of characters in a TextBox control when the text is typed.
- CheckState – Specifies the state of a control, such as a check box, that can be checked, unchecked, or set to an indeterminate state.
- ClickMode – Specifies when the Click event should fire.
- ContactControlPropertyPosition – Controls the position and display of the property of the contact.
- ContactPickerDialogLayout – Specifies how the ContactPickerDialog should display selected properties.
- ContactPropertyCategory – Specifies which value to treat as the default in the case where a property has multiple values for the user could choose from. For example, if "Work" is specified as the preferred category when requesting a phone number property from the ContactPickerDialog, and the user selects a contact with both a work and home phone number, the work phone number appears as the default selection. The user can then use the UI to choose the home phone number instead.
- ContactPropertyType – Specifies a property of a contact that the ContactPickerDialog can ask the user for.
- ContactType – Specifies which contact types to display in the ContactPickerDialog.
- Direction – This enumeration is used by the GeneratorFactory and Generator to specify the direction in which the generator produces UI.

- Dock – Specifies the Dock position of a child element within a DockPanel.
- GeneratorStatus – This enumeration is used by the GeneratorFactory to indicate its status.
- KeyNavigationMode – The type of TabNavigation property specify how the container will move the focus when Tab navigation occurs.
- MenuItemBehavior – Defines the different behaviors that a MenuItem could have.
- MenuItemType – Defines the different placement types of MenuItems.
- Orientation – Slider orientation types.
- PageViewerFit – Selects how pages should be fit into the PageViewer's Client area.
- PageViewerMode – Selects the current PageViewer mode Reflected in the mode dropdown.
- ScrollerVisibility – ScrollerVisibilty defines the visiblity behavior of a scrollbar.
- SelectionMode – Specifies the selection behavior for the ListBox.

"Position" is an example structure associated with the System.Windows.Controls namespace. A user of the Generator describes positions using this structure. For example: To start generating forward from the

beginning of the item list, specify position (-1, 0) and direction Forward. To start generating backward from the end of the list, specify position (-1, 0) and direction Backward. To generate the items after the element with index k, specify position (k, 0) and direction Forward.

The following list contains example delegates associated with the System.Windows.Controls namespace.

- CheckedChangedEventHandler – This delegate is used by handlers of the CheckedChangedEvent event.
- CheckStateChangedEventHandler – This delegate is used by handlers of the CheckStateChangedEvent event.
- ClickEventHandler – Represents the methods that handle the Click event.
- ContactTextBoxSelectionChangedEventHandler – A delegate handler for the ContactTextBoxSelectionChanged event.
- ContactTextBoxTextChangedEventHandler – A delegate handler for the ContactTextBoxTextChanged event.
- ContactTextBoxTextResolvedEventHandler – A delegate handler for the TextResolvedToContact event.
- ContentChangedDelegate – Delegate for the ContentChangedEvent.
- ContextMenuEventHandler - The callback type for handling a ContextMenuEvent.

- DragDeltaEventHandler – This delegate is used by handlers of the DragDeltaEvent event.
- IncludeContactEventHandler – Handler for ContactPickerDialog.IncludeContact event.
- ItemsChangedEventHandler – The delegate to use for handlers that receive ItemsChangedEventArgs.
- OpenedEventHandler – Handler for ContactPickerDialog.Opened event.
- PaginationCompleteDelegate – Delegate for the PaginationCompleteEvent.
- PaginationProgressDelegate – Delegate for the PaginationProgressEvent.
- ScrollChangeEventHandler – This delegate is used by handlers of the ScrollChangeEvent event.
- SelectionChangedEventHandler – The delegate type for handling a selection changed event.
- TextChangedEventHandler – The delegate to use for handlers that receive TextChangedEventArgs.
- ToolTipEventHandler – The callback type for handling a ToolTipEvent.
- ValueChangedEventHandler – This delegate is used by handlers of the ValueChangedEvent event.
- VisibleChangedEventHandler – This delegate is used by handlers of the VisibleChangedEvent event.

Another namespace, System.Windows.Controls.Atoms, is a sub-namespace of the System.Windows.Controls namespace. System.Windows.Controls.Atoms includes associated controls, event arguments and event handlers. The following list contains example classes associated with the System.Windows.Controls.Atoms namespace.

- PageBar – Represents a scrollable pagination control.
- PageElement – Renders a specific page of paginated content. The page to be rendered is specified by the PageSource property.
- PageHoveredEventArgs – PageHoveredEventArgs provides information about where the mouse pointer is hovering.
- PageScrolledEventArgs – The PageScrolledEventArgs contains info pertaining to the PageScrolled Event.
- PageSelectedEventArgs – The PageSelectedEvent is fired when a new row/column range selection is made.
- PageSelector – PageSelector: Allows the user to select a range of rows/columns of pages to be displayed.
- PageSource – Identifies the source of the content to be paginated. It also provides properties and methods for formatting paginated content.

The following list contains example delegates associated with the System.Windows.Controls.Atoms namespace.

- PageHoveredEventHandler – This delegate is used by handlers of the PageHoveredEvent event.
- PageScrolledEventHandler – This delegate is used by handlers of the PageHovered event.
- PageSelectedEventHandler – This delegate is used by handlers of the PageSelectedEvent event.

A System.Windows.Controls.Primitives namespace is another sub-namespace of the System.Windows.Controls namespace. As mentioned above, the Primitives sub-namespace includes controls that are intended to be used as primitives by other more complex controls. The following list contains example classes associated with the System.Windows.Controls.Primitives namespace.

- ButtonBase – When overridden in a derived class, defines the relevant events and properties, and provides handlers for the relevant input events.
- Popup – A control that creates a fly-out window that contains content.
- RangeBase – Represents the base class for elements that have a specific range. Examples of such elements are scroll bars and progress bars. This class defines the relevant events and properties, and provides handlers for the events.

- RepeatButton – RepeatButton control adds repeating semantics of when the Click event occurs.
- ScrollArea – ScrollArea is the effective element for scrolling. It contains content that it clips and provides properties to expose the content's offset and extent. It also provides default input handling such that scrolling can be driven programatically or via keyboard or mouse wheel.
- ScrollBar – The ScrollBar class.
- Selector – The base class for controls that select items from among their children.
- Slider – The Slider class.
- Thumb – The thumb control enables basic drag-movement functionality for scrollbars and window resizing widgets.

"IEnsureVisible" is an example interface associated with the System.Windows.Controls.Primitives namespace. IEnsureVisible is implemented on a visual to scroll/move a child visual into view.

The following list contains example enumerations associated with the System.Windows.Controls.Primitives namespace.

- ArrowButtonStates –
- CloseModeType – Describes how a popup should behave to various mouse events.

- Part – The Part enumeration is used to indicate the semantic use of the controls that make up the scroll bar.
- PartStates – ScrollBar Part States.
- PlacementType – Describes where a popup should be placed on screen.
- SizeBoxStates –

A documents namespace 312 is a collection of semantic and formatting elements that are used to create richly formatted and semantically rich documents. In one embodiment, an "element" is a class that is primarily used in conjunction with a hierarchy of elements (referred to as a "tree"). These elements can be interactive (e.g., receiving user input via keyboard, mouse or other input device), can render images or objects, and can assist with the arrangement of other elements. Example elements include a "Block" element that implements a generic block, a "Body" element that represents content that includes the body of a table, a "Cell" element that contains tabular data within a table, a "Header" element that represents the content included in the header of a table, and a "PageBreak" element that is used to break content across multiple pages.

The following list contains example classes exposed by the System.Windows.Documents namespace.

- AdaptiveMetricsContext - AdaptiveMetricsContext provides the root element for adaptive-flow-format documents. Once a child panel is encapsulated in an AdaptiveMetricsContext element, the content of the panel is processed by the Reading Metrics Engine (RME). The size of the

child panel is used to calculate the number and size of any columns as well as optimum font sizes and line heights.

- Block - Implements a generic block element that does not induce default rendering behavior.
- BlockElement - Implements a base class for all Block elements.
- Body - Represents the content that comprises the body of a Table element.
- Bold - Implements a Bold element derived from Inline.
- BreakRecord - Stores information necessary to continue formatting paginated content across page breaks. Inherit from this class to provide pagination support. This is an abstract class.
- Cell – Cells contain tabular data within a Table. Cell elements are contained within a Row.
- CellCollection - Ordered collection of table cells.
- Column – The Column element is used to apportion the contents of a GridPanel or Table.
- ColumnCollection - A ColumnCollection is an ordered collection of Columns.
- ColumnResult - Represents a column's view-related information.

- ContainerParagraphResult – Provides access to calculated layout parameters for a Paragraph object which contains only other Paragraph objects.

- ContentPosition - Represents the position of content within a paragraph. Inherit from this class to describe the position of associated content. This is an abstract class.

- Document - The purpose of the Document class is to decouple the content of a document from the UI "chrome" that surrounds it. "Decoupling" means that you can author a document without thinking about (and without committing to) its UI. The Document class holds document content. typically a TextPanel or a FixedPanel and its children. A visual tree (by default, a PageViewer) is associated with this element through the WPP control styling mechanism.

- DocumentPage - Represents layout information for a control associated with a page of a document subject to pagination. Inherit from this class to implement to describe the layout information for these controls. This is an abstract class.

- DocumentPageParagraphResult - Provides access to calculated layout parameters for objects affected by pagination.

- FindEngine – Base class for find algorithms.

- FindEngineFactory – Find algorithms factory.

- FixedPage - Provides access to a single page of content within a fixed-format layout document.
- Footer – Represents the content that comprises the footer of a Table element.
- Header - Represents the content that comprises the header of a Table element.
- Heading – Implements a block-level element that renders text as a heading.
- HyphenationDictionary - HyphenationDictionary represents a dictionary for the purpose of providing hyphenation support within applications. It can contain both an inline dictionary and a reference to an external dictionary. The inline dictionary has higher priority and will be applied before entries in the external dictionary.
- Hyphenator – The Hyphenator object maintains reference to hyphenation data within a HyphenationDictionary and also performs hyphenation.
- Inline - Implements a generic Inline element that does not induce any default rendering behavior.
- InlineElement - Implements a generic inline element as base class for all inline elements.
- Italic – Implements an Italic element derived from Inline.
- LineBreak - Represents a markup element that forces a line break.
- LineResult - Provides access to calculated information of a line of text.

- List - Implements a List element. Lists are block-level elements designed to be formatted with markers such as bullets or numbering.
- ListElementItem - Implements a ListElementItem, which supports markers such as bullets or numbering.
- Note - Implements a Note element, which is analagous to the note element in HTML.
- PageBreak - Represents a markup element used to break content across various pages.
- PageDescriptor - Implements PageDescriptor, which stores information necessary to create paginated layout.
- Paragraph - Implements a block-level element used to render text in a paragraph. Rendering behavior is analagous to that of the paragraph element in HTML.
- ParagraphResult - Provides access to calculated layout parameters for a Paragraph object.
- Row – Defines a row within a GridPanel or Table element.
- RowCollection – RowCollection represents an ordered collection of Rows.
- RowGroup – Specifies property defaults for a group of rows in a Table or GridPanel.
- Section - Implements a generic container element. Rendering behavior is analagous to the div element in HTML.

- SmallCaps - Implements an inline SmallCaps element. SmallCaps are typographic forms that render as small capital versions of letters for emphasis, as in a title.

- Subscript - Represents an inline Subscript element. Subscript characters are written immediately below, below and to the left, or below and to the right of other characters.

- Superscript - Represents an inline Superscript element. Superscript characters are typically letters or numbers and render immediately above, above and to the left, or above and to the right of other characters.

- Table – Table is used to display complex data in tabular form using a markup language (e.g., "XAML").

- TextArray - Base API for text access and manipulation.

- TextChangedEventArgs - The TextChangedEventArgs defines the event arguments sent when a TextArray is changed.

- TextElement - TextElement provides TextRange facilities for the TextTree. It is an immutable, continuous TextRange with fixed endpoints. It provides ContentElement Input, Focus and Eventing support. It also provides DependencyObject property support.

- TextNavigator - This can enumerate text content. Implements a movable TextPosition. It can move by text run or be positioned at a know location in text.

- TextParagraphResult - Provides access to calculated layout parameters for text, including floated objects and figures.

- TextPosition - This is an object representing a certain position in a TextArray. A compact object representing a position in text automatically maintains position when text changes. Comparison operations are only applicable to positions within same TextArray (same Context) TextPosition can be static or movable. IsChangeable property tells the kind of position.

- TextRange - TextRange is an abstract class providing generic association of zero or more subranges with properties. Subrange manipulation is defined on derived classes.

- TextRangeMovable – TextRangeMovable is an abstract class for movable TextRanges. It adds the ability to move the start and end points based on TextUnits.

- TextTreeChangedEventArgs - The TextChangedEventArgs defines the event arguments sent when a TextArray is changed.

- TextTreeDumper - TreeDumper is a tree test class that is public due to packaging issues.

- TextTreeNavigator - This is an object representing a certain moveable position in a TextTree. It is a specific implementation of TextNavigator for use only in the TextTree.

- TextTreePosition - This is an object representing a certain immutable position in a TextTree. It is a specific implementation of TextPosition for use only in the TextTree.

- TextTreeRange - Provides TextRange facilities for the TextTree. It is a mutable, continuous TextRange with movable endpoints.

- TextTreeRangeContentEnumerator - Enumerator on object children directly under a TextTreeRange.

- TextUnit - Extensible unit of text navigation.

- TextUnits - Commonly used text units for TextPosition and TextRange.

- Typography - Provides access to a rich set of OpenType typography properties.

- UIElementParagraphResult - The ParagraphResult for a paragraph which is composed entirely of a UIElement. Used for Floaters, Figures and embedded block level UIElements.

- Underline – Implements an Underline element derived from InlineElement.

The following list contains example interfaces associated with the System.Windows.Documents namespace.

- IDocumentContentHost - Implement this interface on a content host so that children of that host can notify the host when content is changing.

- IDocumentFormatter - Implement this interface on an element to provide support for document features such as pagination.
- ITextDocumentResult - Implement this interface to maintain column information for a document.
- ITextParagraphResult - Implement this interface to provide text and positioning information for text paragraphs.

The following list contains example enumerations associated with the System.Windows.Documents namespace.

- ElementEdge - This identifies the edge of an object where a TextPosition is located.
- FindAdvancedOptions - The advanced search options used by FindAlgorithm (search initialization) and TextRangeMovable/TextSelection (simplified search execution) classes.
- FindOptions - The simplified search options used by TextBox.Find methods.
- LogicalDirection - LogicalDirection defines a logical direction for movement in text. It is also used to determine where a TextPosition will move when content is inserted at the TextPosition.
- TextArrayRunType - This identifies the run where a TextPosition is located, taking LogicalDiretion into account.

- TextChangeOptions - Possible text changes for CanChangeText.
- TextMoveOptions - This controls the movement of TextNavigator by specifying conditions to halt navigation.

The following list contains example delegates associated with the System.Windows.Documents namespace.

- ObjectCloneDelegate - Callback method to provide a clone or copy of a DependencyObject when a portion of a TextArray is being copied or moved.
- TextChangedEventHandler - The TextChangedEventHandler delegate is called with TextChangedEventArgs every time content is added to or removed from the TextTree.

A shapes namespace 314 is a collection of vector graphics elements used to create images and objects. The use of vector graphics elements allows the elements to be easily resized to fit the requirements of a particular interface or display device. The following list contains example classes exposed by the System.Windows.Shapes namespace.

- Ellipse – Draws an ellipse.
- Glyphs - Represents a glyph shape in a markup language such as "XAML". Glyphs are used to represent fonts.
- Line – Draws a straight line between two points.

- Path – Draws a series of connected lines and curves.
- Polygon – Draws a polygon (a connected series of lines that forms a closed shape).
- Polyline – Draws a series of connected straight lines.
- Rectangle – Draws a rectangle.
- Shape – An abstract class that provides base functionality for shape elements, such as ellipse, polygon and rectangle.

The System.Windows.Controls, System.Windows.Documents and System.Windows.Shapes namespaces provide an integrated system for developing applications and related components. This integrated system provides a common programming model for all three namespaces, thereby simplifying development of application programs. This interoperability among all three namespaces allows developers to learn a single programming architecture that is applied to any of the features provided by three namespaces. For example, a common markup language is used across all three namespaces. This common markup language provides for the mapping of classes and properties specified in XML markup to an instantiated tree of objects.

Additionally, a consistent programming model and consistent services are used across the three namespaces. For example, a consistent event system is used to initiate and process various events. A common property system is used to style various properties, bind data to a property, or animate a property, regardless of whether the property is associated with the "Controls", "Documents", or "Shapes"

namespace. Additionally, the same input paradigms and layout handling is common across all three namespaces. For example, various controls from the System.Windows.Controls namespace can be nested in the middle of a document's content defined using the System.Windows.Documents namespace.

Example source files will include a set of windows and panes (also referred to as "pages") that are declaratively defined using "Controls", "Documents" and "Shapes". Interaction logic is also provided for the windows and panes. The interaction logic identifies program code that is executed in response to a particular user action or in response to occurrence of an event or activity. The interaction logic is defined, for example, using a Common Language Runtime (CLR) language. A CLR is a runtime environment that handles execution of program code (e.g., .NET program code) and provides various services such as security-related services and memory-related services. Example CLR languages include C# and visual basic. Source files may also include other stand-alone programming language files, such as C# or visual basic files.

Although this discussion refers to the integration of the "Controls", "Documents" and "Shapes" namespaces, this integration may be applied to any or all of the namespaces and sub-namespaces discussed herein.

A data namespace 316 includes classes and interfaces used to bind properties of elements to data sources, data source classes, and data-specific implementations of data collections and views. These classes and interfaces are also used to handle exceptions in data entry and allow runtime creation of a user interface based on information in various data sources. Data can be displayed in textual form or can be utilized to change the formatting of the display, such as displaying dollar amounts in red if they are negative. Example classes include a

"Bind" class that represents a binding declaration object that manages bindings between a dynamic property user interface and source data, and an "XmlDataSource" class that serves as a data source for data binding to XML content nodes.

A media namespace 318 provides various media classes. Application developers as well as component developers may use these classes to develop various presentation functionality. Example classes in media namespace 318 include an "ImageEffect" class that permits certain imaging effects (e.g.. blur and grayscale), and a "Brush" class that provides a mechanism for filling an area using solid colors, gradients, images, video, and the like.

The media namespace 318 includes a sub-namespace System.Windows.Media.Animation that includes services that allow a developer to animate properties and coordinate a set of animations with a set of timelines. An animation is an object that changes a value over a period of time. Animation effects include moving an object on the display, and changing the size, shape, or color of an object. Multiple animation classes are provided to implement various animation effects. Effects can be achieved by associating an animation with an element's property value. For example, to create a rectangle that fades in and out of view, one or more animations are associated with the opacity property of the rectangle.

The media namespace 318 also includes a sub-namespace System.Windows.Media.TextFormatting that provides various text services. For example, a "TextFormatter" text engine provides services for breaking text lines and formatting text presented on a display. "TextFormatter" is capable of handling

different text character formats and paragraph styles as well as handling international text layout.

A design namespace 320 provides classes that enable the editing of forms and text, formatting data and cross-process data sharing. These classes provide an extensible framework for editing documents, applications, and other content.

An input namespace 322 includes an input manager that coordinates inputs received by the system. The input namespace 322 also includes classes that help manage and provide control for different input devices. such as a keyboard or a mouse.

A navigation namespace 324 provides a set of classes and services that allow the building of applications with navigation paradigms, such as a browser application. These classes and services permit the development of applications with customized navigation experiences. For example, when purchasing a product or service from an online merchant, clicking a "Back" button causes the application to display a different page that asks the user if they want to cancel or change their order. In another example, activating a "Refresh" button causes an application to retrieve new data instead of first reloading the application followed by retrieving the new data. The navigation namespace 324 also includes page functions that provide a mechanism for generating a hierarchy of questions that are presented to a user.

An automation namespace 326 provides a set of classes that support accessibility and user interface automation.

A serialization namespace 328 provides a parser that can load or save a hierarchy of objects (e.g., elements) from or to an XML file or a file with a binary

representation. This process also sets properties associated with the objects and associates event handlers.

An interop namespace 330 provides a set of classes that enable interoperability with other operating systems or computing platforms.

A forms.interop namespace 332 provides an element that allows an application to host a form control operation.

Another namespace, System.IO.CompoundFile (not shown in Fig. 3) provides services to utilize a compound file in which various document distributable files are stored. These services allow for the encryption and compression of content. The services also support the storage of multiple renditions of the same content, such as a re-flowable document and a fixed-format document.

## EXEMPLARY COMPUTING SYSTEM AND ENVIRONMENT

Fig. 4 illustrates an example of a suitable computing environment 400 within which the programming framework 132 may be implemented (either fully or partially). The computing environment 400 may be utilized in the computer and network architectures described herein.

The exemplary computing environment 400 is only one example of a computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the computer and network architectures. Neither should the computing environment 400 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary computing environment 400.

The framework 132 may be implemented with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, multiprocessor systems, microprocessor-based systems, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and so on. Compact or subset versions of the framework may also be implemented in clients of limited resources, such as cellular phones, personal digital assistants, handheld computers, or other communication/computing devices.

The framework 132 may be described in the general context of computer-executable instructions, such as program modules, being executed by one or more computers or other devices. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The framework 132 may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

The computing environment 400 includes a general-purpose computing device in the form of a computer 402. The components of computer 402 can include, by are not limited to, one or more processors or processing units 404, a system memory 406, and a system bus 408 that couples various system components including the processor 404 to the system memory 406.

The system bus 408 represents one or more of several possible types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, such architectures can include an Industry Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA) local bus, and a Peripheral Component Interconnects (PCI) bus also known as a Mezzanine bus.

Computer 402 typically includes a variety of computer readable media. Such media can be any available media that is accessible by computer 402 and includes both volatile and non-volatile media, removable and non-removable media.

The system memory 406 includes computer readable media in the form of volatile memory, such as random access memory (RAM) 410, and/or non-volatile memory, such as read only memory (ROM) 412. A basic input/output system (BIOS) 414, containing the basic routines that help to transfer information between elements within computer 402, such as during start-up, is stored in ROM 412. RAM 410 typically contains data and/or program modules that are immediately accessible to and/or presently operated on by the processing unit 404.

Computer 402 may also include other removable/non-removable, volatile/non-volatile computer storage media. By way of example, Fig. 4 illustrates a hard disk drive 416 for reading from and writing to a non-removable, non-volatile magnetic media (not shown), a magnetic disk drive 418 for reading from and writing to a removable, non-volatile magnetic disk 420 (e.g., a "floppy disk"), and an optical disk drive 422 for reading from and/or writing to a

removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other optical media. The hard disk drive 416, magnetic disk drive 418, and optical disk drive 422 are each connected to the system bus 408 by one or more data media interfaces 426. Alternatively, the hard disk drive 416, magnetic disk drive 418, and optical disk drive 422 can be connected to the system bus 408 by one or more interfaces (not shown).

The disk drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules, and other data for computer 402. Although the example illustrates a hard disk 416, a removable magnetic disk 420, and a removable optical disk 424, it is to be appreciated that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes or other magnetic storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other optical storage, random access memories (RAM), read only memories (ROM), electrically erasable programmable read-only memory (EEPROM), and the like, can also be utilized to implement the exemplary computing system and environment.

Any number of program modules can be stored on the hard disk 416, magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by way of example, an operating system 426, one or more application programs 428, other program modules 430, and program data 432. Each of the operating system 426, one or more application programs 428, other program modules 430, and program data 432 (or some combination thereof) may include elements of the programming framework 132.

A user can enter commands and information into computer 402 via input devices such as a keyboard 434 and a pointing device 436 (e.g., a "mouse"). Other input devices 438 (not shown specifically) may include a microphone, joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and other input devices are connected to the processing unit 404 via input/output interfaces 440 that are coupled to the system bus 408, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB).

A monitor 442 or other type of display device can also be connected to the system bus 408 via an interface, such as a video adapter 444. In addition to the monitor 442, other output peripheral devices can include components such as speakers (not shown) and a printer 446 which can be connected to computer 402 via the input/output interfaces 440.

Computer 402 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computing device 448. By way of example, the remote computing device 448 can be a personal computer, portable computer, a server, a router, a network computer, a peer device or other common network node, and so on. The remote computing device 448 is illustrated as a portable computer that can include many or all of the elements and features described herein relative to computer 402.

Logical connections between computer 402 and the remote computer 448 are depicted as a local area network (LAN) 450 and a general wide area network (WAN) 452. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When implemented in a LAN networking environment, the computer 402 is connected to a local network 450 via a network interface or adapter 454. When implemented in a WAN networking environment, the computer 402 typically includes a modem 456 or other means for establishing communications over the wide network 452. The modem 456, which can be internal or external to computer 402, can be connected to the system bus 408 via the input/output interfaces 440 or other appropriate mechanisms. It is to be appreciated that the illustrated network connections are exemplary and that other means of establishing communication link(s) between the computers 402 and 448 can be employed.

In a networked environment, such as that illustrated with computing environment 400, program modules depicted relative to the computer 402, or portions thereof, may be stored in a remote memory storage device. By way of example, remote application programs 458 reside on a memory device of remote computer 448. For purposes of illustration, application programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computing device 402, and are executed by the data processor(s) of the computer.

An implementation of the framework 132, and particularly, the API 142 or calls made to the API 142, may be stored on or transmitted across some form of computer readable media. Computer readable media can be any available media that can be accessed by a computer. By way of example, and not limitation, computer readable media may comprise "computer storage media" and "communications media." "Computer storage media" include volatile and non-volatile, removable and non-removable media implemented in any method or

technology for storage of information such as computer readable instructions. data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, 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 a computer.

"Communication media" typically embodies computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as carrier wave or other transport mechanism. Communication media also includes any 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. Combinations of any of the above are also included within the scope of computer readable media.

Alternatively, portions of the framework may be implemented in hardware or a combination of hardware, software, and/or firmware. For example, one or more application specific integrated circuits (ASICs) or programmable logic devices (PLDs) could be designed or programmed to implement one or more portions of the framework.

A programming interface (or more simply, interface) may be viewed as any mechanism, process, protocol for enabling one or more segment(s) of code to communicate with or access the functionality provided by one or more other

segment(s) of code. Alternatively, a programming interface may be viewed as one or more mechanism(s), method(s), function call(s), module(s), object(s), etc. of a component of a system capable of communicative coupling to one or more mechanism(s), method(s), function call(s), module(s), etc. of other component(s). The term "segment of code" in the preceding sentence is intended to include one or more instructions or lines of code, and includes, e.g., code modules, objects, subroutines, functions, and so on, regardless of the terminology applied or whether the code segments are separately compiled, or whether the code segments are provided as source, intermediate, or object code, whether the code segments are utilized in a runtime system or process, or whether they are located on the same or different machines or distributed across multiple machines, or whether the functionality represented by the segments of code are implemented wholly in software, wholly in hardware, or a combination of hardware and software.

Notionally, a programming interface may be viewed generically, as shown in Fig. 5 or Fig. 6. Fig. 5 illustrates an interface Interface1 as a conduit through which first and second code segments communicate. Fig. 6 illustrates an interface as comprising interface objects I1 and I2 (which may or may not be part of the first and second code segments), which enable first and second code segments of a system to communicate via medium M. In the view of Fig. 6, one may consider interface objects I1 and I2 as separate interfaces of the same system and one may also consider that objects I1 and I2 plus medium M comprise the interface. Although Figs. 5 and 6 show bi-directional flow and interfaces on each side of the flow, certain implementations may only have information flow in one direction (or no information flow as described below) or may only have an interface object on one side. By way of example, and not limitation, terms such as application

programming or program interface (API), entry point, method, function, subroutine, remote procedure call, and component object model (COM) interface, are encompassed within the definition of programming interface.

Aspects of such a programming interface may include the method whereby the first code segment transmits information (where "information" is used in its broadest sense and includes data, commands, requests, etc.) to the second code segment; the method whereby the second code segment receives the information; and the structure, sequence, syntax, organization, schema, timing and content of the information. In this regard, the underlying transport medium itself may be unimportant to the operation of the interface, whether the medium be wired or wireless, or a combination of both, as long as the information is transported in the manner defined by the interface. In certain situations, information may not be passed in one or both directions in the conventional sense, as the information transfer may be either via another mechanism (e.g. information placed in a buffer, file, etc. separate from information flow between the code segments) or non-existent, as when one code segment simply accesses functionality performed by a second code segment. Any or all of these aspects may be important in a given situation, e.g., depending on whether the code segments are part of a system in a loosely coupled or tightly coupled configuration, and so this list should be considered illustrative and non-limiting.

This notion of a programming interface is known to those skilled in the art and is clear from the foregoing detailed description of the invention. There are, however, other ways to implement a programming interface, and, unless expressly excluded, these too are intended to be encompassed by the claims set forth at the end of this specification. Such other ways may appear to be more sophisticated or



42

complex than the simplistic view of Figs. 5 and 6, but they nonetheless perform a similar function to accomplish the same overall result. We will now briefly describe some illustrative alternative implementations of a programming interface.

A. FACTORING

A communication from one code segment to another may be accomplished indirectly by breaking the communication into multiple discrete communications. This is depicted schematically in Figs. 7 and 8. As shown, some interfaces can be described in terms of divisible sets of functionality. Thus, the interface functionality of Figs. 5 and 6 may be factored to achieve the same result, just as one may mathematically provide 24, or 2 times 2 times 3 times 2. Accordingly, as illustrated in Fig. 7, the function provided by interface Interface1 may be subdivided to convert the communications of the interface into multiple interfaces Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As illustrated in Fig. 8, the function provided by interface I1 may be subdivided into multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly, interface I2 of the second code segment which receives information from the first code segment may be factored into multiple interfaces I2a, I2b, I2c, etc. When factoring, the number of interfaces included with the 1st code segment need not match the number of interfaces included with the 2nd code segment. In either of the cases of Figs. 7 and 8, the functional spirit of interfaces Interface1 and I1 remain the same as with Figs. 5 and 6, respectively. The factoring of interfaces may also follow associative, commutative, and other mathematical properties such that the factoring may be difficult to recognize. For instance, ordering of operations may be unimportant, and consequently, a function carried out by an interface may be

carried out well in advance of reaching the interface, by another piece of code or interface, or performed by a separate component of the system. Moreover, one of ordinary skill in the programming arts can appreciate that there are a variety of ways of making different function calls that achieve the same result.

B. REDEFINITION

In some cases, it may be possible to ignore, add or redefine certain aspects (e.g., parameters) of a programming interface while still accomplishing the intended result. This is illustrated in Figs. 9 and 10. For example, assume interface Interface1 of Fig. 5 includes a function call *Square(input, precision, output)*, a call that includes three parameters, *input, precision and output*, and which is issued from the 1st Code Segment to the 2nd Code Segment., If the middle parameter *precision* is of no concern in a given scenario, as shown in Fig. 9, it could just as well be ignored or even replaced with a *meaningless* (in this situation) parameter. One may also add an *additional* parameter of no concern. In either event, the functionality of square can be achieved, so long as output is returned after input is squared by the second code segment. *Precision* may very well be a meaningful parameter to some downstream or other portion of the computing system; however, once it is recognized that *precision* is not necessary for the narrow purpose of calculating the square, it may be replaced or ignored. For example, instead of passing a valid *precision* value, a meaningless value such as a birth date could be passed without adversely affecting the result. Similarly, as shown in Fig. 10, interface I1 is replaced by interface I1', redefined to ignore or add parameters to the interface. Interface I2 may similarly be redefined as interface I2', redefined to ignore unnecessary parameters, or parameters that may be processed elsewhere.



64

The point here is that in some cases a programming interface may include aspects, such as parameters, that are not needed for some purpose, and so they may be ignored or redefined, or processed elsewhere for other purposes.

C. INLINE CODING

It may also be feasible to merge some or all of the functionality of two separate code modules such that the "interface" between them changes form. For example, the functionality of Figs. 5 and 6 may be converted to the functionality of Figs. 11 and 12, respectively. In Fig. 11, the previous 1st and 2nd Code Segments of Fig. 5 are merged into a module containing both of them. In this case, the code segments may still be communicating with each other but the interface may be adapted to a form which is more suitable to the single module. Thus, for example, formal Call and Return statements may no longer be necessary, but similar processing or response(s) pursuant to interface Interface1 may still be in effect. Similarly, shown in Fig. 12, part (or all) of interface I2 from Fig. 6 may be written inline into interface I1 to form interface I1". As illustrated, interface I2 is divided into I2a and I2b, and interface portion I2a has been coded in-line with interface I1 to form interface I1". For a concrete example, consider that the interface I1 from Fig. 6 performs a function call square (*input*, *output*), which is received by interface I2, which after processing the value passed with *input* (to square it) by the second code segment, passes back the squared result with *output*. In such a case, the processing performed by the second code segment (squaring *input*) can be performed by the first code segment without a call to the interface.

D. DIVORCE

A communication from one code segment to another may be accomplished indirectly by breaking the communication into multiple discrete communications. This is depicted schematically in Figs. 13 and 14. As shown in Fig. 13, one or more piece(s) of middleware (Divorce Interface(s), since they divorce functionality and / or interface functions from the original interface) are provided to convert the communications on the first interface, Interface1, to conform them to a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. This might be done, e.g., where there is an installed base of applications designed to communicate with, say, an operating system in accordance with an Interface1 protocol, but then the operating system is changed to use a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. The point is that the original interface used by the 2nd Code Segment is changed such that it is no longer compatible with the interface used by the 1st Code Segment, and so an intermediary is used to make the old and new interfaces compatible. Similarly, as shown in Fig. 14, a third code segment can be introduced with divorce interface DI1 to receive the communications from interface I1 and with divorce interface DI2 to transmit the interface functionality to, for example, interfaces I2a and I2b, redesigned to work with DI2, but to provide the same functional result. Similarly, DI1 and DI2 may work together to translate the functionality of interfaces I1 and I2 of Fig. 6 to a new operating system, while providing the same or similar functional result.

E. REWRITING

Yet another possible variant is to dynamically rewrite the code to replace the interface functionality with something else but which achieves the same

overall result. For example, there may be a system in which a code segment presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is provided to a Just-in-Time (JIT) compiler or interpreter in an execution environment (such as that provided by the .Net framework, the Java runtime environment, or other similar runtime type environments). The JIT compiler may be written so as to dynamically convert the communications from the 1st Code Segment to the 2nd Code Segment, i.e., to conform them to a different interface as may be required by the 2nd Code Segment (either the original or a different 2nd Code Segment). This is depicted in Figs. 15 and 16. As can be seen in Fig. 15, this approach is similar to the Divorce scenario described above. It might be done, e.g., where an installed base of applications are designed to communicate with an operating system in accordance with an Interface 1 protocol, but then the operating system is changed to use a different interface. The JIT Compiler could be used to conform the communications on the fly from the installed-base applications to the new interface of the operating system. As depicted in Fig. 16, this approach of dynamically rewriting the interface(s) may be applied to dynamically factor, or otherwise alter the interface(s) as well.

It is also noted that the above-described scenarios for achieving the same or similar result as an interface via alternative embodiments may also be combined in various ways, serially and/or in parallel, or with other intervening code. Thus, the alternative embodiments presented above are not mutually exclusive and may be mixed, matched and combined to produce the same or equivalent scenarios to the generic scenarios presented in Figs. 5 and 6. It is also noted that, as with most programming constructs, there are other similar ways of achieving the same or similar functionality of an interface which may not be described herein, but

nonetheless are represented by the spirit and scope of the invention, i.e., it is noted that it is at least partly the functionality represented by, and the advantageous results enabled by, an interface that underlie the value of an interface.

## Conclusion

Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.

## CLAIMS

1. A programming interface embodied on one or more computer readable media, comprising:

a first group of services related to generating graphical objects;

a second group of services related to formatting content; and

a third group of services related to creating components of the graphical objects.

2. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common programming model.

3. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services utilize a common markup language.

4. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common event system.

5. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common property definition system.

6. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common input paradigm.

7. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common system for nesting elements associated with a particular group of services within elements associated with another group of services.

8. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines an appearance of the graphical objects.

9. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines a behavior of the graphical objects.

10. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines an arrangement of the graphical objects.

11. A programming interface as recited in claim 1, wherein the first group of services includes a plurality of nested elements that define the graphical objects.

12. A programming interface as recited in claim 1, wherein the graphical objects are comprised of one or more elements defined by vector graphics.

13. A programming interface as recited in claim 1, wherein the first group of services can define window properties in a markup language without launching a new window.

14. A programming interface as recited in claim 1, wherein the first group of services generate a user interface containing a plurality of graphical objects.

15. A programming interface as recited in claim 1, wherein the second group of services arrange the graphical objects.

16. A software architecture comprising the programming interface as recited in claim 1.

**17.** An application program interface embodied on one or more computer readable media, comprising:

a first group of services related to generating graphical objects;

a second group of services related to formatting content; and

a third group of services related to creating components of the graphical objects, wherein the first group of services, the second group of services and the third group of services share a common programming model.

**18.** An application program interface as recited in claim 17, wherein the first group of services, the second group of services and the third group of services utilize a common markup language.

**19.** An application program interface as recited in claim 17, wherein the third group of services includes services to generate geometric shapes.

**20.** An application program interface as recited in claim 17, wherein the second group of services includes arranging a plurality of data elements.

**21.** An application program interface as recited in claim 17, wherein the first group of services includes:

a service that determines an appearance of a graphical object; and

a service that determines a behavior of the graphical object.

**22.** An application program interface as recited in claim 17, wherein the first group of services includes a service that defines window properties in a markup language without launching a new window.

**23.** A computer system including one or more microprocessors and one or more software programs, the one or more software programs utilizing a programming interface to request services from an operating system, the programming interface including separate commands to request services consisting of the following groups of services:

a first group of services for generating graphical objects; and

a second group of services for creating components of the graphical objects, wherein the first group of services and the second group of services share a common programming model.

**24.** A computer system as recited in claim 23, wherein the first group of services includes:

a service for defining an appearance of the graphical objects; and

a service for defining an arrangement of the graphical objects.

**25.** A computer system as recited in claim 23, wherein the second group of services includes services to generate a plurality of geometric shapes.

**26.** A method comprising:

calling one or more first functions to facilitate generating graphical objects; and

calling one or more second functions to facilitate formatting content, wherein the first functions and the second functions share a common programming model.

**27.** A method as recited in claim 26, further including calling one or more third functions to facilitate creating components of the graphical objects.

**28.** A method as recited in claim 26, further including calling one or more third functions to facilitate generating geometric shapes contained in the graphical objects.

**29.** A method as recited in claim 26, wherein the first functions facilitate:

defining window properties in a markup language without launching a new window; and

generating a user interface containing a plurality of graphical objects.

**30.** A system comprising:

means for exposing a first set of functions that enable generating graphical objects; and

means for exposing a second set of functions that enable creating components of the graphical objects, wherein the components of the graphical objects include a plurality of geometric shapes, and wherein the first set of functions and the second set of functions share a common programming model.

**31.** A system as recited in claim 30, wherein the second set of functions further enable arrangement of the geometric shapes on a page to be rendered.

**32.** A system as recited in claim 30, further comprising means for exposing a third set of functions that enable formatting content for display.

**33.** A system as recited in claim 30, wherein the first set of functions and the second set of functions utilize a common markup language.

**34.** A system as recited in claim 30, wherein the first set of functions and the second set of functions share a common event system and a common property definition system.

**ABSTRACT**

A programming interface provides functions for generating applications, documents, media presentations and other content. These functions allow developers to obtain services from an operating system, object model service, or other system or service.

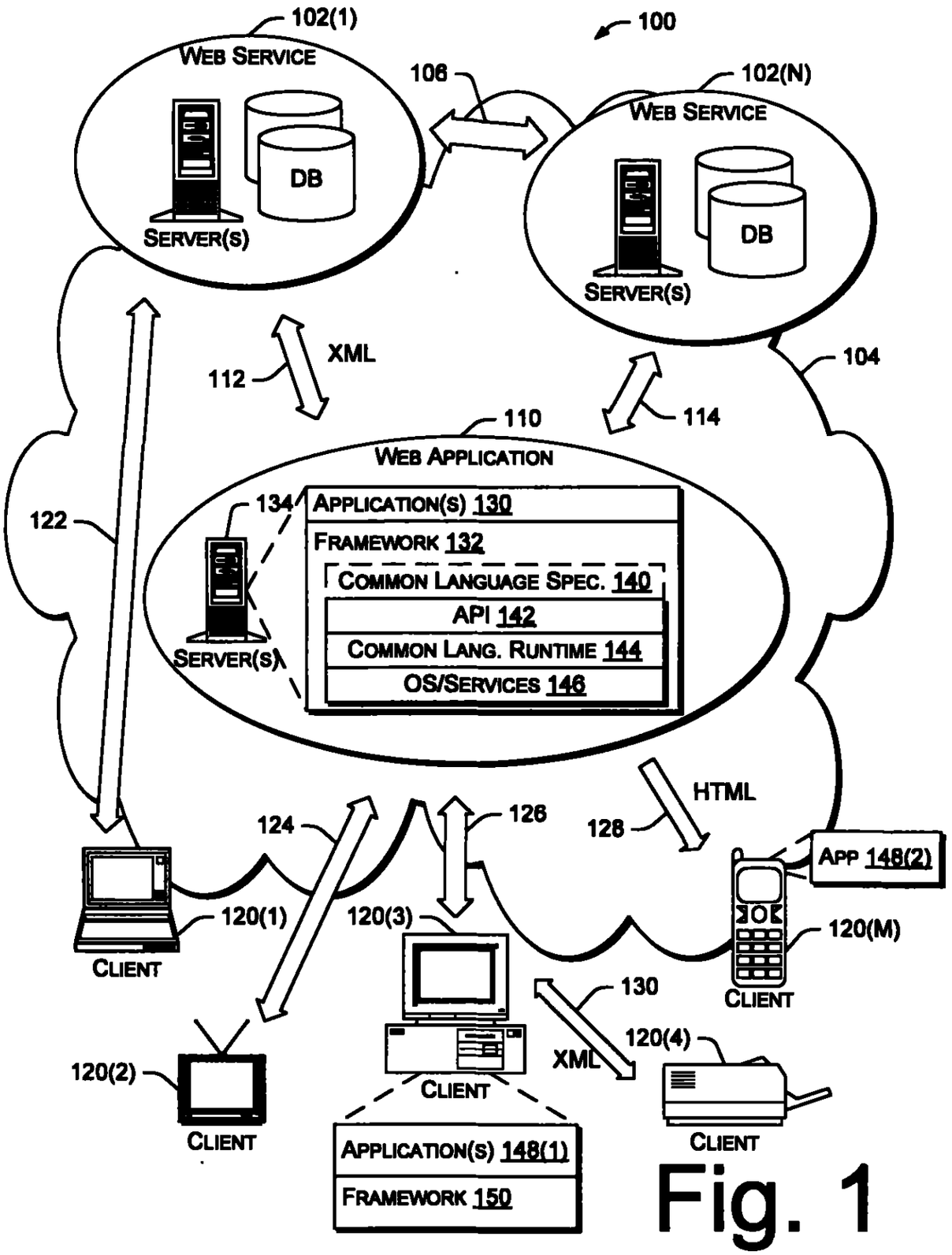

102(1)
WEB SERVICE
DB
SERVER(S)
100
106
102(N)
WEB SERVICE
DB
SERVER(S)
XML
112
104
114
110
WEB APPLICATION
122
134
APPLICATION(S) 130
FRAMEWORK 132
COMMON LANGUAGE SPEC. 140
API 142
COMMON LANG. RUNTIME 144
OS/SERVICES 146
SERVER(S)
HTML
128
124
126
APP 148(2)
120(1)
120(3)
120(M)
CLIENT
CLIENT
CLIENT
130
XML
120(4)
120(2)
CLIENT
CLIENT
APPLICATION(S) 148(1)
FRAMEWORK 150

Fig. 1

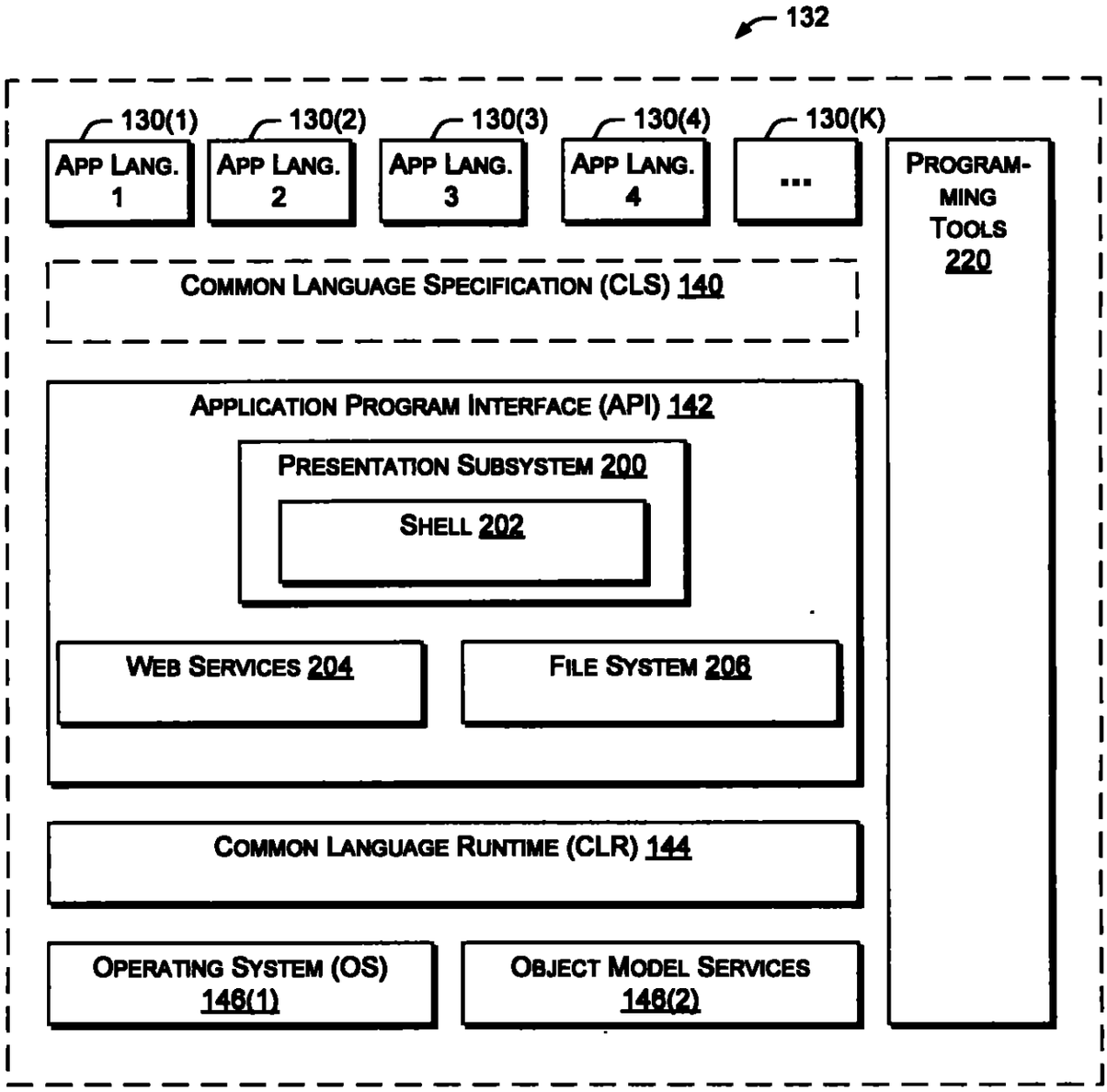

132
130(1)
APP LANG. 1
130(2)
APP LANG. 2
130(3)
APP LANG. 3
130(4)
APP LANG. 4
130(K)
...
PROGRAM-MING TOOLS 220
COMMON LANGUAGE SPECIFICATION (CLS) 140
APPLICATION PROGRAM INTERFACE (API) 142
PRESENTATION SUBSYSTEM 200
SHELL 202
WEB SERVICES 204
FILE SYSTEM 206
COMMON LANGUAGE RUNTIME (CLR) 144
OPERATING SYSTEM (OS) 146(1)
OBJECT MODEL SERVICES 146(2)

Fig. 2

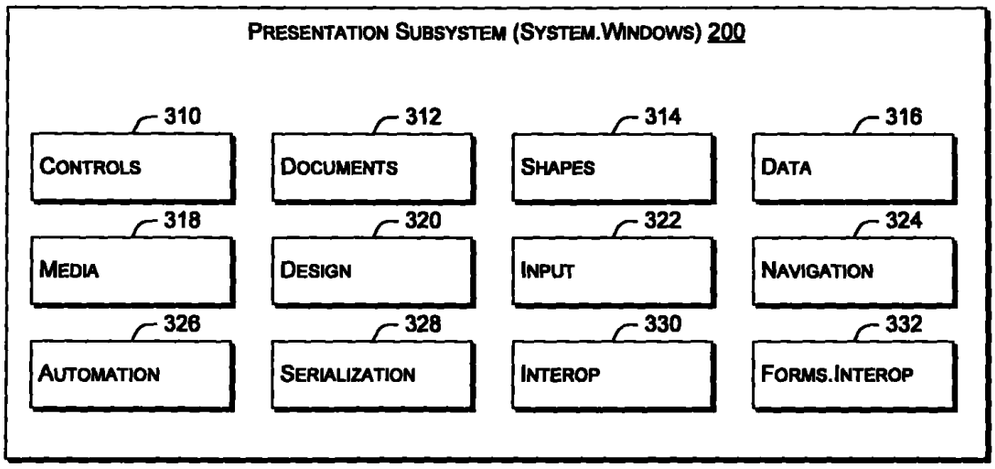

PRESENTATION SUBSYSTEM (SYSTEM.WINDOWS) 200
310
CONTROLS
312
DOCUMENTS
314
SHAPES
316
DATA
318
MEDIA
320
DESIGN
322
INPUT
324
NAVIGATION
326
AUTOMATION
328
SERIALIZATION
330
INTEROP
332
FORMS.INTEROP

Fig. 3

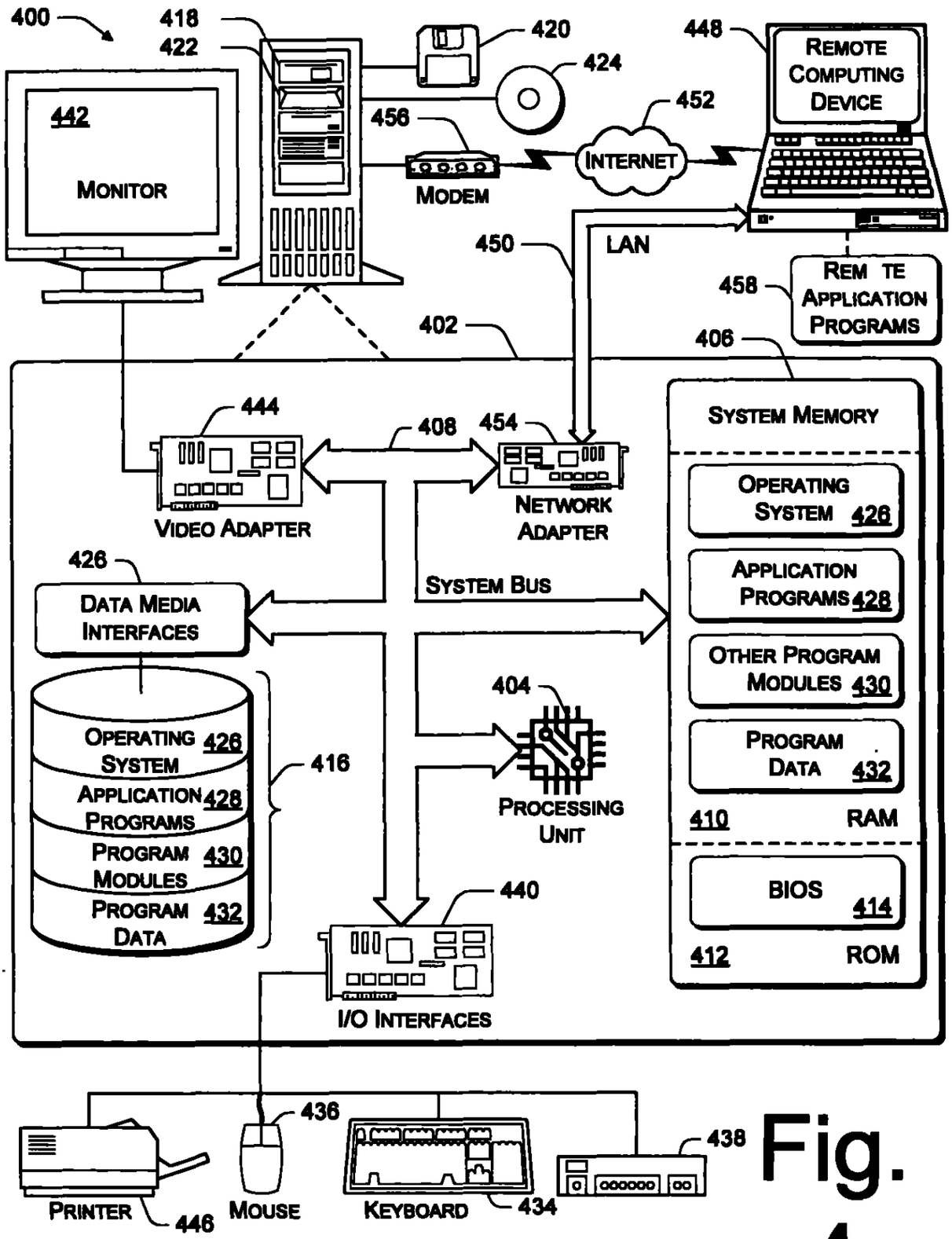

400
418
422
420
424
448
REMOTE COMPUTING DEVICE
442
MONITOR
456
MODEM
452
INTERNET
450
LAN
458
REMOTE APPLICATION PROGRAMS
402
406
SYSTEM MEMORY
444
VIDEO ADAPTER
408
454
NETWORK ADAPTER
OPERATING SYSTEM 426
APPLICATION PROGRAMS 428
OTHER PROGRAM MODULES 430
PROGRAM DATA 432
410 RAM
BIOS 414
412 ROM
426
DATA MEDIA INTERFACES
SYSTEM BUS
404
PROCESSING UNIT
OPERATING SYSTEM 426
APPLICATION PROGRAMS 428
PROGRAM MODULES 430
PROGRAM DATA 432
416
440
I/O INTERFACES
436
438
PRINTER
446
MOUSE
KEYBOARD
434

Fig. 4

80

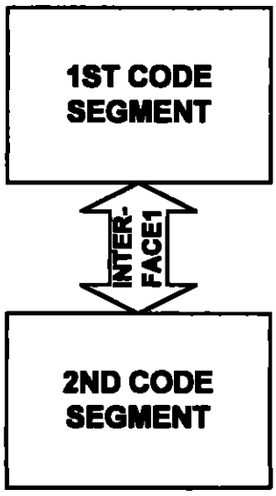



FIGURE 5

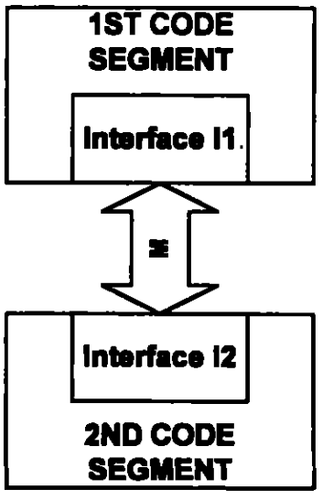



FIGURE 6

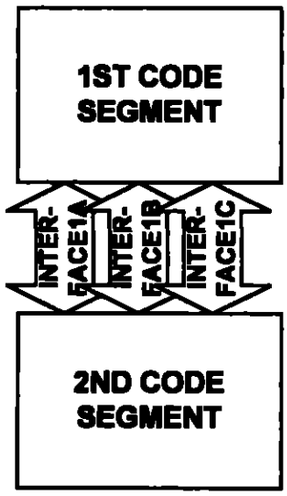



FIGURE 7

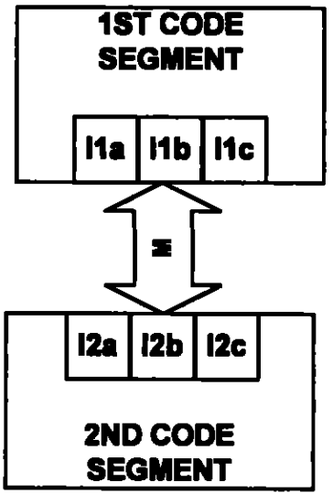



FIGURE 8

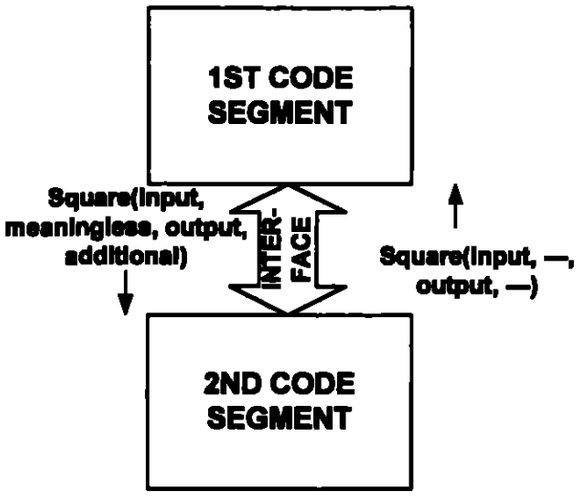



FIGURE 9

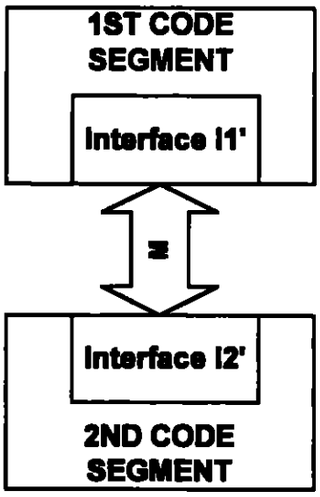



FIGURE 10

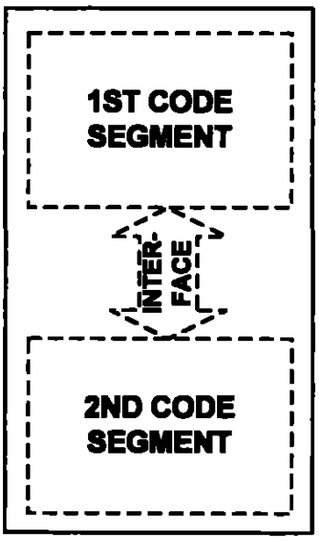



FIGURE 11

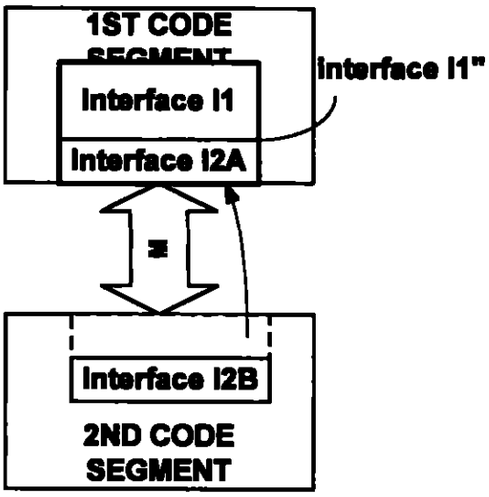



FIGURE 12

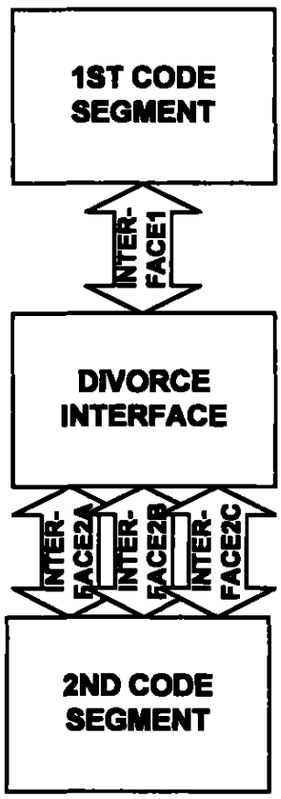



FIGURE 13

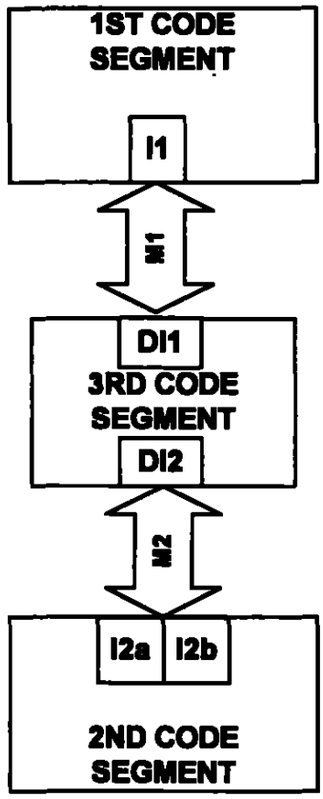



FIGURE 14

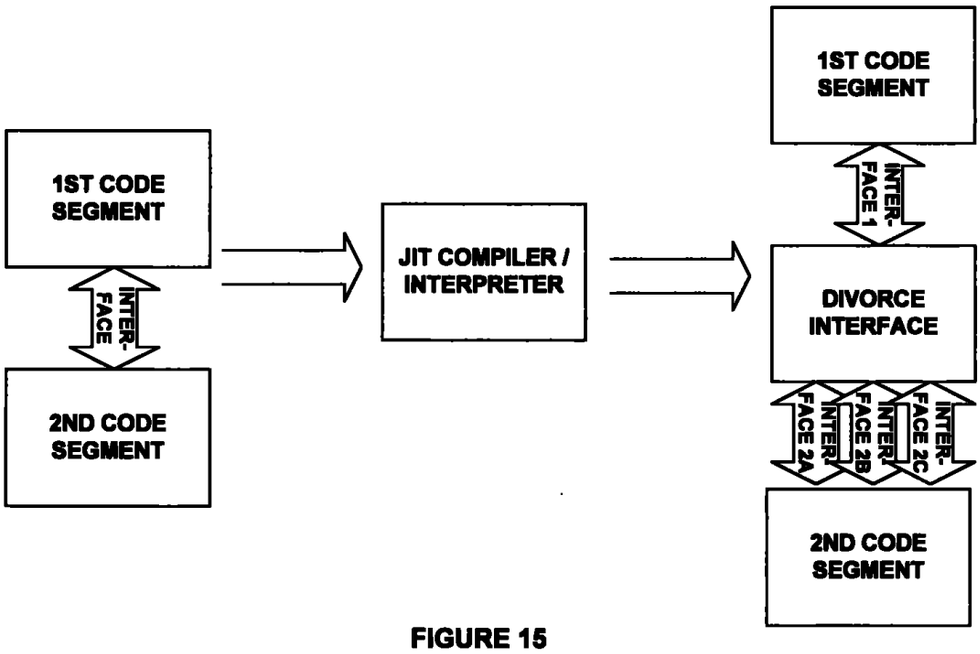

1ST CODE SEGMENT
INTER-FACE
2ND CODE SEGMENT
JIT COMPILER / INTERPRETER
1ST CODE SEGMENT
INTER-FACE 1
DIVORCE INTERFACE
INTER-FACE 2A
INTER-FACE 2B
INTER-FACE 2C
2ND CODE SEGMENT

FIGURE 15

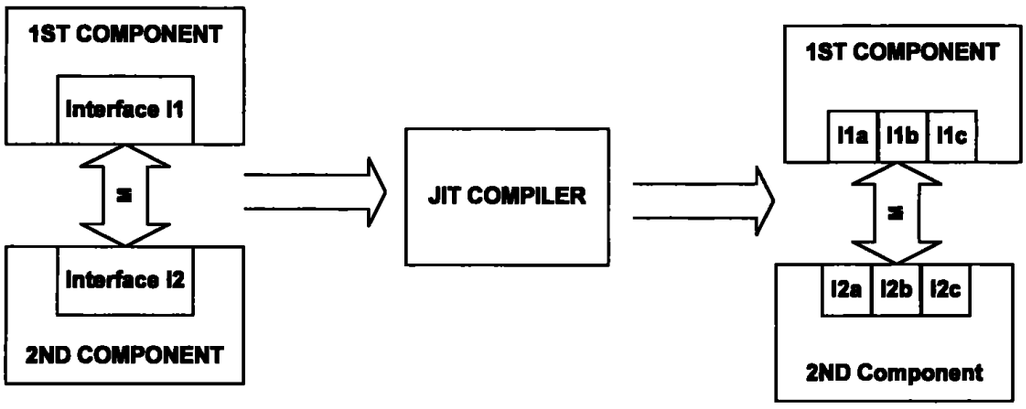

1ST COMPONENT
Interface I1
Interface I2
2ND COMPONENT
JIT COMPILER
1ST COMPONENT
I1a
I1b
I1c
I2a
I2b
I2c
2ND Component

FIGURE 16

## CAMPO TÉCNICO

Esta invención se relaciona con el software y con el desarrollo del mismo. Más particularmente, esta invención se relaciona con una interfaz de programación que facilita el uso de una plataforma de software por programas de aplicación y hardware de computadora.

## BREVE DESCRIPCIÓN QUE ACOMPAÑA LOS DISCOS COMPACTOS

Estas especificaciones están acompañadas por un grupo de tres discos compactos que almacenan el kit de desarrollo de software (SDK- Software Development Kit) para el Codigo-Nombrado "Longhorn" del sistema operativo de Microsoft Windows®. El SDK contiene la documentación para el Codigo-Nombrado "Longhorn" del sistema operativo de Microsoft Windows®. Las copias duplicadas de cada uno de estos tres discos compactos también acompañan esta especificación. El primer disco compacto de los tres (el CD 1 de 3) incluye una carpeta o directorio del archivo nombrada el "lhsdk" que fue creado de octubre el 22 de 2003; tiene 586 Mb de tamaño, contiene 9.692 subdirectorios o subcarpetas, y contiene 44.292 subarchivos. El segundo disco compacto de los tres discos (el CD 2 de 3) incluye una carpeta nombrada "ns" que fue creado en octubre 22 de 2003; tiene 605 Mb de tamaño, contiene 12.628 subdirectorios o subcarpetas, y contiene 44.934 subarchivos. El tercer disco compacto de los tres (el CD 3 de 3) incluye una carpeta nombrada "ns" que fue creado en octubre 22 de 2003; tiene 575 Mb de tamaño, contiene 9.881 subdirectorios o subcarpetas, y contiene 43.630 subarchivos. Los archivos en cada uno de estos tres discos compactos se pueden ejecutar en un dispositivo basado en Windows® (e.g., IBM-PC, o equivalente) que ejecute un sistema operativo de la marca Windows® (e.g., Windows® NT,

Windows® 98, Windows® 2000, Windows® XP, etc.). Los archivos en cada uno de los tres discos compactos estan por este medio incorporados por referencia.

Cada disco compacto del grupo de los tres en sí mismo es un CD-R, y se conforma con el estándar de la ISO 9660. El contenido de cada disco compacto en el grupo de los tres discos compactos está en conformidad con el código ASCII (ASCII).

**El FONDO**

Hace tiempo, el software de computadora vino a ser categorizado como software de "sistema operatvivo" o software de "aplicación". Ampliamente hablando, un software de aplicación es software para realizar una tarea específica para el usuario de la computadora tal como solucionar una ecuación matemática o soporte del procesamiento de textos. El sistema operativo es el software que maneja y controla el hardware. La meta del sistema operativo es poner los recursos de la computadora a disposición de la apliación del programador mientras que en el mismo tiempo, oculta la complejidad necesaria para controlar realmente el hardware.

El sistema operativo hace los recursos disponibles vía funciones que se conocen colectivamente como Iterfase del Programa de Aplicación o el API. El término API también se utiliza en referencia a una sola de estas funciones. Las funciones se agrupan a menudo en términos de qué recurso o servicio proporcionan al programa de aplicación. El software de aplicación solicita recursos llamando funciones individuales API. Las funciones API también sirven como medios por los cuales los mensajes y la información proporcionados por el sistema operativo son retransmitidos de nuevo al software de aplicación.

Adicionalmente a los cambios de hardware, otro factor que conduce a la evolución del software de sistema operativo ha sido el deseo de simplificar y de apresurar el desarrollo del software de aplicación. El desarrollo del software de aplicación puede ser una tarea desalentadora, a veces requiriendo años del tiempo del desarrollador para crear un programa sofisticado con millones de líneas de código. Para un sistema operativo popular tal como varias versiones del sistema operativo de Microsoft Windows, los desarrolladores de software de aplicación escriben miles de diversos aplicaciones cada año que utilizan el sistema operativo. Una base coherente y util del sistema operativo se requiere para apoyar tan diversos desarrolladores de aplicaciones.

A menudo, el desarrollo del software de aplicación puede ser hecho más simple haciendo el sistema operativo más complejo. Es decir, si una función puede ser útil a varios diversos programas de aplicación, puede ser mejor escribirla una vez para incluirla en el sistema operativo, que requerir a docenas de desarrolladores de software que escriban docenas de veces para incluirlo en docenas de aplicaciones. De este modo, si el sistema operativo apoya una amplia gama de la funcionalidad común requerida por un número de aplicaciones, los ahorros significativos en los costos y el tiempo del desarrollo de software de aplicación pueden ser alcanzados. Sin importar donde la línea entre el sistema operativo y el software de aplicaión es dibujada, está claro que para un sistema operativo útil, el API entre el sistema operativo y el hardware y el software de aplicación es tan importante como la operación interna eficiente del sistema operativo en sí mismo. Al desarrollar aplicaciones, los desarrolladores utilizan una variedad de herramientas para generar los artículos gráficos y otro contenido. Las herramientas adicionales están

disponibles para arreglar artículos gráficos y otros datos que van a ser mostrados o entregados. Estas herramientas son creadas típicamente por diversas entidades o diferentes desarrolladores de herramientas. Consecuentemente, las herramientas no proporcionan un ambiente de programación constante. Así, un desarrollador usando estas diversas herramientas necesita aprender cómo utilizar cada una de las herramientas y procurar hacer que se comuniquen la una con la otra. Estas actividades pueden ser tediosas y consumidoras de tiempo, desperdiciando tiempo lejos de la tarea real del desarrollo actual.

Durante los últimos años, la adopción universal del Internet, y la tecnología de una red en general, han cambiado el panorama para los desarrolladores de software. Tradicionalmente, los desarrolladores de software centrados en los usos del software para las computadoras de escritorio independientes (standalone), o computadoras basadas en red de area local (LAN) que fueron conectadas con un número limitado de otras computadoras vía una red de área local (LAN). Tales aplicaciones de software fueron referidos típicamente como productos "paquetes" porque el software fue puesto y vendido en un paquete (shrink-wrapped). Las aplicaiones utilizaron las bien definidsa APIs para tener acceso al sistema operativo subyacente de la computadora.

Como el Internet evolucionó y ganó una amplia aceptación, la industria comenzó a reconocer el poder de las aplicaciones hospedadas (de Hosting) en varios sitios de la WEB mundial (o simplemente el "Web"). En el mundo interconectado, los clientes desde cualquierparte podrían someter peticiones a las aplicaiones basadas en servidores y localizaciones en diversos sitios y recibir respuestas en fracciones de segundo. Estos usos del Web, sin embargo, fueron desarrollados típicamente usando la misma plataforma del sistema operativo que fue desarrollado

originalmente para las máquinas que computaban independientes o de área local. Desafortunadamente, en algunos casos, estas aplicacions no transfieren adecuadamente al ambiente de cómputo distribuido. La plataforma subyacente no fue construida simplemente con la idea de apoyar números ilimitados de computadoras interconectadas.

Para acomodarse al cambio del ambiente de cómputo distribuido (Distributed Computing Environment) que estaba incluido por el Internet, Microsoft Corporation desarrolló una plataforma de software de red conocida como ".NET" (leído como "Punto NET ó Punto RED"). El .NET de Microsoft es software para conectar a la gente, la información, los sistemas, y los dispositivos. La plataforma permite que los desarrolladores creen los servicios Web que se ejecutarán sobre Internet. Este cambio dinámico fue acompañado por un conjunto de funciones API dentro del sistema .NET™ de Microsoft. Mientras que el uso de la marca .NET™ ha llegado a ser cada vez más común, se han identificado las maneras de aumentar la eficacia y/o el funcionamiento de la plataforma. Los inventores han desarrollado un sistema único de funciones API que tienen en cuenta para cada incrementar tal eficacia y/o funcionamiento crecientes.

## RESUMEN

Un programa de interfase, tal como un API, proporciona las funciones para generar las aplicaciones, los documentos, las presentaciones y otros contenidos. Estas funciones permiten que los desarrolladores obtengan servicios del sistema operativo, del servicio modelo del objeto, o de otro sistema o servicio. En una personalización, las funciones permiten que un desarrollador genere un interfaz gráfica de usuario.

## BREVE DESCRIPCIÓN DE LOS DIBUJOS

Los mismos números que se utilizan a través de los dibujos son para referir como características.

Fig. 1 ilustra una arquitectura de red en la cual los clientes tienen acceso a servicios Web sobre Internet usando protocolos convencionales.

Fig. 2 es un diagrama de bloque de una arquitectura de software para una plataforma de la red, que incluye una Interfase del Programa de Aplicación (Application Program Interface - API).

Fig. 3 es un diagrama de bloque de al presentación de un subsistema apoyado por el API, así como las clases varias funciones del API.

Fig. 4 es un diagrama de bloque de una computadora ejemplar que pueda ejecutar todo o una parte de la arquitectura del software.

Figs. 5. 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 y 16 ilustran varias puestas en práctica de una interfaz de programación.

## DESCRIPCIÓN DETALLADA

Este divulgación trata sobre una Interfase de Programa de Apliacación (Application Program Interface - API) para una plataforma de la red sobre la cual los desarrolladores pueden aplicacions servicios Web. Más particularmente, un API ejemplo es descrito para los sistemas operativos que hacen uso una plataforma de la red, tal como el Sistema .NET™ creado por Microsoft Corporation. El Sistema .NET™ es una plataforma de software para los servicios Web y las aplicaciones Web implementadas en ambientes de cómputo distribuido. Representa la generación siguiente de cómputo en Internet, con estándares abiertos

de comunicación para comunicarse entre los servicios libremente agrupados en la Web que están colaborando para realizar una tarea particular.

En la implementación descrita, la plataforma de red utiliza XML (Extensible Markup Language), un estándar abierto para describir datos. XML es manejado por el consorcio mundial Word Wide Web (W3C). XML se utiliza para definir elementos de datos en una página de Web y documentos de negocio-a-negocio. XML utiliza una estructura similar de la etiqueta como HTML; sin embargo, mientras que el HTML define cómo se exhiben los elementos, XML define lo que contienen esos elementos. Las aplicaciones de HTML predefinieron las etiquetas, pero XML permite que las etiquetas sean definidas por el desarrollador de la página. Así, virtualmente cualquier artículo de datos se puede identificar, permitiendo que las páginas del Web funcionen como registros de la base de datos. Con el uso de XML y de otros protocolos abiertos, tales como protocolo de acceso de objeto simple (Simple Object Acces Protocol - SOAP), la plataforma de red permite la integración de una amplia gama de servicios que se pueden adaptar a las necesidades del usuario. Aunque las personalizaciones descritas adjuntas son descritas conjuntamente cón XML y otros estándares abiertos, estas no se requieren para la operación de la invención demandada. Otras tecnologías igualmente viables serán suficientes para implementar la invención aquí descrita.

DE acuerdo con lo aquí descrito, la frase interfase del programa de aplicación o API incluye las interfaces tradicionales que emplean llamadas de método o de función, así como las llamadas remotas (e.g., un proxy, una relación flexible) y los llamados de SOAP/XML.

Debe ser apreciado que algunas de las descripciones siguientes del namespace, descripciones de ciertas clases, interfaces, las enumeraciones y los delegados son

dejados en blanco. Descripciones más completas de estas clases, interfaces, enumeraciones y delegados se pueden encontrar en el tema de los discos compactos que almacenan el SDK referido arriba.

## EJEMPLO DE AMBIENTE DE RED

La fig. 1 muestra un ambiente de red 100 en el cual una plataforma de red, tal como el El Sistema .NET™, puede ser puesto en ejecución. El ambiente de red 100 incluye los servicios representativos del Web 102(1)..., 102(N), que proporcionan los servicios que se pueden acceder sobre una red 104 (e.g., Internet). Los servicios Web, referidos generalmente como número 102, son los componentes de aplicación programable que son reutilizables e interactivos programáticamente sobre la red 104, típicamente a través de los protocolos estandar de la industria, tales como, XML, SOAP, WAP (Wireless Aplication Protocol – Protocolo de Aplicación Inalámbrica), HTTP (protocolo de transporte de hypertext), y SMTP (Simple Mail Transfer Protocol) aunque otros medios de interacción con los servicios Web sobre la red pueden también ser utilizados, por ejemplo el tipo tecnología del objeto corredor ó el Remote Procedure Call (RPC). Un servicio del Web puede ser descrito en si mismo y se define a menudo en términos de formatos y orden de mensajes. Los servicios 102 del Web son accesibles directamente por otros servicios (según lo representado por el vínculo de comunicaciones 106) o un software de aplicación, tal como la aplicación Web 110 (según lo representado por los vínculos de comunicaciones 112 y 114). Cada servicio 102 del Web se ilustra como incluyendo unos o más servidores que ejecuten software para manejar los pedidos de servicios particulares. Tales servicios mantienen a menudo las bases de datos que almacenan la información

que se entregará de nuevo a los solicitantes. Los servicios del Web se pueden configurar para realizar una variedad de diversos servicios. Los ejemplos de los servicios del Web incluyen la verificación de la conexión, la notificación, el almacenamiento en la base de datos, la cuota de inventario, los directorios de localización, el mapeo, la música, la carpeta electrónica, el calendario-agenda, los listados del teléfono, las noticias y la información, juegos, tiquetes, etcétera. Los servicios Web se pueden combinar con cada uno con otras aplicaiones para construir experiencias interactivas inteligentes. El ambiente de red 100 también incluye los dispositivos representativos del cliente 120(1), 120(2), 120(3), 120(4)..., 120(M) que utilizan los servicios Web 102 (según lo representado por el vínculo de comunicaciones 122) y/o la aplicación Web 110 (según lo representado por los vínculos de comunicaciones 124, 126, y 128). Los clientes pueden comunicar el uno con el otro usando también protocolos estándares, según lo representado por ejemplo XML vínculo 130 entre los clientes 120(3) y 120(4).

Los dispositivos del cliente, referidos generalmente como número 120, se pueden poner en ejecución de muchas diversas maneras. Los ejemplos de las puestas en práctica posibles del cliente incluyen, sin la limitación, las computadoras portables, las computadoras inmóviles, Tablet PC, las cajas de televisions/set-top, los dispositivos de comunicación inalámbrica, los PDAs (ayudantes digitales personales), las consolas de juego, las impresoras, las fotocopiadoras, y otros dispositivos inteligentes. La aplicaión Web 110 es una aplicación diseñada para correr sobre la plataforma de red y puede utilizar los servicios 102 del Web al manejar y responder peticiones de los clientes 120. La aplicación Web 110 se compone de uno o más software de aplicación que funcionan sobre el sistema de programación 132, los cuales se están ejecutando en

uno o más servidores 134 u otros sistems de cómputo. Observe que una porción del uso 110 del Web puede residir realmente en uno o más de los clientes 120. Alternativamente, la plicación Web 110 puede coordinar con el otro software en los clientes 120 para lograr realmente sus tareas.

El sistema de programación 132 es la estructura que apoya las aplicaciones y los servicios desarrollados por los desarrolladores de aplicaciones. Esto permite el desarrollo del multi-lenguage y la integración completa apoyanda en múltiples lenguages. Esto soporta protocolos abiertos, tales como SOAP, y encapsula los servicios del modelo objeto y del sistema operativo subyacentes. El sistema proporciona un ambiente robusto y seguro de ejecución para los lenguajes de programación múltiple y ofrece librerías seguras, integradas de la clase. El sistema 132 es una arquitectura multi-capa que incluye un API capa 142, un tiempo de pasada del lenguaje común (Common Language Runtime - CLR) capa 144, y un, y una capa de servciios del sistema operativo 146 de system/services. Esta capa de arquitectura permite actualizaciones y modificaciones a las varias capas sin la afectación de otras porciones del sistema. Una especificación del lenguaje común (CLS) 140 permite que los diseñadores de varios lenguages escriban el código que puede tener acceso a funcionalidad subyacente de la biblioteca. La especificación 140 funciona como un contrato entre los diseñadores del lenguage y los diseñadores de la biblioteca que pueden ser utilizados para promover interoperabilidad del lenguage. Adhiriendo al CLS, las librerías escritas en un lenguage pueden ser directamente accesibles a los módulos del código escritos en otros lenguages para alcanzar la integración completa entre los módulos del código escritos en un lenguage y los módulos del código escritos en otro lenguage. Un ejemplo de una puesta en práctica detallada de un CLS se describe en un estándar

de ECMA creado por los participantes en ECMA TC39/TG3. Dirigen al lector a la página Web de ECMA en www.ecma.ch. La capa API 142 presenta grupos de las funciones que las aplicaciones 130 pueden llamar para tener acceso a los recursos y a los servicios proporcionados por la capa 146. Exponiendo las funciones API para una plataforma de red, los desarrolladores de aplicaciones pueden crear las aplicaciones Web para los sistemas de cómputo distribuido que hacen el uso completo de los recursos de la red y de otros servicios Web, sin necesitad de entender los (trabajos internos de la red) interworkings complejos y de cómo esos recursos de la red realmente funcionan o se hacen disponibles. Por otra parte, las aplicaciones Web se pueden escribir en cualquier número de lenguajes de programación, y traducir a un lenguage intermedio apoyado por el tiempo de pasada (runtime) común 144 e incluido como parte de la especificación de lenguaje común 140. De esta manera, la capa API 142 puede proporcionar los métodos para una variedad amplia y diversa de aplicaciones.

Adicionalmente, el sistema 132 se puede configurar para soportar las llamadas API puestas por las aplicaciones remotas que se ejecutan remotamente de los servidores 134 que reciben el sistema. Las aplicaiones representativas 148(1) y 148(2) residiendo en clientes 120(3) y 120(M), respectivamente, pueden utilizar funciones API haciendo llamadas directamente, o indirectamente, a la capa API 142 sobre la de red 104. El sistema se puede también poner en ejecución en los clientes. El cliente 120(3) representa la situación donde un sistema 150 se pone en ejecución en el cliente. Este sistema puede ser idéntico al sistema establecido en el servidor (server-based) 132, o modificado para los propósitos del cliente. Alternativamente, el sistema establecido en el cliente (cliente-based) puede ser condensado en caso que el cliente sea un dispositivo limitado o dedicado de la

función, tal como un teléfono portátil, un PDA, computador de mano personal, u otro dispositivo de comunicación o de cómputo.

**SISTEMA DE PROGRAMACIÓN DE LOS DESARROLLADORES**

La fig. 2 muestra el marco de programación 132 más detalladamente. Los las especificaciones de lenguaje común (CLS) capa 140 que soportan las aplicaciones escritas en una variedad de lenguages 130(1), 130(2), 130(3), 130(4)..., 130(K). Tales lenguajes de aplicación incluyen Visual Basic, C++, C #, COBOL, Jscript, Perl, Eiffel, Python, etc. La especificación de lenguaje común 140 incluye un subconjunto de características o de reglas sobre las características que, si están seguidas, permiten que varios lenguages se comuniquen. Por ejemplo, algunos lenguages no soportan un tipo dado (e.g., un "int*" type) que de otra manera podría ser soportado por el tiempo de pasada del lenguaje común 144. En este caso, la especificación del lenguaje común 140 no incluye el tipo. Por otra parte, las variables que son soportadas por todos o por la mayoría de los lenguages (e.g., "int[ ]") esta incluida en la especificación de lenguaje común 140 y es una librería que los desarrolladores pueden usar libremente y con la seguridad de que los lenguajes puden manejarla. Esta habilidad para comunicar resulta en una inconsútil integración entre los módulos de código escritos en un lenguaje y los módulos de código escritos en otro lenguaje. Desde que los lenguajes están destinados a tareas específicas, la completa integración entre los lenguajes permite a un desarrollador seleccionar un lenguaje en particular para un módulo de código específico con la habilidad de utilizar un módulo de código con módulos escritos en diferentes lenguajes. El lenguaje de tiempo de pasada común 144 permite el desarrollo de multi-lenguaje completo, a trvés de la herencia del lenguaje, y ofrece un róbust y seguro ambiente para los lenguajes de programación

múltiple. Para más información sobre la especificación de lenguaje común 140 y el tiempo de pasada de lenguaje común 144, el lector puede consultar el compendio de aplicaciones titulada "Method and System for Compiling Multiple Languages", - Método y Sistema para la Compilación de Múltiples Lenguajes -, editado el 6/21/2000 (número de serie 09/598,105) y "Unified Data Type System and Method" – Sistema de Tipos de Datos Unificado – editado el 7/10/2000 (número de serie 09/613,289), los cuales están incluidos por referencia.

El sistema 132 encapsula el sistema operativo 146(1) (e.g., sistemas operativos de la marca Windows®) y los servicios modelo del objeto 146(2) (e.g., el modelo del componente objeto (COM) o COM distribuido). El sistema operativo 146(1) proporciona funciones convencionales, tales como administración de archivo, notificación, manipulación de eventos, interfaces de usuario (e.g., ventanas, menús, diálogos, etc.), seguridad, autentificación, verificación, los procesos e hilos de comunicación, administración de la memoria, etc. Los servicios del modelo objeto 146(2) proporcionan la interconexión con otros objetos para realizar varias tareas. Las llamadas hechas a la capa 142 API se dan a la capa de tiempo de pasada (runtime) 144 del lenguaje común para la ejecución local por el sistema operativo 146(1) y/o los servicios modelo del objeto 146(2). El API 142 agrupa funciones del API en nombres de espacio (namespaces) múltiples. Los nombres de spacio (Namespaces) esencialmente definen una colección de clases, de interfaces, de delegados, de enumeraciones, y de las estructuras, que colectivamente se llaman "Types"-"tipos", que proporcionan un sistema específico de funcionalidad relacionada. Una clase representa todos los datos asignados manejados que tienen semántica de asignación de referencia. Un delegado es un indicador orientado a objeto de la función. Una enumeración es

una clase especial de tipo del valor que represente constantes nombradas. Una estructura representa los datos asignados estáticos que tienen semántica de asignación de valor. Una interfaz define un contrato que otros tipos "types" puedan implementar.

Usando espacios de nombre (namespaces), un diseñador puede organizar un sistema de tipos en un espacio de nombre jerárquico. El diseñador puede crear grupos múltiples del sistema de tipos, con cada grupo conteniendo por lo menos un tipo que exponga funcionalidad lógicamente relacionada. En la implementación del ejemplo, el API 142 se organiza para incluir tres namespaces de la raíz. Debe ser observado que aunque solamente tres namespaces de la raíz se ilustran en fig. 2, los namespaces adicionales de la raíz se pueden también incluir en API 142. Los tres namespaces de la raíz ilustrados en API 142 son: un primer namespace 200 para un subsistema de presentación (que incluye un namespace 202 para un shell de interfase de usuario), un segundo namespace 204 para los servicios Web, y un tercer namespace 206 para un sistema de archivos. A cada grupo puede entonces ser asignado un nombre. Por ejemplo, tipos en el namespace 200 del subsistema de presentación puede ser asignado el nombre "Windows", y tipos adentro el sistema de archivos namespace 206 puede ser asignado el nombre de "storage" - "almacenamiento". Los grupos nombrados pueden ser organizados debajo de un solo namespace de la "raíz general o principal" para las APIs de nivel del sistema, tal como un sistema total namespace. Seleccionando y prefijando un identificador del nivel superior, los tipos en cada grupo se pueden referir fácilmente por un nombre jerárquico que incluya el identificador seleccionado del nivel superior prefijado al nombre del grupo que contiene el tipo. Por ejemplo, tipos dentro del sistema de ficheros namespace 206 puede ser referido usando el nombre jerárquico

"System.Storage". De esta manera, los namespaces individuales 200, 204, y 206 se convierten en ramas importantes de apagado del sistema namespace y pueden llevar una designación donde los namespaces individuales se prefijan con un designador, tal como un "Sistema." prefijo.

El namespace 200 del subsistema de presentación pertenece a la programación y al desarrollo de contenido. Provee los tipos que permiten la generación de aplicaiones, de los documentos, de las presentaciones de los medios y de otro contenido. Por ejemplo, el namespace 200 del subsistema de la presentación proporciona un modelo de programación que permite que los desarrolladores obtengan servicios del sistema operativo 146(1) y/o de los servicios modelo del objeto 146(2).

El namespace 202 del shell pertenece a la funcionalidad de la interfase de usuario. Provee los tipos que permiten a los desarrolladores encajar la funcionalidad de la interfase de usuario en sus aplicaciones, y permite mucho más que los desarrolladores amplíen la funcionalidad de la interfase de usuario.

El namespace 204 de los servicios Web pertenece a una infraestructura para permitir la creación de una variedad amplia de aplicaiones, e.g. aplicaciones tan simples como un chat que funcione entre dos puntos en Intranet, y/o tan complejos como un servicio scalable del Web para millones de usuarios. La infraestructura descrita es ventajosa y altamente variable en que una necesidad solamente utiliza esas piezas que sean apropiadas a la complejidad de una solución particular. La infraestructura proporciona una base para la construcción de aplicaciones basadas en mensajes (message-based) de varias escalas y complejidad. La infraestructura o el sistema proporciona APIs para la mensajería básica, la mensajería segura, la mensajería confiable y la mensajería tramitada. En la personalización descrita

abajo, las APIs asociadas se han descompuesto en factores en una jerarquía de namespaces de una manera que se ha hecho a mano cuidadosamente para balancear utilidad, funcionalidad, amplitud y versatilidad.

El sistema de ficheros namespace 206 pertenece al almacenamiento. Provee los tipos que permiten almacenamiento y la recuperación de información. Además del sistema 132, las herramientas de programación 210 se proporcionan para asistir al desarrollador en la construcción de servicios y/o aplicaiones Web. Un ejemplo de las herramientas de programación 210 es Visual Studio™, un conjunto de herramientas de programación multi-lenguaje ofrecidas por Microsoft Corporation.

NAMESPACES API DE LA RAIZ

La fig. 3 muestra una porción del subsistema 200 de presentación más detalladamente. En una personalización, los namespaces se identifican según una convención de nombramiento jerárquica en la cual las cadenas de nombres se concatenen con períodos. Por ejemplo, el namespace 200 del subsistema de la presentación es identificado por el nombre "System.Windows" de la raíz. Dentro de "Sytem.Windows" el namespace es otro namespace para varios controles, identificados como "System.Windows.Controls", que identifica más allá otro namespace para los primitivos (no mostrados) conocidos como "System.Windows.Controls.Primitives". Con esta convención de nombramiento en mente, lo que sigue provee de una descripción general de los namespaces seleccionados del API 142, aunque otras convenciones de nombramiento podrían ser utilizadas con efecto igual.

Según lo demostrado en la fig. 3, el subsistema 200 de la presentación incluye namespaces múltiples. Los namespaces mostrados en la fig. 3 representan

una personalización particular del subsistema 200 de la presentación. Otras personalizaciones del subsistema 200 de presentación pueden incluir uno o más namespaces adicionales o pueden omitir uno o más de los namespaces mostrados en fig. 3. El subsistema 200 de presentación es el namespace de la raíz para mucha de la funcionalidad de la presentación del API 142. Un namespace 310 de controles incluye los controles usados para construir una pantalla de la información, tal como una interfaz de usuario, y las clases que permiten que un usuario interactúe con una aplicaión. Los controles del ejemplo incluyen el "botón" que crea un botón en la pantalla, "RadioButton" que genera un botón del radio-estilo (radio-style) en la pantalla, el "Menú" que crea un menú en la pantalla, "ToolBar" que cree una barra de herramientas en la pantalla, "Image" que genera una imagen en la pantalla y el "TreeView" que crea una vista jerárquica de la información.

Ciertos controles son creados jerarquizando y arreglando elementos múltiples. Los controles tienen un modelo lógico que oculta los elementos usados para crear los controles, de tal modo que simplifica el modelo de programación. Los controles pueden ser diseñados y esquematizados por un desarrolladro o un usuario (e.g., modificando el aspecto y la pariencia de acuerdo con los requisitos particulares de los botones de la interfaz de usuario). Algunos controles tienen componentes direccionables que permitan que un individuo ajuste el estilo de los controles individuales. Además, los controles pueden ser sub clasificados y ampliados por los desarrolladores de aplicaión y de componentes. Los controles son interpretados usando gráficos de vector de tal forma que pueden ser redimensionados para suplir los requisitos de una interfase particular o de otra pantalla. Los controles son capaces de utilizar la animación para realzar, por

una personalización particular del subsistema 200 de la presentación. Otras personalizaciones del subsistema 200 de presentación pueden incluir uno o más namespaces adicionales o pueden omitir uno o más de los namespaces mostrados en fig. 3. El subsistema 200 de presentación es el namespace de la raíz para mucha de la funcionalidad de la presentación del API 142. Un namespace 310 de controles incluye los controles usados para construir una pantalla de la información, tal como una interfaz de usuario, y las clases que permiten que un usuario interactúe con una aplicaión. Los controles del ejemplo incluyen el "botón" que crea un botón en la pantalla, "RadioButton" que genera un botón del radio-estilo (radio-style) en la pantalla, el "Menú" que crea un menú en la pantalla, "ToolBar" que cree una barra de herramientas en la pantalla, "Image" que genera una imagen en la pantalla y el "TreeView" que crea una vista jerárquica de la información.

Ciertos controles son creados jerarquizando y arreglando elementos múltiples. Los controles tienen un modelo lógico que oculta los elementos usados para crear los controles, de tal modo que simplifica el modelo de programación. Los controles pueden ser diseñados y esquematizados por un desarrolladro o un usuario (e.g., modificando el aspecto y la pariencia de acuerdo con los requisitos particulares de los botones de la interfaz de usuario). Algunos controles tienen componentes direccionables que permitan que un individuo ajuste el estilo de los controles individuales. Además, los controles pueden ser sub clasificados y ampliados por los desarrolladores de aplicaión y de componentes. Los controles son interpretados usando gráficos de vector de tal forma que pueden ser redimensionados para suplir los requisitos de una interfase particular o de otra pantalla. Los controles son capaces de utilizar la animación para realzar, por

ejemplo, la sensación interactiva de un interfase de usuario para mostrar acciones y reacciones.

El namespace 310 de controles incluye uno o más paneles, que son los controles que miden y ordenan a sus hijos (e.g., elementos jerarquizados). Por ejemplo, un panel de "DockPanel" ordena sus hijos ubicando a cada hijo en la parte de arriba, izquierda, o en la la parte inferior derecha de la pantalla, y llena el espacio restante con otros datos. Un panel particular puede ubicar los menús y las barras de gerramientas en la parte superior de la pantalla, una barra de estado en la parte inferior de la pantalla, una lista de carpetas al lado izquierdo de la pantalla, y llena el resto del espacio de una lista de mensajes.

Según lo mencionado arriba, System.Windows.Controls.Primitives es un namespace que incluye los controles múltiples que son componentes usados típicamente por los desarrolladores de controles en el namespace de System.Windows.Controls y por los desarrolladores que crean sus propios controles. Ejemplos de estos componentes incluyen el "The thumb and RepeatButton" (El pulgar y el botón de repetición).

RepeatButton ". La barra de desplazamiento "ScrollBar", otro componente, es creado usando cuatro botones de repetición (uno para la "línea arriba", uno para la "línea abajo", uno para la "página arriba", y uno para la "página abajo") y un "pulgar" para arrastrar la vista actual a otra localización en el documento. En otro ejemplo, El desplazamiento de visualización "ScrollViewer" es un control creado usando dos "ScrollBars" y un "ScrollArea" para proporcionar un área de desplazamiento.

La lista siguiente contiene las clases del ejemplo expuestas por el namespace del System.Windows.Controls. Estas clases permiten que un usuario

interactúe con, por ejemplo, una aplicación a través de varias entradas y capacidades de salida así como capacidades adicionales de la pantalla.

- La tecla de acceso -AccessKey- es un elemento del FrameworkElement que envuelve un carácter, indicando que debe recibir las señales del teclado que denotan el carácter como mnemónica del teclado. Por defecto, la señal del teclado es una raya (underline).
- Audio - Elemento Audio.
- Frontera – (Border) dibuja una frontera, un fondo, o ambos alrededor de otro elemento.
- Botón - representa el componente estándar del botón que intrínsecamente reacciona al evento del Click.
- Lona – (Canvas) define un área dentro de la cual un usuario pueda colocar explícitamente elementos del hijo por coordenadas concerniente al área de la lona.
- CheckBox – Caja de Chequeo utilice un CheckBox para dar al usuario una opción, tal como verdadero/falso. CheckBox permite que el usuario elija de una lista de opciones. Los controles de CheckBox dejan al usuario escoger una combinación de opciones.
- CheckedChangedEventArgs (Evento de chequeo de cambio de argumentos) - La clase CheckedChangedEventArgs contiene la información adicional sobre el evento CheckedChangedEvent.
- CheckStateChangedEventArgs (Evento de chequeo de cambio del estado de los argumentos) - Esta clase de

CheckStateChangedEventArgs contiene la información adicional sobre el evento de CheckStateChangedEvent.

- ClickEventArgs (Evento de tecleo "click" de argumentos) - contiene la información sobre el evento del tecleo (click).
- ColumnStyle - representa un objeto cambiable de ColumnStyle (estilo de columna).
- ColumnStyles - Objeto cambiable de IList del patrón que es una colección de elementos cambiables.
- Control de ComboBox - de ComboBox.
- ComboBoxItem - Control que implementa un item seleccionable dentro de un ComboBox.
- ContactPickerDialog - permite que un usuario seleccione uno o más contactos.
- ContactPropertyRequest – (Requerimiento de propiedad de contacto). Permite que una aplicación solicite la información sobre una característica del contacto con un ContactPickerDialog. Esta clase no puede ser heredada.
- ContactPropertyRequest Collection - Representa una colección de los objetos de ContactPropertyRequest.
- ContactSelection - Información sobre un contacto seleccionado del Sistema de Archivos de Microsoft Windows®, código-nombrado "WinFS" o Directorio Activo de Microsoft (Microsoft Active Directory®).
- ContactSelectionCollection - Representa una colección de los objetos de ContactSelection.

- ContactTextBox – Es un control de edición que soporta recolección de contactos o características de contactos.
- ContactTextBoxSelectionChangedEventArgs - Argumentos para el ContactTextBoxSelectionChanged.
- ContactTextBoxTextChangedEventArgs - Argumentos para el evento ContactTextBoxTextChanged.
- ContactTextBoxTextResolvedEventArgs - Argumentos para el evento TextResolvedToContact.
- ContentChangedEventArgs - Argumentos del evento para el ContentChangedEvent.
- ContentControl – Es la clase base para todos los controles con una parte sencilla de contenido.
- ContentPresenter – Presentador de Contenido. ContentPresenter es utilizado dentro del estilo de un control de contenido para denotar el lugar en el árbol visual de control (plantilla de cromo) donde el contenido será adicionado.
- ContextMenu - Control que define un menú de opciones para que los usuarios invoquen.
- ContextMenuEventArgs – Son los datos enviados sobre un ContextMenuEvent.
- Control - Representa la clase base para todos los elementos interactivos de usuario. Esta clase proporciona un conjunto base de características para sus subclases.

- Decorador - Clase base para los elementos sobre los cuales se aplican los efectos o alrededor de un solo elemento del hijo, tal como frontera.
- DockPanel - Define un área dentro de la cual usted pueda arreglar elementos del hijo horizontalmente o verticalmente, relacionado uno con el otro.
- DragDeltaEventArgs - Esta clase de DragDeltaEventArgs contiene la información adicional sobre el evento DragDeltaEvent.
- FixedPanel - Es el elemento de la raíz usado en documentos de formato-fijo para contener las páginas fijas para la paginación. FixedPanel exhibe una página de contenido paginado uno a la vez o como páginas apiladas desplazables.
- FlowPanel - FlowPanel se utiliza para romper, para envolver, y para alinear el contenido que excede la longitud de una sola línea. FlowPanel proporciona las características del rompimiento de línea line-breaking y de la alineación que pueden ser utilizadas cuando el flujo del contenido del contenedor, texto por ejemplo, es probable que exceda la longitud de una sola línea.
- Frame – (Capítulo) – Es un área que puede cargar el contenido de otro árbol.
- Generador - Generador - Es el objeto que genera un UI (Interfased de Usuario) a nombre de un ItemsControl, trabajando bajo supervisión de un GeneratorFactory.
- GeneratorFactory - Un GeneratorFactory es responsable de generar la UI a nombre de un ItemsControl. Mantiene la asociación entre los

items en el ItemsCollection de control (visión plana) y los UIElements correspondientes. El control del contenedor de item (item-container) puede pedir la fabricación de un generador, que hace la generación real de UI.

- GridPanel - Define un área de grilla que consiste de columnas y filas.
- HeaderItemsControl - La clase base para todos los controles que contienen múltiples items y tienen un encabezado.
- HorizontalScrollBar - La clase horizontal de ScrollBar (Barra de Desplazamiento).
- HorizontalSlider - La clase horizontal del deslizador.
- HyperLink – Hipervínculo - La clase del hyperLink implementa el control de la navegación. El presentador por defecto es TextPresenter.
- Image - Imagen - Proporciona una manera fácil de incluir una imagen en un documento o una aplicación.
- IncludeContactEventArgs – Argumentos transferidos a los manejadores del evento ContactPickerDialog.IncludeContact.
- ItemCollection – Colección de Ítems - Mantiene una colección de items discretos dentro de un control. Proporciona los métodos y las características que permiten cambiar el contenido de la colección y la obtención de datos sobre el contenido.
- ItemsChangedEventArgs - El evento de ItemsChanged es levantado por un GeneratorFactory para informar las disposiciones que la colección de items ha cambiado.

- ItemsControl – Control de Ítems - Es la clase base para todos los controles que tienen hijos múltiples.
- ItemsView – Vista de Ítems - Proporciona una vista plana de una colección de ítems (ItemCollection).
- KeyboardNavigation – Navegación de Teclado – La clase de KeyboardNavigation proporciona los métodos para la navegación lógica (Tab - lengüeta) y direccional (arroz - flecha) entre los controles objetivo.
- ListBox - Control que implementa unaa lista de items seleccionables.
- ListItem – Lista de Item - Control que implementa un item seleccionable dentro de un ListBox.
- Menu - Control que define un menú de opciones para que los usuarios invoquen.
- MenuItem - Un item hijo del menú. MenuItems pueden ser seleccionados para invocar comandos. MenuItems pueden ser separadores. MenuItems pueden ser encabezados para los submenus. MenuItems pueden ser seleccionado o no.
- PageViewer – Visor de Página - Representa un control compuesto de la visión del documento (document-viewing) que contiene un control de paginación, una barra de harramientas, y un control de la página de la barra.
- PaginationCompleteEventArgs – Es el evento de argumentos las para el evento de paginación completa (PaginationCompleteEvent.

- PaginationProgressEventArgs – Es el evento de argumentos para el PaginationProgressEvent.
- Pane - Cristal - Proporciona una manera de definir características de la ventana en un lenguaje de mayor beneficio (e.g., el "XAML") sin cargar una nueva ventana.
- Panel - Proporciona una clase base para todos los elementos del panel. Para instanciar un elemento del panel, utiliza la clase concreta derivada.
- RadioButton – Botón de Radio - RadioButton implementa un botón de opción en ejecución con dos estados: verdadero o falso.
- RadioButtonList – Botón de Lista de Radio - Este control sirve como control de agrupación para los RadioButtons y es la pieza que maneja la exclusividad mutua de RadioButton. El RadioButtonList hereda del selector. El RadioButtonList es esencialmente un solo selector de Modo Selección (SelectionMode) y el concepto de la selección (del selector) es desapropiado de la característica comprobada del RadioButton que está agrupando.
- RowStyle – Estilo de linea - Elementos cambiables del patrón cambiable.
- RowStyles – Estilos de Línea - Objeto cambiable de IList del patrón que es una colección de elementos cambiables.
- ScrollChangeEventArgs - El ScrollChangeEventsArgs describe un cambio de estado del movimiento en sentido vertical.
- ScrollViewer –

- SelectedItemsCollection – Colección de Ítems seleccionados – Es un contenedor para los artículos seleccionados en un selector.
- SelectionChangedEventArgs – Son las entradas a un manejador de de eventos cambiados de selección.
- SimpleText – Texto Simple - SimpleText es un elemento de texto ligero, multilínea, de formato simple, previsto para el uso en escenarios de la interfase de usuario (UI).
- SimpleText expone varias de las mismas características del formato que el texto y se puede utilizar a menudo para un aumento del funcionamiento en el coste de una cierta flexibilidad.
- StyleSelector – Selector de Estilo - StyleSelector permite que el escritor del app proporcione lógica de selección de estilo personal. Por ejemplo, con un error (bug) de clase como el contenido, utiliza un estilo particular para los errores Pri1 y un estilo diferente para los errores Pri2. Un escritor de aplicaión puede eliminar el método de SelectStyle en una clase derivada del selector y asignar un caso de esta clase a la característica de StyleSelector en la clase de ContentPresenter.

- Text – Texto - Representa un control del texto que permite la representación de formatos múltiples de texto. El texto se utiliza mejor dentro de una aplicación UI; escenarios más avanzados de texto se benefician del grupo de características adicionales del TextPanel. En la mayoría de los casos donde se requiere la ayuda

relativamente simple del texto, el texto es el elemento preferido debido a su naturaleza y gama de características.

- TextBox - representa el control que proporciona una región editable que acepte la entrada de texto.
- TextChangedEventArgs - La clase TextChangedEventArgs representa un tipo de RoutedEventArgs que son relevantes a los eventos levantados por el TextRange.SetText().
- TextPanel – Son formatos, tamaños, y dibujos de texto. TextPanel soporta líneas múltiples de texto y formatos de texto múltiples.
- ToolTip – Es un control para mostrar información cuando el usuario asoma sobre un control.
- ToolTipEventArgs – Son los datos enviados en un ToolTipEvent.
- TransformDecorator – Decorador de Transformación - TransformDecorator contiene a hijo y aplica una transformación específica sobre él. TransformDecorator implementa la lógica para medir y acomoda al hijo en sus coordenadas locales (pre-transformación - pre-transforme) tales que después de la transformación, los ajustes del hijo encajan firmemente dentro del espacio del decorador y utiliza el área máxima. El hijo por lo tanto no necesita tener ningún conocimiento que una transformación se ha aplicado a él.
- UIElementCollection - Una UIElementCollection es una colección solicitada de UIElements.

- ValueChangedEventArgs - Esta clase de ValueChangedEventArgs contiene la información adicional sobre el evento de ValueChangedEvent.
- VerticalScrollBar – Es la clase vertical de ScrollBar.
- VerticalSlider - Es la clase vertical del deslizador.
- Vídeo – Ejecuta un flujo de archivos de video o audio en un rectángulo especificado dentro del sistema coordinado actual del usuario.
- VisibleChangedEventArgs - La clase de VisibleChangedEventArgs contiene la información adicional sobre el evento de VisibleChangedEvent.
- El espacio de nombre (namespace) del control del sistema Windows (System.Windows.Controls) también contiene varias enumeraciones. La lista siguiente contiene las enumeraciones del ejemplo asociadas con el namespace de System.Windows.Controls.
- CharacterCase – Caso de Carácter - Especifica la caja de caracteres en un control TextBox cuando se escribe el texto.
- CheckState – Estado de chequeo - Especifica el estado de un control, tal como una caja de chequeo, que puede estar seleccionada o no, o configurada en un estado indeterminado.
- ClickMode – Modo de Tecleo (click) - Especifica cuando el evento click debería encende.
- ContactControlPropertyPosition – Posición de Propiedad del control de contacto - Controla la posición y el despliegue de la característica de contacto.

- ContactPickerDialogLayout - Especifica cómo el ContactPickerDialog debería desplegar las características seleccionadas.
- ContactPropertyCategory – Categoría Propietaria de Contacto - Especifica qué valor a tratar por defecto en el caso donde una característica tiene múltiples valores de los cuales el usuario podría elegir. Por ejemplo, si "Work" se especifica como la categoría preferida cuando la petición de una característica del número de teléfono del ContactPickerDialog, y el usuario selecciona un contacto con ambos un número de teléfono del trabajo y el número de teléfono de la casa, el número de teléfono del trabajo aparece como la selección por defecto. El usuario puede entonces utilizar el UI para elegir el número de teléfono de la casa en lugar de otro.
- ContactPropertyType – Tipo de Propiedad de Contacto - Especifica una característica de un contacto por la cual el ContactPickerDialog puede preguntar al usuario.
- ContactType – Tipo de Contacto - Especifica qué tipos de contacto para mostrarlo en en el ContactPickerDialog.
- Direction - Dirección - Esta enumeración es utilizada por el GeneratorFactory y el Generador para especificar la dirección en la cual el generador produce UI.
- Dock - Muelle - Especifica la posición del muelle de un elemento hijo dentro de un DockPanel.
- GeneratorStatus – Generador de Estatus - Esta enumeración es utilizada por el GeneratorFactory para indicar su estado.

- KeyNavigationMode – Clave de Modo de Navegación - El tipo de característica de nagación mediante tabulador (TabNavigation) especifica cómo el contenedor moverá el foco cuando ocurre la navegación del tabulador.
- MenuItemBehavior – Comportamiento del Item del Menú - Define los diversos comportamientos que un MenuItem podría tener.
- MenuItemType – Tipo de Item del Menú - Define los diversos tipos de la ubicación de MenuItems.
- Orientation - Orientación - Tipos de la orientación del deslizador.
- PageViewerFit – Acceso del Visor de Página - Selecciona cómo las páginas se deben encajar en el área del cliente del PageViewer – Visor de Página.
- PageViewerMode – Modo de Visor de Página - Selecciona el actual modo de PageViewer – Visor de Página reflejado en el modo dropdown – soltar hacia abajo.
- ScrollerVisibility – Visibilidad de Enrollador - ScrollerVisibilty define el comportamiento de la visibilidad de una barra de desplazamiento.
- SelectionMode – Modo de Selección - Especifica el comportamiento de la selección para el ListBox (Lista de Caja).

"La posición" es una estructura de ejemplo asociada al namespace de System.Windows.Controls (Controles del Sistema Windows). Un usuario del Generador describe posiciones usando esta estructura. Por ejemplo: Para comenzar a generar adelante del principio

de la lista del artículo, especifique la posición (-1, 0) y la dirección hacia adelante. Para comenzar a generar al revés del extremo de la lista, especifique la posición (-1, 0) y la dirección al revés. Para generar los items después del elemento con el índice k, especifique la posición (k, 0) y la dirección hacia adelante.

La lista siguiente contiene a delegados del ejemplo asociados al namespace de System.Windows.Controls.

- CheckedChangedEventHandler – Manejador de Evento Cambiado Seleccionado – Este delegado es usado por los manejadores del evento CheckedChangedEvent (evento Cambiado Seleccionado).
- CheckStateChangedEventHandler – Manejador de Evento Cambiado de Estado de Selección – Este delegado es utilizado por los manejadores del CheckStateChangedEvent (Evento Cambiado del Estado de Selección).
- ClickEventHandler – Manejador del evento Clic - Representa los métodos que manejan el evento Click (tecleo).
- ContactTextBoxSelectionChangedEventHandler – Manejador de evento Cambiado de la Selección del Contacto de la Caja de Texto. Un manejador del delegado para el evento del ContactTextBoxSelectionChanged.
- ContactTextBoxTextChangedEventHandler – Un manejador del delegado para el evento de ContactTextBoxTextChanged.
- ContactTextBoxTextResolvedEventHandler - Un manejador del delegado para el evento de TextResolvedToContact.

- ContentChangedDelegate – Delegado de contenido Cambiado – Es el delegado para el ContentChangedEvent.
- ContextMenuEventHandler – Manejador de evento del contexto del Menú - El tipo de retorno de llamada para dirigir un ContextMenuEvent (Evento de Contexto de Menú).
- DragDeltaEventHandler – Manejador de Evento de Cambio de Arrastre – Este delegado es utilizado por los manejadores del evento DragDeltaEvent (evento de Cambio de Arrastre).
- IncludeContactEventHandler – Manejador de Evento de Inclusión de Contacto – Manejador para el evento de ContactPickerDialog.IncludeContact.
- ItemsChangedEventHandler – Manejador de Evento de Ítems Cambiados – Es el delegado a utilizar por los manejadores que reciben ItemsChangedEventArgs (Argumentos de Eventos de Ítems Cambiados).
- OpenedEventHandler – Manejadores de Evento Abierto – Es el manejador para el evento de ContactPickerDialog.Opened.
- PaginationCompleteDelegate – Delegado de Paginación Completa – Es el delegado para el evento de PaginationCompleteEvent.
- PaginationProgressDelegate – Delegado de Paginación en Progreso – Es el delegado para el PaginationProgressEvent.

- ScrollChangeEventHandler – Manejador de Evento de Cambio en el Scroll (rollo) – Este delegado es utilizado por los manejadores del evento ScrollChangeEvent.
- SelectionChangedEventHandler – Manejador de Evento de Selección Cambiada – Es el tipo de delegado para manejar una selección cambió evento.
- TextChangedEventHandler – Manejador de Evento de Texto Cambiado – Es el delegado a utilizar para los manejadores que reciben TextChangedEventArgs (Argumentos de Evento de Texto Cambiado).
- ToolTipEventHandler – Es el tipo de retorno de llamada para dirigir un ToolTipEvent.
- ValueChangedEventHandler – Este delegado es utilizado por los manejadores del evento ValueChangedEvent.
- VisibleChangedEventHandler – Este delegado es utilizado por los manejadores del evento VisibleChangedEvent utiliza a este delegado.

Otro namespace, System.Windows.Controls.Atoms (Atomos de controles del sistema Windows), es un secundario-namespace (sub-namespace) del namespace System.Windows.Controls. System.Windows.Controls.Atoms incluye controles asociados, también argumentos y manejadores de evento. La lista siguiente contiene las clases del ejemplo asociadas al namespace System.Windows.Controls.Atoms.

- PageBar – Barra de Página - Representa un control enrollable de la paginación.
- PageElement – Elemento de Página – Suministra una página específica del contenido paginado. La página que se suministrará es especificada por la característica de recurso de página (PageSource).
- PageHoveredEventArgs – Argumentos de Evento de Cubierta de Página - PageHoveredEventArgs proporciona la información alrededor donde está mostrando el indicador del ratón.
- PageScrolledEventArgs – Argumentos de Evento de Página Enrollable - El PageScrolledEventArgs contiene la información que pertenece al evento de PageScrolled.
- PageSelectedEventArgs – Argumentos de Evento de Página Seleccionada - Se activa el PageSelectedEvent cuando se hace una nueva selección de la gama de fila/columna (row/column).
- PageSelector – Selectror de página – El PageSelector: Permite que el usuario seleccione una gama de filas/columnas (rows/columns) de páginas para ser mostradas.
- PageSource – Fuente de Página - Identifica la fuente del contenido que se paginará. También proporciona características y los métodos para ajustar al formato el contenido paginado.

La lista siguiente contiene ejemplos de delegados asociados al namespace de System.Windows.Controls.Atoms.

- PageHoveredEventHandler – Manejador de evento de Cubierta de Página - Este delegado es utilizado por los manejadores del evento PageHoveredEvent.
- PageScrolledEventHandler – Manejador de Evento de Página Enrollada – Este delegado es utilizado por los manejadores de PageSelectedEventHandler.
- PageSelectedEventHandler – Manejador de Evento de Página Seleccionada – Este delegado es utilizado por los manejadores del evento PageSelectedEvent.

Un namespace de System.Windows.Controls.Primitives es otro secundario-namespace (sub-namespace) del namespace de System.Windows.Controls. Según lo mencionado arriba, el secundario-namespace de los primitivos incluye los controles que se piensan para ser utilizados como primitivos por otros controles más complejos. La lista siguiente contiene ejemplos de clases asociadas con el namespace System.Windows.Controls.Primitives

- ButtonBase – Botón Base – Cuando es rechazado en una clase derivada, define los acontecimientos y las características relevantes, y proporciona manejadores para los acontecimientos relevantes de la entrada.
- Popup – Aparece de Repente - Es una ventana que se crea y aparece de repente que contiene el contenido.
- RangeBase – Rango Base - Representa la clase base para los elementos que tienen una rango específico. Los ejemplos de tales elementos son barras de desplazamiento y barras de progreso. Esta clase define los

acontecimientos y las características relevantes, y proporciona manejadores para los eventos.

- RepeatButton – Botón de Repetición - RepeatButton agrega la repetición de la semántica cuando ocurre el evento de click (tecleo).
- ScrollArea – Area de Desplazamiento o Enrollamiento - ScrollArea es el elemento eficaz para el movimiento en sentido vertical. Contiene contenido que acorta y proporciona características para exponer la compensación y el alcance del contenido. También proporciona el manejo de la entrada por defecto tales que el movimiento en sentido vertical se puede conducir programatically vía teclado o de la rueda del ratón.
- ScrollBar – Barra de Desplazamiento - La clase de ScrollBar.
- Selector - La clase base para los controles que seleccionan items entre de sus hijos.
- Slider - Deslizador - La clase del resbalador.
- Thumb - Pulgar - El control del pulgar permite la funcionalidad básica del movimiento de arrastre (drag-movement) para las barras de desplazamiento y las ventanas de maximización y minimización.

"IEnsureVisible" Seguro Visible - Es un interfaz del ejemplo asociado al namespace de System.Windows.Controls.Primitives. IEnsureVisible es implementado en una representación visual de desplaza/mueve (scroll/move) una representación visual del hijo en la vista.

La lista siguiente contiene las enumeraciones del ejemplo asociadas al namespace de System.Windows.Controls.Primitives.

- ArrowButtonStates – Estados de Botón Flecha –
- CloseModeType – Tipo de Modo Cerca - Describe cómo un popup debe comportarse a varios eventos del ratón.
- Part - Parte - La enumeración de la pieza se utiliza para indicar el uso semántico de los controles que conforman la barra de desplazamiento – enrollamiento.
- PartStates - Estados De la Pieza - Barra de Desplazamiento.
- PlacementType – Tipo de ubicación - Describe donde un popup se debe poner en la pantalla.
- SizeBoxStates – Estados de Tamaño de Caja –

Los documentos namespace 312 son una colección de elementos semánticos y de formato que se utilizan para crear los documentos ricamente y semánticamente ajustados al formato.

En una personalización, un "elemento" es una clase que se utiliza sobre todo conjuntamente con una jerarquía de los elementos (designados a un "árbol"). Estos elementos pueden ser interactivos (e.g., recibiendo una entrada de usuario vía teclado, ratón u otro dispositivo de entrada), pueden entregar imágenes u objetos, y pueden apoyar con el arreglo de otros elementos. Los elementos del ejemplo incluyen un elemento "bloque" que implementa un bloque genérico, un elemento del "cuerpo" que representa el contenido que incluye el cuerpo de una tabla, un elemento de la "célula" que contiene datos tabulares dentro de una tabla,

un elemento del "encabezado" que representa el contenido incluido en el encabezado de una tabla, y un elemento de "Salto de Página" que es utilizado para separar el contenido a través de múltiples páginas.

La lista siguiente contiene las clases de ejemplo expuestas por el namespace System.Windows.Documents.

- AdaptiveMetricsContext – Contexto de Metricas Adaptables - Proporciona el elemento de la raíz para los documentos adaptative-flow-formato (Formato de Flujo Adaptable). Una vez que un panel del hijo se encapsule en un elemento AdaptiveMetricsContext, el contenido del panel es procesado por el Motor de Métrica de Lectura (RME – reading Metrics Engine). El tamaño del panel del hijo se utiliza para calcular el número y el tamaño de cualquier columna así como los tamaños de fuente y la altura de línea óptimos.
- Block - Bloque – Implementa un elemento de bloque genérico que no induce el comportamiento suministrado pordefecto.
- BlockElement – Elemento de Bloque – Implementa una clase base para todos los elementos del bloque.
- Body - Cuerpo - Representa el contenido que compromete el cuerpo del elemento de tabla.
- Bold – Negrilla – Implementa un elemento negrilla derivado de en línea (Inline).
- BreakRecord – Registro de Rompimirnto o de Separación – Almacena la información necesaria para continuar formateando el

contenido paginado a través de los separadores de página. Herencia de esta clase para proporcionar la ayuda de la paginación. Esta es una clase abstracta.

- Cell - Célula - Las células contienen datos tabulares dentro de una tabla. Los elementos de la célula están contenidos dentro de un registro.
- CellCollection – Colección de Célula - Colección ordenada de células de tabla.
- Column - Columna - El elemento columna es utilizado para repartir los contenidos de un GridPanel (Panel de Grilla) o de una tabla.
- ColumnCollection – Colección de Columna – Una colección de columna (ColumnCollection) es una colección ordenada de columnas.
- ColumnResult – Resultado de Columna - Representa la información de vista-relacionada de columna.
- ContainerParagraphResult – Resultado de Párrafo de Contenedor - Proporciona el acceso a los parámetros calculados del arreglo para un párrafo objeto que contenga solamente otros objetos del párrafo.
- ContentPosition – Posición de Contenido - Representa la posición del contenido dentro de un párrafo. Hereda de esta clase para describir la posición del contenido asociado. Esto es una clase abstracta.

- Document – Documento - El propósito de la clase del documento es desemparejar el contenido de un documento del "chrome" de UI que le rodea. "Decoupling" – desemparejar significa que usted puede ser autor de un documento sin pensar a cerca de (y sin confiar) de su UI – Interfase de Usuario. La clase del documento que contiene el contenido del documento, típicamente un TextPanel – Panel de Texto ó un FixedPanel (Panel Reparado y sus hijos. Un árbol visual (por defecto, un Visor de Página - PageViewer) está asociado con este a través del control WPP que labra el mecanismo.

- DocumentPage – Página de Documento - Representa la información dispuesta para un control asociado a una página de un documento sjeto a paginación. Hereda de esta clase para implementar para describir la información dispuesta para estos controles. Esto es una clase abstracta.

- DocumentPageParagraphResult – Resultado de Párrafo de Documento de Página - Proporciona el acceso a los parámetros dispuestos calculados para los objetos afectados por al paginación.

- FindEngine – Motor de Búsqueda - Clase base para encontrar los algoritmos.

- FindEngineFactory - Fábrica de Motor de Búsqueda – fábrica de búsqueda de algoritmos.

- FixedPage – Página Fija - Proporciona el acceso a una sola página de contenido dentro de un documento dispuesto de formato fijo.

- Footer – Pie de Página - Representa el contenido que abarca el pie de página de un elemento tabla.

- Header - Encabezado - representa el contenido que abarca el encabezado de un elemento de tabla.

- Heading – Membrete - Título – Implementa un elemento de nivel de bloque (block-level) que suministra el texto como un membrete.

- HyphenationDictionary – Diccionario de Separación - Representa un diccionario con el fin de proporcionar la ayuda de la inscripción con separación dentro de las aplicaciones. Puede contener un diccionario en línea y una referencia a un diccionario externo. El diccionario en línea tiene prioridad más alta y será aplicado antes de las entradas en el diccionario externo.

- Hyphenator – Separador - Es el objeto de separación que mantiene referencia a los datos de la inscripción dentro de un Diccionario de Separación (HyphenationDictionary) y también realiza la inscripción con separador.

- Inline - En línea – Implementa un elemento en línea genérico que no induce ningún comportamiento suministrado por defecto.

- InlineElement – Elemento en línea – Implementa un elemento en línea genérico como clase base para los elementos en línea.

- Italia - Itálico – Implementa un elemento itálico derivado en línea.

- LineBreak – Rompimiento de Línea – Representa un elemento del margen que obliga un salto de línea.
- LineResult – Resultado de Línea - Proporciona el acceso a la información calculada de una línea del texto.
- List - Lista – Implementa un elemento de lista. Las listas son elementos del nivel de bloqueo (block-level) diseñados para ser formateados con los marcadores tales como viñetas o enumeración.
- ListElementItem – Item del Eelemento de Lista – Implementa un ListElementItem, que apoya marcadores tales como viñetas o enumeración.
- Note - Nota – Implementa un elemento, que es análago al elemento de la nota en el HTML.
- PageBreak – Salto de Página - Representa un elemento del margen usado para separar el contenido a través de varias páginas.
- PageDescriptor – Descriptor de Página – Implementa un PageDescriptor, que almacena la información necesaria para crear la muestra paginada.
- Paragraph - Párrafo – Implementa un eleemnto nivel de bloque utilizado para representar el texto en un párrafo. El comportamiento de representación es análogo a la del elemento del párrafo en el HTML.

- ParagraphResult – Resultado de Párrafo - Proporciona el acceso a los parámetros de la muestra calculada para un objeto del párrafo.
- Row - Fila - Define una fila dentro de un elemento panel de grilla (GridPanel) o elemento de tabla.
- RowCollection – Colección de Filas - Representa una colección pedida de filas.
- RowGroup – Grupo de Filas - Especifica la característica por defecto para un grupo de filas en una tabla o en un panel de grilla (GridPanel).
- Section - Sección - Implementa un elemento contenedor genérico. El comportamiento de la representación es analálogo al elemento div en HTML.
- SmallCaps – Cápsulas Pequeñas – Implementa un elemnto de cápsula pequeña en línea. SmallCaps son las formas tipográficas que representan una versión principal pequeño de las letras para el énfasis, como en un título.
- SubscriptSubíndice - Representa un elemento suscrito en línea. Los caracteres suscritos se escriben inmediatamente debajo, debajo y a la izquierda, o debajo y a la derecha de otros caracteres.
- Superscript - Exponente - Representa un elemento en la línea del exponente. Los caracteres del exponente son típicamente letras o

números y se muestran inmediatamente arriba, arriba y a la izquierda, o arriba y a la derecha de otros caracteres.

- Table - Tabla - La tabla se utiliza para mostrar datos complejos en forma tabular usando un lenguaje de mayor beneficio (e.g., "XAML").

- TextArray - Arreglo de Texto - Base API para el acceso y la manipulación del texto

- TextChangedEventArgs – Argumetnos de Evento de Texto Cambiado - Define los argumentos del evento enviados cuando se cambia un Arreglo de Texto (TextArray)

- TextElement – Elemento Texto - Proporciona las facilidades del Rango de Texto (TextRange) para el Arbol de Texto (TextTree). Es un TextRange inmutable, continuo con puntos finales fijos. Proporciona la entrada al elemento de contenido (ContentElement), ayuda del evento y del Foco. Proporciona la ayuda de la propiedad, del Objeto de Dependencia (DependencyObject)ContentElement.

- TextNavigator – Navegador de Texto - Este puede enumerar el contenido del texto. Implementa una posición de texto (TextPosition) móvil. Puede moverse por corrida del texto o ser posicionado en una localización conocida en el texto.

- TextParagraphResult – Resultado de Párrafo de Texto- Proporciona el acceso a los parámetros calculados dispuestos para el texto, incluyendo objetos y figuras flotantes.

- TextPosition – Posición del Texto - Este es un objeto que representa cierta posición en un arreglo de texto (TextArray). Es un objeto compacto que representa una posición en el texto y mantiene automáticamente la posición cuando el texto cambia. Las operaciones de la comparación son solamente aplicables a las posiciones dentro del mismo TextArray (el mismo contexto) La posición de texto (TextPosition) puede ser estática o móviles. La característica de (Es cambaible) IsChangeable dice la clase de posición.

- TextRange – Rango de Texto - Es una clase abstracta que provee la asociación genérica de cero o más subranges con características. La manipulación de Subrange es definida en clases derivadas.

- TextRangeMovable – Texto de Rango Movible - Es una clase abstracta para los rangos de texto (TextRanges) móviles. Agrega la capacidad de mover al comienzo y al final de los puntos basados en unidaes de texto (TextUnits).

- TextTreeChangedEventArgs – Argumentos de Evento Cambiado del Arbol del Arbol de Texto - Define las argumentos del evento enviadas cuando el TextArray es cambiado.

- TextTreeDumper – Depósito del Arbol de Texto - Es una clase de la prueba del árbol que es público debido a las ediciones de empaquetado

- TextTreeNavigator – Nvegador del Arbol de Texto – Este es un objeto que representa cierta posición movible en el árbol de texto (TextTree). Es una puesta en práctica específica del navegador de texto (TextNavigator) para el uso solamente en el aárbol de texto (TextTree).

- TextTreePosition – Posición del Arbol de Texto - Este es un objeto que representa cierta posición inmutable en un árbol de texto (TextTree). Es una puesta en práctica específica de la posición del texto (TextPosition) para el uso solamente en el TextTree.

- TextTreeRange – Rango del Arbol de Texto - Proporciona las facilidades del rango de texto (TextRange) para el árbol de texto (TextTree). Es un TextRange cambiable, continuo con puntos finales móviles.

- TextTreeRangeContentEnumerator – Enumerador sobre el objeto hijo directamente bajo un rango del texto del arbol.

- TextUnit – Unidad de Texto - Unidad extensible de la navegación del texto.

- TextUnits – Unidades de Texto - Unidades de texto comúnmente usadas para la posición del texto (TextPosition) y el rango del texto (TextRange).

- Typography - Tipografía - Proporciona el acceso a un sistema rico de características de tipografía tipo abierto (OpenType).

- UIElementParagraphResult – Resultado de Párrafo del Elemento de la Interfase de Usuario – Es el ParagraphResult para un párrafo que se compone enteramente de un elemento de interfase de usuario (UIElement). Utilizado para los flotadores, las figuras y el nivel de bloque incluido UIElements.

- Underline - Raya - Implementa un elemento de la raya derivado del elemento en línea (InlineElement).

La lista siguiente contiene interfaces de ejemplo asociadas con el namespace System.Windows.Documents.

- IDocumentContentHost – Alojamiento del Contenido del Idocument - Implementa esta interfase sobre un contenido huésped de modo que los hijos de ese anfitrión puedan notificar el anfitrión cuando el contenido está cambiando.

- IDocumentFormatter – Formato de Docmento – Implementa esta interfase sobre un elemento para proporcionar la ayuda para las características del documento tales como paginación.

- ITextDocumentResult – Resulatdo de Texto de Documento - Implementa esta interfase para mantener la información de la columna para un documento.

- ITextParagraphResult – Resulatdo del Párrafo de Itexto – Implementa este interfase para proveer texto e información de posicionamiento para los párrafos del texto.

La lista siguiente contiene las enumeraciones del ejemplo asociadas al namespace System.Windows.Documents.

- ElementEdge – Eje de Elemnto - Este identifica el eje de un objeto donde se localiza una posición de texto (TextPosition).

- FindAdvancedOptions – Búsqueda de Opciones de Avance - Las opciones de búsqueda avanzadas utilizadas por el algoritmo de búsqueda (FindAlgorithm) (inicialización de la búsqueda) y las clases de TextRangeMovable/TextSelection (ejecución simplificada de búsqueda).

- FindOptions – Búsqueda de Opciones – Es la búsqueda simplificada de opciones utilizada por los métodos de búsqueda de caja de texto (TextBox.Find).

- LogicalDirection – Dirección Lógica - Define una dirección lógica para el movimiento en texto. También se utiliza para determinar dónde una posición de texto (TextPosition) se moverá cuando el contenido es insertado en la posición de texto (TextPosition).

- TextArrayRunType – Tipos de Recorrido del Arreglo de Texto - Este identifica el recorrido donde se localiza un TextPosition, tomando en cuenta la dirección lógica (LogicalDiretion)

- TextChangeOptions – Opciones de Cambio de Texto – Son los posibles cambios de texto para un texto que puede cambiar (CanChangeText).

- TextMoveOptions – Opciones de Movimiento de Texto – Controla el movimiento de un navegador de texto (TextNavigator) especificando condiciones de parada de la navegación.

La lista siguiente contiene ejemplos de delegados asociados con el namespace documentos del sistema Windows (System.Windows.Documents).

- ObjectCloneDelegate – Delegado de Objeto Clon - Método de retorno de llamada para proporcionar un clon o una copia de un Objeto de Dependencia (DependencyObject) cuando una porción de un arreglo de texto (TextArray) se está copiando o se está moviendo.

- TextChangedEventHandler – Manejador de Evento de RTExto Cambiado - El TextChangedEventHandler delegado es llamado con argumentos de evento cambiado de texto (TextChangedEventArgs) cada vez que el contenido es adicionado o removido del árbol de texto (TextTree).

Un namespace 314 de formas es una colección de elementos gráficos de vector usados para crear imágenes y objetos. El uso de elementos gráficos de vector permiten que los elementos sean fácilmente redimensionados para ajustarse a los requisitos de una interfase particular o dispositivo de despliegue. La lista siguiente contiene ejemplos de clases expuestas por el namespace formas del sistema Windows (System.Windows.Shapes).

- Ellipse - Elipse - Dibuja una elipse.
- Grabados - Glyphs - Representa una forma de grabado en un lenguaje enriquecido como "XAML". Los grabados se utilizan para representar fuentes.
- Line - Línea - Dibuja una línea recta entre dos puntos.
- Path – Ruta ó Trayectoria - Dibuja una serie de líneas y de curvas conectadas.
- Polygon - Polígono - Dibuja un polígono (una serie conectada de líneas que forman una figura cerrada).
- Polyline - Dibuja una serie de líneas rectas conectadas.
- Rectangle - Rectángulo - dibuja un rectángulo.
- Shape - Forma - Una clase abstracta que proporciona la funcionalidad base para elementos de forma, tales como elipse, polígono y rectángulo.

Los espacios de nombre (namespaces) (System.Windows.Controls, System.Windows.Documents y System.Windows.Shapes proporcionan un sistema integrado para el desarrollo de aplicaciones y de componentes relacionados. Este sistema integrado proporciona un modelo de programación común para los tres namespaces, de tal modo que simplifica el desarrollo de los programas de aplicación. Esta interoperabilidad entre los tres namespaces permite que los desarrolladores aprendan una sola arquitectura de programación que está aplicada a cualesquiera de las características proporcionadas por tres namespaces. Por ejemplo, un lenguaje común enriquecido es utilizado a través de los tres namespaces. Este lenguaje común enriquecido proporciona para el mapeo de las tres clases y las características especificadas en XML enriquecido en un arbol de instancia de objetos.

Además, un modelo de programanción y servicios consistentes son utilizados a través de los tres namespaces. Por ejemplo, un sistema de evento constante es utilizado para iniciar y procesar varios eventos. Un sistema de característica común es utilizado para construir varias características, para atar datos a una característica, o para animar una característica, sin importar si la característica está asociada con "controles" ("controls"), documentos ("documents"), o formas ("forms") de namespace. Adicionalmente, los mismos paradigmas de entrada y manejo de lo dispuesto es comun a través de los tres namespaces. Por ejemplo, varios controles del namespace System.Windows.Controls se

pueden jerarquizar en el centro del contenido de un documento definido usando el namespace de System.Windows.Documents.

Los archivos de fuente del ejemplo incluirán un sistema de ventanas y panels (también referidos como "paginas") que son definidas declarativamente usando controles ("controls"), documentos y formas "documents" y "shapes". La lógica de interacción también se proporciona para las ventanas y los paneles. La lógica de la interacción identifica el programa de código que es ejecutado en respuesta a una acción particular del usuario o en respuesta a ocurrencia de un evento o de una actividad. La lógica de interacción se define, por ejemplo, usando un lenguaje tiempo de pasada común (CLR). Un CLR es un ambiente de tiempo de ejecución que maneja la ejecución del código del programa (e.g., código del programa de NET) y proporciona varios servicios tales como servicios relacionados de seguridad (security-related) y servicios relacionados de memoria (memory-related). Los lenguajes del ejemplo CLR incluyen C# y Visual Basic. Los archivos fuente también incluyen otros archivos independientes (stand-alone) del lenguaje de programación, por ejemplo C# o archivos Visual Basic.

Aunque esta discusión se refiere la integración de los namespaces de controles "Controls", de documentos "Documents" y de formas "Shapes", esta integración se puede aplicar a cualquiera o a todos los namespaces y namespaces secundarios (sub-namespaces) discutidos aquí.

Un namespace 316 de datos incluye las clases e interfaces utilizados para enlàzar las características de los elementos a las fuentes de datos, a las clases de fuente de datos, y a las implementaciones específicas de datos (data-especic), de colecciones de datos y de las vistas. Estas clases e interfaces también se utilizan para manejar excepciones en la entrada de datos y para permitir la creación del tiempo de ejecución de una interfase de usuario basado en la información de varias fuentes de datos. Los datos se pueden desplegar en forma textual o se pueden utilizar para cambiar el formato de la pantalla, tal como exhibir cantidades del dólar en rojo si son negativos. Las clases del ejemplo incluyen una clase enlace "bind" que represente un objeto obligatorio de declaración que maneje enlaces entre una característica dinámica de la interfase de usuario y los datos fuente, y una clase "XmlDataSource" que sirva como fuente de datos para los datos que atan el contenido de los nodos XML.

Un namespace 318 de medios proporciona varias clases de medios. Los desarrolladores de aplicación así como los desarrolladores de componentes pueden utilizar estas clases para desarrollar varias funcionalidades de la presentación. Las clases del ejemplo en namespace 318 de los medios incluyen una clase de efecto de imagen "ImageEffect" que permite ciertos efectos de imagen (e.g., definición y escala de grises – blur and grayscale), y una clase cepillo "brush" que proporciona un mecanismo para llenar un área usando colores sólidos, gradientes, imágenes, vídeo, y similares.

El namespace 318 de medios incluye un namespace secundario sub-namespace System.Windows.Media.Animation que incluye los servicios que permiten a un desarrollador animar características y coordinar un conjunto de animaciones con un conjunto de tiempos de referencia (timelines). Una animación es un objeto que cambia un valor durante un periodo de tiempo. Los efectos de animación incluyen el movimiento de un objeto en la pantalla, y el cambio del tamaño, la forma, o el color de un objeto. Las clases múltiples de animación se proporcionan a varios efectos de animación de la implementación. Los efectos pueden ser alcanzados asociando una animación al valor de característica de un elemento. Por ejemplo, para crear un rectángulo que se descoloriza dentro y fuera de la vista, una o más animaciones se asocian a la característica de la opacidad del rectángulo.

El namespace 318 de medios también incluye un secundario-namespace System.Windows.Media.TextFormatting que proporcione varios servicios de texto. Por ejemplo, un motor de texto "TextFormatter" proporciona los servicios para separar líneas de texto y ajustar al formato el texto presentado en una pantalla. Formateador de texto ("TextFormatter") es capaz de manejar diversos formatos de carácter de texto y estilos de párrafo así como la manipulación de la propuesta internacional del texto.

Un namespace 320 de diseño proporciona las clases que permiten la edición de formas y de texto, compartir datos que se ajustan al formato y al proceso de cruce (cross-process). Estas clases proporcionan un área

de trabajo extensible para la edición de documentos, aplicacions, y otro contenido. Un namespace 322 de entrada incluye un administrador de entrada que coordina las entradas recibidas por el sistema. El namespace 322 de entrada también incluye las clases que ayudan a manejar y proporcionan el control para diversos dispositivos de entrada, tales como un teclado o un ratón.

Un namespace 324 de navegación proporciona un conjunto de clases y de servicios que permiten la construcción de aplicacions con paradigmas de navegación, tales como una aplicación de browser. Estas clases y servicios permiten el desarrollo de aplicaciones con experiencias de navegación personalizadas. Por ejemplo, cuando se compra un producto o un servicio de comercio en línea, tecleando un botón trasero ("back") hace que la aplicación se despliegue en una página diferente y pregunte al usuario si desea cancelar o cambiar su orden. En otro ejemplo, al activar el botón de actualización ("refresh") causa ha ce que una aplicación recupere nuevos datos en vez primero de recargar la aplicación seguida por la recuperación de los nuevos datos. El namespace 324 de navegación también incluye funciones de página que proporcionan un mecanismo para generar una jerarquía de preguntas que son presentadas al usuario.

Un namespace 326 de automatización proporciona un conjunto de clases que apoyan la accesibilidad y la automatización de la interfase de usuario.

Un namespace 328 de serialización proporciona un programa de análisis que puede cargar o guardar una jerarquía de objetos (e.g., elementos) desde o hacia un archivo XML o un archivo con una representación binaria. Este proceso también fija características asociadas con los objetos y asocia manejadores de evento.

Un namespace 330 de interoperabilidad proporciona un conjunto de clases que permiten interoperabilidad con otros sistemas operativos o plataformas de cómputo.

Un namespace 332 de interoperabilidad de fromas (forms.interop) proporciona un elemento que permite a una aplicación recibir una operación de control de forma.

Otro namespace, System.IO.CompoundFile (no mostrado en la fig. 3) proporciona servicios para utilizar un archivo compuesto en el cual se almacenen los archivos distribuibles de varios documentos. Estos servicios permiten la encrpción y la compresión de contenido. Los servicios también apoyan el almacenamiento de múltiples entregas del mismo contenido, tales como un documento que puede cambiar varias veces (re-flowable) y un documento de formato fijo (fixed-format)

## AMBIENTE Y SISTEMA DE COMPUTO EJEMPLAR

La fig. 4 ilustra un ejemplo de un ambiente que computa conveniente 400 dentro de 1 cual el marco de programación 132 pueda ser implementado (completamente o parcialmente). El ambiente de

cómputo 400 se puede utilizar en las arquitecturas de cómputoy de red descritas aquí.

El ambiente de cómputo ejemplar 400 es solamente un ejemplo de un ambiente de cómputo y no se ha intentado sugerir ninguna limitación en cuanto al alcance del uso o de la funcionalidad de las arquitecturas de cómputo y de red. Nadie debería interprestar el ambiente 400 de cómputo como si tuviera alguna dependencia o requisito relacionado a cualquier combinación de componentes ilustrados en el ejemplo de ambiente de cómputo 400.

El marco de trabajo 132 se puede implementar con numerosos otros fines generales o ambientes o configuraciones especiales del sistema de cálculo del propósito. Ejemplos bien conocidos de los sistemas de cómputo, y/o configuraciones que pueden ser ajustables para el uso incluyen, pero no se limitan a, ordenadores personales, servidores, sistemas de multiprocesamiento, sistemas basados en microprocesador, PCs de la red, minicomputadoras, ordenadores centrales (mainframes), ambientes de cómputo distribuidos que incluyen cualesquiera de los sistemas o dispositivos de los dispositivos, etcétera. Las versiones del acuerdo o del subconjunto del marco se pueden también implementar en los clientes de recursos limitados, tales como teléfonos celulares, ayudantes digitales personales (PDAs), computadoras de mano, u otros dispositivos de communication/computing.

El marco 132 se puede describir en el contexto general de instrucciones de computadora ejecutables, tales como módulos del

programa, siendo ejecutado por uno o más computadores u otros dispositivos. Generalmente, los módulos del programa incluyen las rutinas, los programas, los objetos, los componentes, las estructuras de datos, etc. que realizan tareas particulares o implementan los tipos de datos abstractos particulares. El marco 132 se puede también practicar en los ambientes de cómputo distribuido donde las tareas son realizadas por los dispositivos de proceso remoto que se ligan a través de una red de comunicaciones. En un ambiente de cómputo distribuido (Distributed Computing Environment), los módulos del programa se pueden situar en medios de almacenamiento de cómputo locales y remotos incluyendo los dispositivos de almacenamiento de memoria.

El ambiente de cómputo 400 incluye un dispositivo de cómputo de propósito general (general-purpose) en la forma de una computadora 402. Los componentes de la computadora 402 pueden incluir, pro no se limitan a, unos o más procesadores o unidades de proceso 404, un sistema de memoria 406, y un sistema de bus 408 que empareja los componentes de varios sistemas incluyendo el procesador 404 hasta el sistema de memoria 406.

El sistema de bus 408 representa uno o más de los varios tipos posibles de estructuras de bus, incluyendo un bus de memoria o controlador de memoria, un bus periférico, un puerto acelerador de gráficos, y un procesador o bus local usando cualquiera de una variedad de arquitecturas de bus. A modo de ejemplo, tales arquitecturas pueden incluir un bus estándar de la arquitectura de la industria (ISA-Industry

Standard Architecture), un bus de arquitectura de micro canal (MCA-Micro Channel Architecture), un bue extendido ISA (EISA-Enhanced ISA), un bus local (VESA-Video Electronics Standards Association) – Asociación de Estándares Electrónicos, y un componente de interconección periférico bus (PCI-Peripheral Component Interconects) también conocido como Bus Mezzanine.

La computadora 402 incluye típicamente una variedad de medios legibles por computador. Tales medios pueden ser cualquier medio disponible que es accesible por la computadora 402 e incluye ambos medios volátiles y permanentes (non-volatile), removibles y no removibles (non-removable).

La memoria del sistema 406 incluye medios legibles por computador en forma de memoria volátil, tal como memoria de acceso aleatorio (RAM-Random Access Memory) 410, y/o memoria permanente, tal como la memoria de sólo lectura (ROM-Read Only Memory) 412. Un sistema básico de la entrada-salida (BIOS) 414, que contiene las rutinas básicas que ayudan a transferir la información entre los elementos dentro de la computadora 402, por ejemplo durante el inicio (start-up), se almacena en ROM 412. La RAM 410 contiene típicamente los datos y/o los módulos de programa que son inmediatamente accesibles y/o actualmente operados por la unidad de proceso 404.

La computadora 402 puede también incluir otro removible/fijo (removable/non-removable), medio de almacenamiento volátil/permanente (volatile/non-volatile). A modo de ejemplo, la fig. 4 ilustra una unidad de disco duro 416 de lectura y escritura para un medio magnético fijo, permanente (no mostrados), una unidad de disco magnético 418 de lectura y escritura para un disco magnético permanente, removible 420 (e.g., "un diskette"), y una unidad de disco óptico 422 para leer desde y/o escribir a un disco óptico removible, permanente 424 tales como un CD-ROM, a DVD-ROM, u otro medio óptico. La unidad de disco duro 416, la unidad de disco magnético 418, y la unidad de disco óptico 422 están cada una conectadas al sistema de bus 408 por uno o más interfaces de medios de datos 426. Alternativamente, la unidad de disco duro 416, la unidad de disco magnético 418, y la unidad de disco óptico 422 se pueden conectar con el bus 408 del sistema por una o más interfaces (no mostrados).

Las unidades de disco y sus medios legibles por computador asociados proporcionan el almacenamiento permanente de instrucciones legibles por computador, de estructuras de datos, módulos de programa, y de otros datos para la computadora 402. Aunque el ejemplo ilustra un disco duro 416, un disco magnético removible 420, y un disco óptico removible 424, debe ser apreciado que otros tipos de medios legibles por computador puedan almacenar los datos que son accesibles por una computadora, tal como cassettes magnéticos u otros dispositivos de almacenaje magnéticos, tarjetas de memoria de destello (memoria

flash), CD-ROM, los discos versátiles digitales (DVD) u otro almacenamiento óptico, las memorias de acceso aleatorio (RAM), memorias de solo lectura (ROM), la memoria eléctricamente borrable programable memoria de solo lectura (EEPROM), y similares, se pueden también utilizar para implementar el ambiente y el sistema de cómputol del ejemplo.

Cualquier número de los módulos del programa se puede almacenar en el disco duro 416, el disco magnético 420, el disco óptico 424, ROM 412, y/o RAM 410, incluyendo a modo de ejemplo, un sistema operativo 426, uno o más programas de aplicación 428, otros módulos de programa 430, y los datos del programa 432. Cada uno del sistema operativo 426, de unos o más programas de aplicación 428, de otros módulos de programa 430, y los datos de programa 432 (o alguna combinación de ellos) puede incluir elementos del sistema de programación 132.

Un usuario puede incorporar comandos y la información en la computadora 402 vía los dispositivos de entrada tales como un teclado 434 y un dispositivo de punteo 436 (e.g., un ratón "mouse"). Otros dispositivos de entrada 438 (no mostrados específicamente) pueden incluir un micrófono, una palanca de mando (joystick), una tabla de juego, un plato basado en los satélites, un puerto serial, un digitalizador, y/o similares. Éstos y otros dispositivos de entrada están conectados con la unidad de proceso 404 vía las interfaces 440 de entrada-salida que se juntan al bus 408 del sistema, pero se pueden conectar por otras

estructuras de interfaz y del bus, tales como un puerto paralelo, puerto de juego, o un bus serial universal (USB).

Un monitor 442 u otro dispositivo de pantalla se puede también conectar con el bus 408 del sistema vía una interfase, tal como un adaptador de video 444. Además del monitor 442, otros dispositivos periféricos de salida pueden incluir componentes tales como parlantes (no mostrados) y una impresora 446 la cuál se puede conectar con la computadora 402 vía las interfaces 440 de entrada-salida.

La computadora 402 puede funcionar en un ambiente conectado de red usando conexiones lógicas a una o más computadoras remotas, tales como un dispositivo de cómputo remoto 448. A modo de ejemplo, el dispositivo de cómputo remoto 448 puede ser el ordenador personal, la computadora portable, el servidor, el enrutador, la computadora de red, el dispositivo par (peer) u otro nodo de red común, etcétera. El dispositivo de cómputo remoto 448 se ilustra como una computadora portable que puede incluir muchos o todos los elementos y características descritos aquí concernientes a la computadora 402.

Las conexiones lógicas entre la computadora 402 y la computadora remota 448 se representan como una red de área local (LAN) 450 y red de área amplia general (WAN) 452. Tales ambientes de una red son comúnmente establecidos en oficinas, redes de ordenadores de empresas grandes, intranets (Internet Privadas), y el Internet.

Cuando se implementa en un ambiente de red del LAN, la computadora 402 es conectada con una red local 450 vía una interfase o un adaptador 454 de red. Cuando es implementada en un ambiente WAN red, la computadora 402 incluye típicamente un módem 456 u otro medio para establecer comunicaciones sobre la red amplia 452. El módem 456, que puede ser interno o externo a la computadora 402, se puede conectar con el bus 408 del sistema vía los mecanismos apropiados de las interfaces 440 u otra de la entrada-salida. Debe ser apreciado que las conexiones de red ilustradas son de ejemplo y que otros medios para establecer el vínculo (link(s)) de comunicación entre las computadoras 402 y 448 pueden ser empleados.

En un ambiente conectado de red, tal como es ilustrado con el ambiente de cómputo 400, los módulos del programa representados relacionados a la computadora 402, o las partes de este, pueden ser almacenados en un dispositivo de almacenamiento de memoria remoto. A modo de ejemplo, los programas de aplicación remotos 458 residen en un dispositivo de memoria de la computadora remota 448. Para los propósitos de la ilustración, los programas de aplicación y otros componentes dxe programa ejecutables tales como el sistema operativo se ilustran aquí como bloques discretos, aunque se reconoce que tales programas y componentes residen varias veces en diversos componentes de almacenamiento del dispositivo de cómputo 402, y son ejecutados por el procesador(es) de datos de la computadora.

Una implementación del sistema 132, y particularmente, del API 142 o de las llamadas hechas al API 142, se puede almacenar o transmitir a través de una cierta forma de medios legibles por computador. Los medios legibles por computador pueden ser cualquier medio disponible que se pueda acceder por una computadora. A modo de ejemplo, y sin limitación, los medios legibles por computador pueden abarcar "medios de almacenamiento de computadora" y "medios de comunicaciones." "los medios almacenamiento de computadora " incluyen los medios volátiles y permanentes, removibles y fijos implementados en cualquier método o tecnología para el almacenamiento de la información tal como instrucciones legibles por computador, estructuras de datos, módulos de programa, u otros datos. Los medios de almacenamiento de la computadors incluyen, pero no se limitan a, RAM, ROM, EEPROM, memoria flash u otra tecnología de memoria, CD-ROM, discos versátiles digitales (DVD) u otro almacenaje óptico. los cassettes magnéticos, cinta magnética, almacenamiento en discos magnético u otros dispositivos de almaenamiento magnético, o cualquier otro medio que pueda ser utilizado para almacenar la información deseada y que se pueda acceder por una computadora.

Los "medios de comunicación" incorporan típicamente las instrucciones legibles por computador, estructuras de datos, módulos de programa, u otros datos en señales de datos modulados, por ejemplo onda de portador u otro mecanismo de transporte. Los medios de comunicación también incluyen cualquier medio de entrega de la

información. El término "modulated data signal" - señal modulada de los datos significa una señal que tiene uno o más del conjunto de sus características fijas o cambiadas en tal manera para codificar la información en la señal. A modo de ejemplo, y sin limitación, los medios de comunicación incluyen medios atados cableados tales como una red unida con cableado o una conexión cableada directa (wired-direct) con alambre, y medios inalámbricos tales como acústica, RF, infrarrojo, y otros medios inalámbricos. Las combinaciones de cualquiera de las anteriores descritas también se incluyen dentro del alcance de medios legibles por computador.

Alternativamente, las partes del sistemas se pueden poner en ejecución en hardware o en una combinación de hardware, software, y/o de los soportes lógico inalterables (firmware). Por ejemplo, uno o más circuitos integrados específicos de aplicación (ASICs) o los dispositivos de lógica programables (PLDs) se podrían diseñar o programar para implementar una o más partes del sistema.

Una interfase de programación (o simplemente, interfase) se puede ver como cualquier mecanismo, proceso, protocolo para permitir uno o más segmento(s) de código para comunicarse con o para tener acceso a la funcionalidad proporcionada por uno o más de los otros segmento(s) de código. Alternativamente, una interfase de programación se puede ver como uno o más mecanismos, métodos, llamadas de funciones, módulos, objetos, etc. de un componente de un sistema capaz de

acoplarse comunicativamente a uno o más mecanismos, métodos, llamadas de funciones, módulos, objetos, etc. de otros componentes. El término "segmento de código" en la oración precedente se piensa para incluir una o más instrucciones o líneas de código, e incluye, e.g., los módulos de código, objetos, subprogramas, funciones, etcétera, sin importar la terminología aplicada o si los segmentos de código están compilados por separado, o si los segmentos de código están proporcionados como fuente, intermedio, o código de objeto, si los segmentos de código son utilizados en un sistema o un proceso de tiempo de ejecución, o si están situados en la misma o en diferentes máquinas o distribuidos a través de múltiples máquinas, o si la funcionalidad representada por los segmentos del código está implementada completamente en el software, ompletamente en hardware, o en una combinación de hardware y de software.

Prácticamente, una interfaz de programación se puede ver genéricamente, según lo mostrado en fig. 5 o fig. 6. La fig. 5 ilustra una interfaz de Interface1 como un conducto a través del cual los segmentos de código primero y segundo se comunican. La fig. 6 ilustra una interfase como los objetos de interfase que abarcan I1 e I2 (que pueden o no ser parte de los primeros y segundos segmentos de código), que habilitan el primer y segundo segmentos de código de un sistema para comunicarse vía el medio M. En la vista de fig. 6, uno puede considerar los objetos de interfase I1 e I2 como las interfaces separados del mismo sistema y pueden también considerar que los objetos I1 e I2 más el

medio M abarcan al interfase. Aunque las Figs.. 5 y 6 muestran el flujo bidireccional y las interfases en cada lado del flujo, ciertas implementaciones pueden solamente tener o no flujo de información en una dirección (o ningún flujo de información según lo descrito abajo) o pueden solamente tener un objeto de interfase en un lado. A modo de ejemplo, y sin limitación, los términos tales como programación de aplicación o interfase de programa (API), el punto de entrada, el método, la función, el subprograma, el llamado a procedimiento remoto (Remote Procedure Call), y la interfase del modelo del componente objeto (COM), se abarcan dentro de la definición de interfase de programación.

Los aspectos de tal interfase de programación pueden incluir el método por el que el primer segmento de código transmita la información (donde la "información" se utiliza en su sentido más amplio e incluye datos, comandos, peticiones, etc.) al segundo segmento de código; el método por el que el segundo segmento de código recibe la información; y la estructura, la secuencia, el sintaxis, la organización, el esquema, la sincronización y el contenido de la información. En este respeto, el medio de transporte subyacente en sí mismo puede ser poco importante a la operación de la interfase, si el medio es cableado o inalámbrico, o a una combinación de ambos, mientras la información se transporta de la manera definida por la interfase. En ciertas situaciones, la información no se puede pasar en una o ambas direcciones en el sentido convencional, mientras que la transferencia de información

puede ser vía otro mecanismo (e.g. información puesta en un almacenador intermedio (buffer), un archivo, etc. separado del flujo de información entre los segmentos de código) o no existente, como cuando un segmento de código tiene acceso simplemente a la funcionalidad realizada por un segundo segmento de código. Cualesquiera o todos estos aspectos pueden ser importantes en una situación dada, e.g., dependiendo de si los segmentos de código son parte de un sistema en una configuración libremente o firmemente acoplada, y así que esta lista se debería considerar ilustrativa y no limitante.

Esta noción de una interfase de programación es conocida para los expertos en la materia y es clara la descripción detallada precedente de la invención. Hay, sin embargo, otras maneras de implementar una interfase de programación, y, a menos que expresamente esté excluida, éstos se piensan también para ser abarcados por las demandas dispuestas al final de esta especificación. Otras maneras pueden aparecer y ser más sofisticadas o complejas que la vista simplista de las figs. 5 y 6, pero ellos no obstante realizan una función similar para lograr el mismo resultado total. Ahora describiremos brevemente algunas implementaciones alternativas ilustrativas de una interfase de programación.

## A. FACTORIZANDO

Una comunicación de un segmento de código a otro puede ser lograda indirectamente rompiendo la comunicación en múltiples

comunicaciones discretas. Esto se representa esquemáticamente en los figs. 7 y 8. Según lo mostrado, algunas interfaces se pueden describir en términos de sistemas divisibles de funcionalidad. Así, la funcionalidad de la interfase de las figs. 5 y 6 se puede descomponer en factores para alcanzar el mismo resultado, exactamente como uno puede obtener matemáticamente 24, ó 2 x 2 x 3 x 2. Por consiguiente, según lo ilustrado en fig. 7, la función proporcionada por la interfase Interface1 se puede subdividir para convertir las comunicaciones de la interfase en múltiples interfases Interface1A, interfase 1B, interfase 1C, etc. mientras que alcanza el mismo resultado. Según lo ilustrado en fig. 8, la función proporcionada por el interfaz I1 se puede subdividir en las múltiples interfases I1a, I1b, I1c, etc. mientras que alcanza el mismo resultado. De forma similar, la interfase I2 del segundo segmento de código que recibe la información del primer segmento de código se puede descomponer en factores en las múltiples interfases I2a, I2b, I2c, etc. Al descomponer en factores, el número de las interfases incluidas con el 1er segmento de código no es necesario que coincida el número de lss interfases incluidas con el 2do segmento de código. En cualquiera de las cajas de las Figs. 7 y 8, el espíritu de funcionalidad de las interfases Interface1 e I1 siguen siendo iguales que con las Figs. 5 y 6, respectivamente. La factorización de interfases puede también seguir características asociativas, conmutativas, y otras matemáticas tales que al descomponer en factores puede ser difícil de reconocer. Por ejemplo, el ordenar operaciones puede ser poco importante, y

por lo tanto, una función realizada por una interfase se puede realizar bien por adelantado del alcance de la interfase, por otra parte del código o de la interfase, o realizada por un componente separado del sistema. Por otra parte, una habilidad común en las artes de programación puede apreciar que hay una variedad de maneras diversas de hacer llamadas de función que alcancen el mismo resultado.

## B. REDEFINICIÓN

En algunos casos, puede ser posible ignorar, agregar o redefinir ciertos aspectos (e.g., parámetros) de una interfase de programación mientras que todavía se logra el resultado previsto. Esto se ilustra en las Figs. 9 y 10. Por ejemplo, asuma que la interfase Interface1 de la fig. 5 incluye una llamada de función *Square(input, precisión, output)*, una llamada que incluye tres parámetros, entrada, precisión y salida, y que se pasa del 1er segmento de código al 2do segmento de código, si el parámetro medio de *precision* no depende de un panorama dado, según lo demostrado en la fig. 9, podría del mismo modo ser ignorado o aún sustituido por un un parámetro sin valor (*meaningless)* (en esta situación). Uno puede también agregar un parámetro adicional *additional* sin preocupación. En cualquier evento, la funcionalidad del cuadrado puede ser alcanzada, siempre y cuando se retorne la salida después de que la entrada sea ajustada por el segundo

segmento de código. La *precisión* muy bien puede ser un parámetro significativo para alguna corriente u otra parte del sistema de cómputo; sin embargo, una vez que se reconozca que la *precisión* no es necesaria para el propósito estrecho de calcular el cuadrado, puede ser substituida o hacer caso omiso. Por ejemplo, en vez de pasar un valor válido de *precisión*, un valor sin setido tal como una fecha de nacimiento podría ser pasado sin al contrario afectar el resultado. Similarmente, según lo mostrado en la fig. 10, la interfase I1 es substituida por la interfase I1', redefinida para ignorar o para agregar parámetros a la interfase. La interfase I2 se puede redefinir similarmente como interfaz I2', redefinido para ignorar parámetros innecesarios, o parámetros que se pueden procesar en cualquier parte. El punto aquí está en algunos casos en que una interfase de programación puede incluir aspectos, tal como parámetros, que no son necesarios para un cierto propósito, y así pueden ser ignorados o ser redefinidos, o ser procesados en otra parte para otros propósitos.

## C. CODIFICACIÓN EN LÍNEA

Esta puede también ser factible para combinar alguna o toda la funcionalidad de dos módulos de código separados tales que la "interfase" entre ellos cambioa la forma. Por ejemplo, la funcionalidad de las Figs. 5 y 6 se pueden convertir a la funcionalidad de las Figs. 11 y 12, respectivamente. En fig. 11, los anteriores 1ros y 2dos segmentos de código de la fig. 5 se combinan

en un módulo que los contiene a ambos. En este caso, los segmentos de código pueden todavía comunicarse con cada uno pero la interfase se puede adaptar a una forma que sea más conveniente a el solo módulo. Así, por ejemplo, las declaraciones formales, de llamada (Call) y de retorno (Return) pueden no ser más necesarias, pero el proceso similar o la(s) respuesta(s) conforme a la interfaz Interface1 puede en efecto permanecer. Similarmente, mostrado en la fig. 12, parte (o toda) la interfaz I2 de la fig. 6 se puede escribir en línea en la interfase I1 para formar la interfase I1". Según lo ilustrado, la interfase I2 se divide en I2a e I2b, y la parte I2a de la interfase se ha codificado en línea con la interfase I1 para formar la interfase I1". Para un ejemplo concreto, considere que la interfase I1 de la fig. 6 realiza un llamado de la de función cuadrado (entrada, salida), que es recibida por el interfase I2, que después de procesar el valor pasó con la entrada (a elevarla al cuadrado) por el segundo segmento de código, devuelve el resultado al cuadrado con salida. En tal caso, el proceso realizado por el segundo segmento de código (que eleva al cuadrado la entrada) se puede realizar por el primer segmento de código sin una llamada a la interfase.

## D. DIVORCIE

Una comunicación de un segmento de código a otro puede ser logrado indirectamente rompiendo la comunicación en múltiples comunicaciones discretas. Esto se representa esquemáticamente en las Figs. 13 y 14. Según lo mostrado en la fig. 13, una o más

pieza(s) del middleware (Divorcio de Interfase(s), puesto que dividen funcionalidad y/o funciones de la interfase original) se proporciona para convertir las comunicaciones sobre la primer interfase, Interface1, para que ellas conformen una interfase diferente, en este caso las interfases Interfase2A, la Interfase2B y la Interfase2C. Esto podría ser realizado, ej., donde hay una base instalada de aplicaciones diseñados para comunicarse con, por ejemplo, un sistema operativo de acuerdo con un protocolo de Interfase1, pero entonces el sistema operativo se cambia para utilizar una interfase diferente, en las interfases de este caso Interface2A, Interface2B e Interface2C. El punto es que la interfase original usada por el 2do segmento de código es cambiado y ya no es compatible con la interfase usada por el 1er segmento de código, y así que un intermediario es utilizado para hacer las viejas y nuevas interfases compatibles. Similarmente, según lo mostrado en ls fig. 14, un tercer segmento de código se puede introducir con la interfase de separación DI1 para recibir las comunicaciones de la interfase I1 y con el interfase de separación DI2 para transmitir la funcionalidad de la interfase a, por ejemplo, las interfases I2a e I2b, rediseñadas para trabajar con DI2, pero para proporcionar el mismo resultado funcional. Similarmente, DI1 y DI2 pueden trabajar juntas para traducir la funcionalidad de las interfases I1 e I2 de la fig. 6 a un nuevo sistema operativo, mientras que proporciona el mismo o el resultado funcional similar.

## E. REESCRITURA

Otra variante posible debe reescribir dinámicamente el código para substituir la funcionalidad de la interfase por algo más pero que alcance el mismo resultado total. Por ejemplo, puede haber un sistema en el cual un segmento de código presentó un lenguaje intermedio (ej. Microsoft IL, Java ByteCode, etc.) es proporcionado a un compilador o a un interpretador Just-in-Time (JIT) - Justo A Tiempo en un ambiente de ejecución (tal como se proporcionó por el sistema .Net, el ambiente runtime de Java, o el otro tipo runtime similar ambientes). El compilador JIT puede ser escrio para convertir dinámicamente las comunicaciones del 1er segmento de código al 2do segmento de código, es decir, para que ellas se conformen en interfases diferentes como puede ser requerido por el 2do segmento de código (el 2do segmento de código original o diferente). Esto se representa en las figs. 15 y 16. Como puede ser visto en fig. 15, este acercamiento es similar al panorama de divorcio descrito arriba. Eso podría ser hecho, ej., donde una base instalada de aplicaciones es diseñada para comunicarse con un sistema operativo de acuerdo con un protocolo de la interfase 1, pero luego el sistema operativo es cambiado para utilizar una interfase diferente. El compilador JIT se puede utilizar para conformar las comunicaciones en marcha desde las plicaciones de base instaladas con la nueva interfase del sistema operativo. Según lo representado en la fig. 16, este acercamiento de

interfases de reescritura dinámicamente puede ser aplicado a un factor dinámico, o altera también de otra manera la interfase.

También es observado que los escenarios arriba descritos para alcanzar el mismo o el resultado similar como una interfase vía personalizaciones alternativas se pueden también combinar de varias maneras, en serie y/o en paralelo, o con otro código que interviene. Así, las personalizaciones alternativas presentadas arriba no son mutuamente exclusiva y se pueden mezclar, emparejar y combinar para producir el mismo o los panoramas equivalentes a los panoramas genéricos presentados en las Figs. 5 y 6. También se observa que, como con la mayoría de las construcciones de programación, hay otras maneras similares de alcanzar el mismo o la funcionalidad similar de una interfaseque no se puede describir aquí, pero no obstante son representadas por el espíritu y el alcance de la invención, es decir, él está observado que por lo menos en parte la funcionalidad representada por, y los resultados ventajosos permitidos, una interfase que es la base del valor de una interfase.

## Conclusión

Aunque la invención se ha descrito en lenguaje específico a las características estructurales y/o a los actos metodológicos, debe ser entendido que la invención definida en las demandas añadidas no está limitada a las características específicas ni a los actos descritos. Un poco, los actos y las características específicas se divulgan como formas ejemplares de implementación de la invención demandada.

**DEMANDAS**

1. Una interfase de programación incorporada en uno o más medios legibles por computador, comprende:

un primer grupo de servicios relacionados con la generación de objetos gráficos;

un segundo grupo de servicios relaconados con el contenido del formato; y

un tercer grupo de servicios relacionados con la creación de componentes de objetos gráficos.

2. Una interfase de programación como se describe en la demanda 1, en donde el primer, segundo y tercer grupos de servicios, comparten un modelo de programación común.

3. Una interfase de programación como se describe en la demanda 1, en donde el primer, el segundo y el tercer grupo de servicios, utilizan una lengua común enriquecido.

4. Una interfase de programación como se describe en la demanda 1, en donde el primer, el segundo y el tercer grupo de servicios comparten un sistema de vento común.

5. Una interfase de programación como se describe en la demanda 1, en donde el primer, el segundo y el tercer grupo de servicios comparten un sistema de definición de característica común.

6. Una interfase de programación como se describe en la demanda 1, en donde el primer, el segundo y el tercer grupo de servicios comparten un paradigma de entradacomún.

7. Una interfase de programación como se describe en la demanda 1, en donde el primer, el segundo y el tercer grupo de servicios comparten un sistema común para anidar los elementos asociados a un grupo particular de servicios dentro de los elementos asociados a otro grupo de servicios.

8. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios incluye un servicio que determine un aspecto de los objetos gráficos.

9. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios incluye un servicio que determina un comportamiento de los objetos gráficos.

10. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios incluye un servicio que determina un arreglo de los objetos gráficos.

11. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios incluye una pluralidad de elementos anidados que definen los objetos gráficos.

12. Una interfase de programación como se describe en la demanda 1, en donde los objetos gráficos comprenden uno o más elementos definidos por los gráficos del vector.

13. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios puede definir propiedades de la ventana en un lenguaje enriquecido sin cargar una nueva ventana.

14. Una interfase de programación como se describe en la demanda 1, en donde el primer grupo de servicios genera una interfase conteniendo una pluralidad de objetos gráficos.

15. Una interfase de programación como se describe en la demanda 1, en donde el segundo grupo de servicios organiza los objetos gráficos.

16. Una arquitectura de software que comprende la interfase de programación como se describe en la demanda 1.

17. Una interfase de programa de aplicación incorporada en uno o más medios legibles por computador, comprende:

un primer grupo de servicios relacionados con la generación de objetos gráficos;

un segundo grupo de servicios relacionados con el contenido del formato; y

un tercer grupo de servicios relacionados con la creación de componentes de objetos gráficos, en donde el primer, el segundo y el tercer grupo de servicios comparten un modelo de programación común.

18. Una Interfase de Programa de Aplicación (API-Application Program Interface) como se describe en la demanda 17, en donde el primer, el segundo y el tercer grupo de servicios utilizan un lenguaje común enriquecido.

19. Una Interfase de Programa de Aplicación como se describe en la demanda 17, en donde el tercer grupo de servicios incluye servicios para generar formas geométricas.

20. Una Interfase de Programa de Aplicación como se describe en la demanda 17, en donde el segundo grupo de servicios incluye el arreglo de una pluralidad de elementos de datos.

21. Una Interfase de Programa de Aplicación como se describe en la demanda 17, en donde el primer grupo de servicios incluye: un servicio que determina un aspecto de un objeto gráfico; y

un servicio que determina un comportamiento del objeto gráfico.

22. Una Interfase de Programa de Aplicación como se describe en la demanda 17, en donde el primer grupo de servicios incluye un

servicio que define características de la ventana en un lenguaje enriquecido sin cargar una ventana nueva.

23. Un sistema de cómputo que incluye uno o más microprocesadores y uno o más programas de software, el que utiliza uno o más programas de software utiliza una interfase de programación para solicitar servicios de un sistema operativo, la interfase de programación incluye comandos separados para solicitar los servicios de los grupos siguientes de servicios:

un primer grupo de servicios para generar objetos gráficos; y

un segundo grupo de servicios para crear los componentes de los objetos gráficos, en donde el primer y segundo grupo de servicios comparten un modelo de programación común.

24. Un sistema de cómputo como se describe en la demanda 23, en donde el primer grupo de servicios incluye:

un servicio para definir un aspecto de los objetos gráficos; y

un servicio para definir un arreglo de los objetos gráficos.

25. Un sistema de cómputo como se describe en la demanda 23, en donde el segundo grupo de servicios incluye servicios para generar una pluralidad de formas geométricas.

26. Un método comprende:

llamar una o más primeras funciones para facilitar la generación de objetos gráficos;

y

llamar una o más segundas funciones para facilitar el contenido del formato, en donde las primeras funciones y las segundas funciones comparten un modelo de programación común.

27. Un método como se describe en la demanda 26, incluye más llamadas de una o más terceras funciones para facilitar la creación de los componentes de los objetos gráficos.

28. Un método como se describe en la demanda 26, incluye más llamadas de una o más terceras funciones para facilitar la generación de formas geométricas contenidas en los objetos gráficos.

29. Un método como se describe en la demanda 26, en donde las primeras funciones facilitan:

la definición de las características de la ventana en una lenguaje enriquecido sin cargar una ventana nueva; y

la generación de una interfase de usuario que contiene una pluralidad de objetos gráficos.

30. Un sistema comprende:

los medios para exponer un primer conjunto de funciones que permiten la generación de objetos gráficos; y

los medios para exponer un segundo conjunto de funciones que permiten crear los componentes de los objetos gráficos, en donde los componentes de los objetos gráficos incluyen una pluralidad de formas geométricas, y en donde el primer conjunto de funciones y el segundo conjunto de funciones comparten un modelo de programación común.

31. Un sistema como se describe en la demanda 30, en donde el segundo sistema de funciones futuras permite más el arreglo de las formas geométricas en una página a ser entregada.

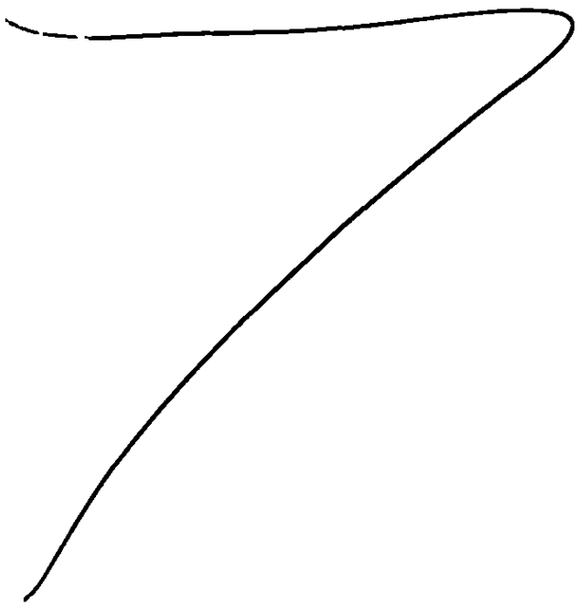
32. Un sistema como se describe en la demanda 30, comprende otros medios para exponer un tercer conjunto de funciones que permiten formatear el contenido a mostrar.

33. Un sistema como se describe en la demanda 30, en donde el primer conjunto de funciones y el segundo conjunto de funciones utilizan una lenguaje enriquecido.

34. Un sistema como se describe en la demanda 30, en donde el primer conjunto de funciones y el segundo sistema de funciones comparten un sistema de evento común y un sistema de definición de característica común.

## RESUMEN

Una interfase de programación proporciona funciones para generar las aplicaciones, los documentos, los medios de presentaciones y otro contenido. Estas funciones permiten que los desarrolladores obtengan servicios de un sistema operativo, el servicio modelo del objeto, u otro sistema o servicio.



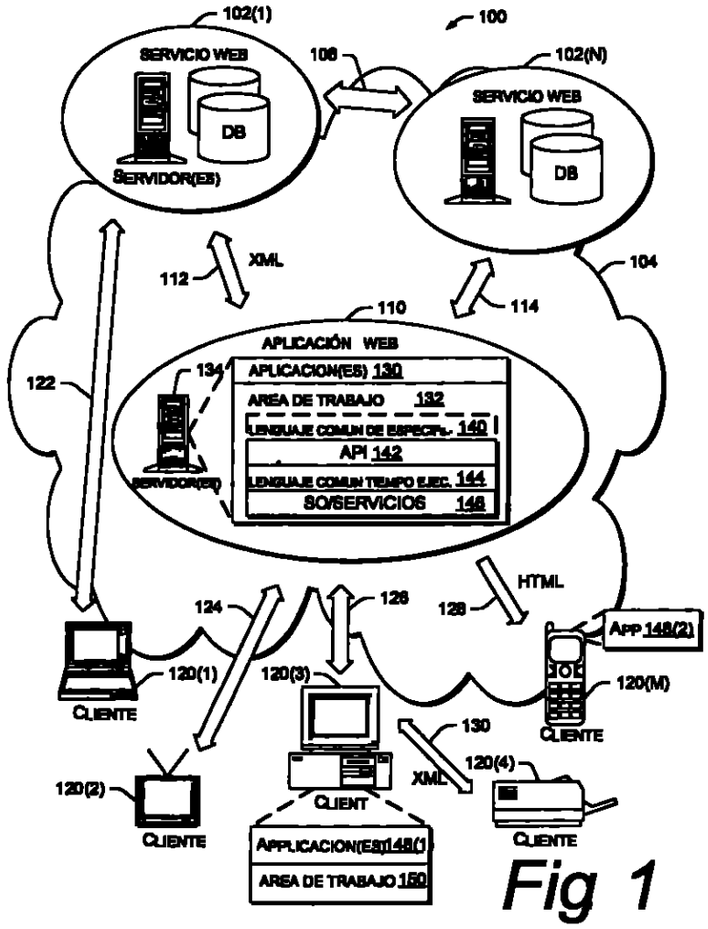

102(1)
100
SERVICIO WEB
106
102(N)
SERVICIO WEB
DB
SERVIDOR(ES)
DB
XML
112
104
110
114
APLICACIÓN WEB
APLICACION(ES) 130
AREA DE TRABAJO 132
LENGUAJE COMUN DE ESPECIFI. 140
API 142
LENGUAJE COMUN TIEMPO EJEC. 144
SO/SERVICIOS 146
122
134
SERVIDOR(ES)
HTML
128
124
126
APP 148(2)
120(1)
120(3)
120(M)
CLIENTE
CLIENTE
CLIENTE
130
120(4)
XML
120(2)
CLIENT
CLIENTE
APPLICACION(ES)148(1)
AREA DE TRABAJO 150

Fig 1

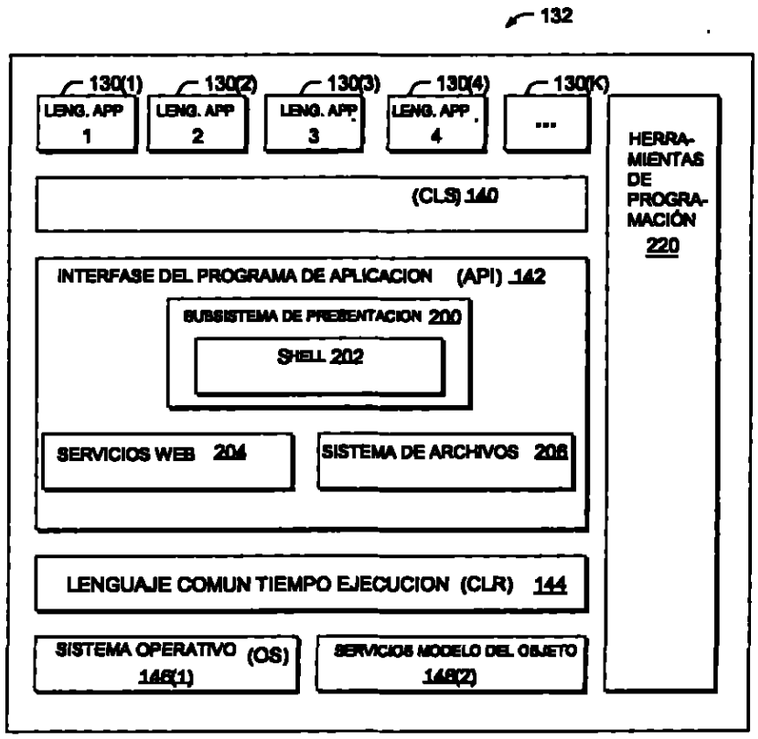

132
130(1)
130(2)
130(3)
130(4)
130(K)
LENG. APP 1
LENG. APP 2
LENG. APP 3
LENG. APP 4
...
HERRA-MIENTAS DE PROGRA-MACIÓN 220
(CLS) 140
INTERFASE DEL PROGRAMA DE APLICACION (API) 142
SUBSISTEMA DE PRESENTACION 200
SHELL 202
SERVICIOS WEB 204
SISTEMA DE ARCHIVOS 206
LENGUAJE COMUN TIEMPO EJECUCION (CLR) 144
SISTEMA OPERATIVO (OS) 146(1)
SERVICIOS MODELO DEL OBJETO 146(2)

Fig 2

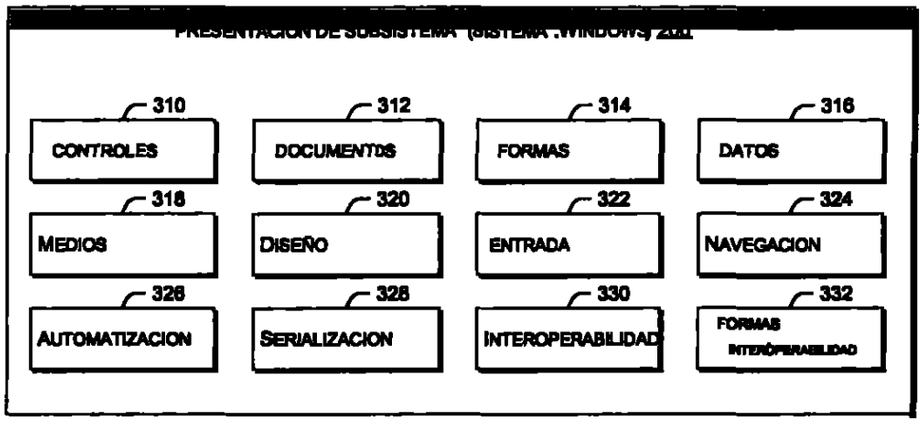

310
CONTROLES
312
DOCUMENTOS
314
FORMAS
316
DATOS
318
MEDIOS
320
DISEÑO
322
ENTRADA
324
NAVEGACION
326
AUTOMATIZACION
328
SERIALIZACION
330
INTEROPERABILIDAD
332
FORMAS
INTEROPERABILIDAD

Fig 3

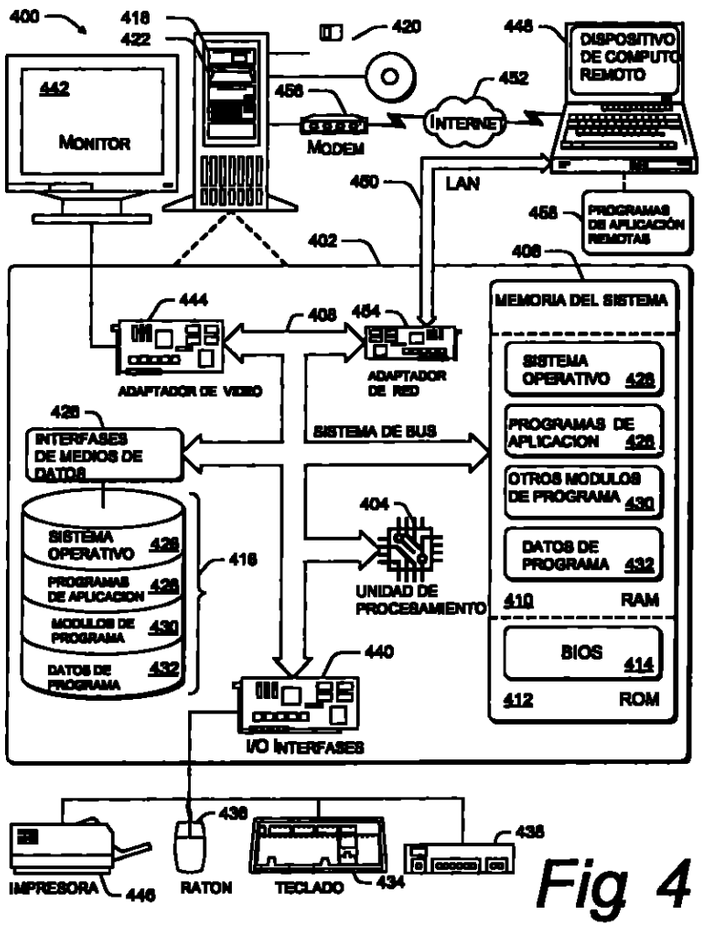

400
418
422
420
448
DISPOSITIVO DE COMPUTO REMOTO
442
MONITOR
456
MODEM
452
INTERNET
460
LAN
458
PROGRAMAS DE APLICACIÓN REMOTAS
402
406
MEMORIA DEL SISTEMA
444
408
454
ADAPTADOR DE VIDEO
ADAPTADOR DE RED
SISTEMA OPERATIVO 428
426
INTERFASES DE MEDIOS DE DATOS
SISTEMA DE BUS
PROGRAMAS DE APLICACION 428
OTROS MODULOS DE PROGRAMA 430
404
SISTEMA OPERATIVO 428
PROGRAMAS DE APLICACION 428
MODULOS DE PROGRAMA 430
DATOS DE PROGRAMA 432
416
UNIDAD DE PROCESAMIENTO
DATOS DE PROGRAMA 432
410
RAM
BIOS 414
440
412
ROM
I/O INTERFASES
436
438
IMPRESORA
446
RATON
TECLADO
434

Fig 4


FIGURA 5

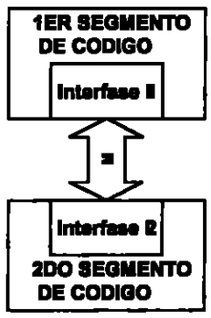



FIGURA 6


FIGURA 7

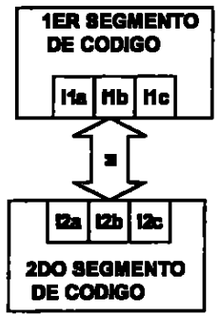



FIGURA 8

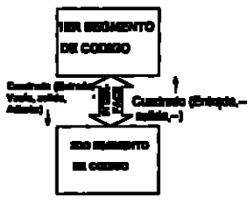



FIGURA9

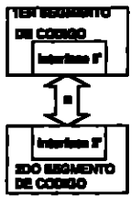



FIGURA10


FIGURA11

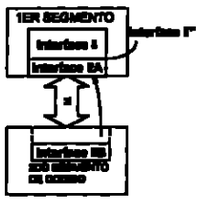



FIGURA12


FIGURA 13

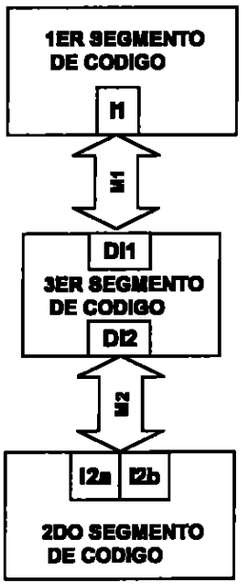



FIGURA 14

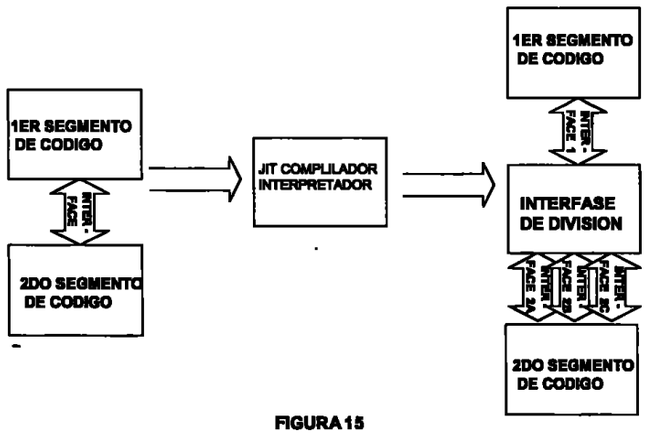
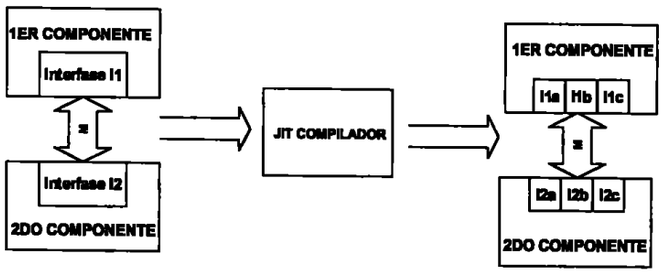

1ER SEGMENTO DE CODIGO
INTER-FACE
2DO SEGMENTO DE CODIGO
JIT COMPLILADOR INTERPRETADOR
1ER SEGMENTO DE CODIGO
INTER-FACE 1
INTERFASE DE DIVISION
INTER-FACE 2A
INTER-FACE 2B
INTER-FACE 2C
2DO SEGMENTO DE CODIGO

FIGURA 15




MS1-1780US

FIGURA 16

178

SUPERINDUSTRIA Y COMERCIO
Radicacion : 04088586 00000000 Folios 178
Fecha (AMD): 2004-09-08 12:28:39
Tramite : 002 PATENTES D 1 REGISTRO/ 411 PRESENTA
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

179

# REQUISITOS NECESARIOS PARA PRESENTACION Y AGILIZACION DEL TRAMITE

| | USUARIO | RADICADOR |
|---|---|---|
| **PATENTE DE INVENCIÓN** 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 o 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

Fecha

179

# PROTOCOLO

Expediente 04-88586

SUPERINDUSTRIA Y COMERCIO
SISTEMA DE REGISTRO

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 |

Encontrados (1/1) registro(s)

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

Industria y Comercio
SUPERINTENDENCIA

(80)

Folio 181

2020
Bogotá, D. C.

Doctor(a)
DARIO CARDENAS NAVAS
Apoderado(a)

Oficio No. 00782

**REFERENCIA:**

| | |
|---|---|
| Radicación | 04-88586 |
| Trámite | 002 |
| Evento | 001 |
| Actuación | 430 |
| Folios | 001 |

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

- Allegar copia del documento en que conste la cesión del derecho a la patente otorgado por el(los) inventor(es) al solicitante o a su causante. (art. 26-k,Dec 486/2000).

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

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

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

23 SE

El anterior requerimiento fue notificado por fijación en lista de fecha _____

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

202027

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