Industria y Comercio

SUPERINTENDENCIA

Delegatura de propiedad Industrial

División de Nuevas Creaciones

SOLICITUD

PATENTE DE INVENCIÓN

21 **EXPEDIENTE No.** 04-84792.

54 **TÍTULO** Diseño De Interfaces De Programación De Aplicación (API)

51 **CLASIFICACIÓN INTERNACIONAL** G 06 F 13/42

71 **SOLICITANTE** Microsoft Corporation.

DOMICILIO Redmond, Washington, Estados Unidos De America

74 **APODERADO** Dario Cardenas Navas.

22 **BOGOTÁ, D. C.,** _____

(FORMA P-10)

PETITORIO

Industria y Comercio
SUPERINTENDENCIA

SUPERINDUSTRIA Y COMERCIO
Radicacion : 04004792 00000000 Folios: 336
Fecha (AMD): 2004-08-30 16:00:43
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

			IDENTIFICACION
SOLICITANTE (71)	Nombre:	MICROSOFT CORPORATION	
	Dirección:	One Microsoft Way Redmond, Washington 98052, Estados Unidos de America	C.C. [ ] NIT [ ]
	Teléfono:	3123600 Fax: 3122410	C.E. [ ] Otro [X]
	E-Mail:	general@cardenasycardenas.com	
	Lugar de Constitución	Estado de Washington, Estados Unidos de América	Número
	Domicilio:	REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA.	

			IDENTIFICACION
REPRESENTANTE O APODERADO (74)	Nombre:	DARIO CARDENAS NAVAS	
	Dirección:	Carrera 7 No 71-52 Torre B Piso 9	C.C. [X] NIT [ ]
	Teléfono:	3123600 Fax: 3122410	C.E. [ ] Otro [ ]
	E-Mail:	general@cardenasycardenas.com	Número 17.068.629 BTA TP 1.250 C.S.J.

			IDENTIFICACION
INVENTOR(ES) (72)	Nombre:	1. Krzysztof J. Cwalina 2. Bradley M. Abrams 3. Anthony J. Moore 4. Christopher L. Anderson 5. Michael J. Pizzo 6. Robert A. Brigham II	
	Dirección:	1. 7583 Old Redmond Road, Apt. A204, Redmond, Washington 98052 2. 7517 128th Pl. NE Kirkland, Washington 98033 3. 4418 Corliss Avenue North, Apt. A Seattle, WA 98112 4. 16812 SE 45th Street, Issaquah, Washington 98027 5. 528 W Lake Sammamish PKWY SE Bellevue, Washington, 98008 6. 21226 NE 132 nd Ct. Woodinville, Washington 98072	C.C. [ ] NIT [ ]
	Teléfono:	3123600 Fax: 3123600	C.E. [ ] Otro [X]
	E-Mail:	general@cardenasycardenas.com	Número
	Domicilio:	1. WASHINGTON, ESTADOS UNIDOS DE AMERICA 2. WASHINGTON, ESTADOS UNIDOS DE AMERICA 3. WASHINGTON, ESTADOS UNIDOS DE AMERICA 4. WASHINGTON, ESTADOS UNIDOS DE AMERICA 5. WASHINGTON, ESTADOS UNIDOS DE AMERICA 6. WASHINGTON, ESTADOS UNIDOS DE AMERICA	

3 Título(54):

DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIÓN (API)"

6 Prioridad SI [X] NO [ ]

(33)País de Origen	(32)Fecha	(31)N° de Solicitud
U.S.A.	Octubre 23, 2003	10/692320

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

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

AMR/JAG/amg
No P-278

Comprobante de Pago No.: 04-66,139 / 04-66,140 Fecha Agosto 23, 2004

ANEXOS

- [X] Comprobante de pago de la tasa de presentación de la solicitud. (No. 04-66,139)
- [ ] 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 04-66,140)
- [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)
- [X] Copia certificada de la primera solicitud, si se reivindica prioridad.
- [X] Traducción simple de la primera solicitud, si se reivindica prioridad.
- [X] 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.

10

NOMBRE: DARIO CARDENAS NAVAS

FIRMA:

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

3

SUPERINDUSTRIA Y COMERCIO

Radicacion : 04084792 00000000 Folios: 336

Fecha (AMD): 2004-08-30 18:00:13

Tramite : 002 PATENTES I 1 REGISTRO/ 411 PRESENTA

Remitente: PARA DIVISION DE MARCAS DEPARTAMEN

NIT : 900176089-2

RECIBO OFICIAL DE CAJA : 04 - 66,139

FECHA : AGOSTO 23 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		24585399	23/08/2004	3,192,000.00

C O N C E P T O

CANT.	REMTISTICO	CONCEPTO	TOTAL CONCEPTO
1	50005-01-01 SOLICITUDES	1 TRAMITES DE SOL. DE PATENTE DE IN	422,000.
		TOTAL : $	422,000.00

SON: CUATROCIENTOS VEINTIDOS MIL PESOS

CARDENAS & CARDENAS

ABOGADOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACION (API)

3

SUPERINDUSTRIA Y COMERCIO
Radicacion : 04884792 00000000 Folios: 336
Fecha (AMD): 2004-08-30 10:00:43
Tramite : 002 PATENTES D 1 REGISTRO/ 411 PRESENTAL
Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 04 - 66,140
FECHA : AGOSTO 23 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	24585399	23/08/2004	3,192,000.00

C O N C E P T O

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

SON: CIENTO DIEZ Y OCHO MIL PESOS

RESPONSABLE : _____

CARDENAS & CARDENAS
ABOGADOS

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACION (API)

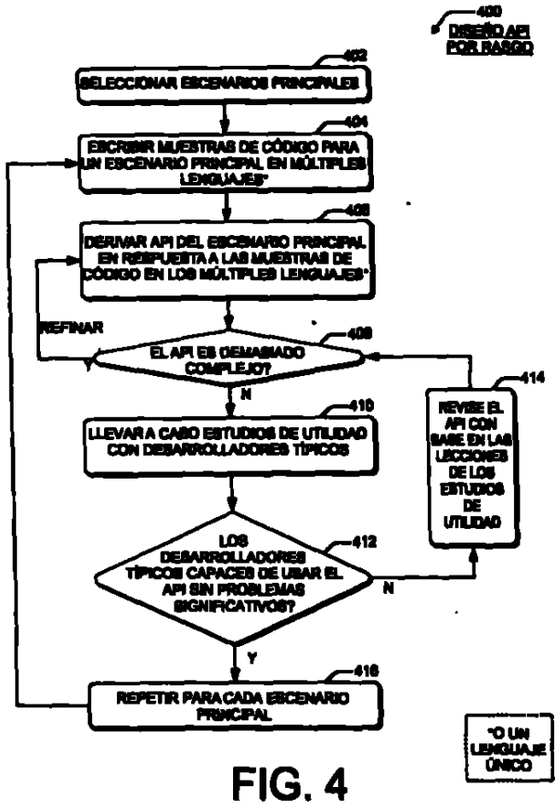

400
DISEÑO API POR RASGO
402
SELECCIONAR ESCENARIOS PRINCIPALES
404
ESCRIBIR MUESTRAS DE CÓDIGO PARA UN ESCENARIO PRINCIPAL EN MÚLTIPLES LENGUAJES*
406
DERIVAR API DEL ESCENARIO PRINCIPAL EN RESPUESTA A LAS MUESTRAS DE CÓDIGO EN LOS MÚLTIPLES LENGUAJES*
REFINAR
408
¿EL API ES DEMASIADO COMPLEJO?
Y
N
410
LLEVAR A CABO ESTUDIOS DE UTILIDAD CON DESARROLLADORES TÍPICOS
412
¿LOS DESARROLLADORES TÍPICOS CAPACES DE USAR EL API SIN PROBLEMAS SIGNIFICATIVOS?
N
Y
414
REVISE EL API CON BASE EN LAS LECCIONES DE LOS ESTUDIOS DE UTILIDAD
416
REPETIR PARA CADA ESCENARIO PRINCIPAL
*O UN LENGUAJE ÚNICO

FIG. 4

SHEET OF PARTICULARS

Application Information

MS File Number: 305796.01
Title of Invention: Design of Application Programming Interfaces (APIs)
Priority information:
Country: United States
Serial Number: 10/692320
Priority Filing Date: 10/23/2003
Additional dependencies, if any: see patent family below

Inventors:
Please use Microsoft Corporations address as the address for the inventor(s) below.

Inventors/Contributors		
Inventor Name	**Resource Type**	**Citizenship**
Anthony J. Moore	Inventor	United States
Bradley M. Abrams	Inventor	United States
Christopher L. Anderson	Inventor	United States
Krzysztof J. Cwalina	Inventor	United States
Michael J. Pizzo	Inventor	United States
Robert A. Brigham	Inventor	United States

Applicant: Microsoft is the only applicant unless additions are noted below as co-assignees

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
United States of America

State of Incorporation: Washington

Application document:

Attachments		
Document Type	**Attachment**	**Document Title**
Application Document	305796.1-Design of Application Programm	
Application Drawings	305796.1-Figures.vsd	

Figure for publication: 4

Specific Instructions

Please provide a copy of the application documents filed in this case, as well as a copy of all correspondence filed with and issued by the patent office throughout prosecution of this application. If you require any additional documentation at this time, please let us know immediately. In the absence of instructions, please take all actions necessary to maintain this application. This application should not be abandoned nor allowed to lapse without our prior written consent.

When forwarding an Office Action issued by the Patent Office, we ask that you send us a suitable draft response along with it for our review as we will be relying on your expertise and knowledge of local patent rules and laws.

Thank you for your assistance with this matter. **Please promptly acknowledge receipt of our instructions by electronic mail to sender with a copy to msintl@microsoft.com.**

Additional Information

The following information is provided as additional information for you to use when assigning your internal resources and to help you understand the strategy/relationships that this invention has within Microsoft. We do not have specific instructions as to the use of this information.

Patent Family: List of all other applications that have a parent/child relationship with instructions being sent.

Patent Family							
File #	**Parent File #**	**Priority File #**	**Additional Parent #**	**Case Type**	**Matter Type**	**File Type**	**Country**
305796.01	305796.01			NPR	ORG	NAT	US
305796.02	305796.01			NPR	ORG	EPC	EP
305796.03	305796.01			NPR	ORG	NAT	JP
305796.04	305796.01			NPR	ORG	NAT	CN
305796.05	305796.01			NPR	ORG	NAT	KR
305796.06	305796.01			NPR	ORG	NAT	IN
305796.07	305796.01			NPR	ORG	NAT	CA
305796.08	305796.01			NPR	ORG	NAT	AU
305796.09	305796.01			NPR	ORG	NAT	BR
305796.10	305796.01			NPR	ORG	NAT	RU
305796.11	305796.01			NPR	ORG	NAT	MX
305796.12	305796.01			NPR	ORG	NAT	ZA
305796.13	305796.01			NPR	ORG	NAT	TH
305796.14	305796.01			NPR	ORG	NAT	MY
305796.15	305796.01			NPR	ORG	NAT	TW
305796.16	305796.02			NPR	ORG	NAT	HK

305796.17	305796.01			NPR	ORG	NAT	NO
305796.18	305796.01			NPR	ORG	NAT	EG
305796.19	305796.01			NPR	ORG	NAT	PH
305796.20	305796.01			NPR	ORG	NAT	ID
305796.21	305796.01			NPR	ORG	NAT	IL
305796.22	305796.01			NPR	ORG	NAT	SG
305796.23	305796.01			NPR	ORG	NAT	CL
305796.24	305796.01			NPR	ORG	NAT	CO
305796.25	305796.01			NPR	ORG	NAT	NZ

ROMP Class: Microsoft internal classification listing. This can be used when assigning work within your firm.

ROMP	
ROMP Code	**ROMP Description**
18.1	Interfaces
6.5	Programming Models/Design Patterns

Microsoft division: Server Platform

Products: Known products relating to this invention

Related Products
Selected Products
Longhorn
- Whidbey

US Law Firm: This firm/attorney handled the original drafting of the application.

OC Resource		
Law firm	**Attorney**	**OC Reference**
Lee and Hayes	Keith Saunders	MS1-1748US

Page counts: **these counts are specific to the pagination as printed by US Counsel**
Specification: 62
Drawings: 10
Claims: 16

Keywords:

Keywords	
Related Keywords	**Keyword Type**
Tier 3	Country Tiers

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Design of Application Programming Interfaces (APIs)

Inventor(s):
Krzysztof J. Cwalina
Bradley M. Abrams
Anthony J. Moore
Christopher L. Anderson
Michael J. Pizzo
Robert A. Brigham II

ATTORNEY'S DOCKET NO. MS1-1748US

Design f Application Programming Interfaces (APIs)

TECHNICAL FIELD

This disclosure relates in general to application programming interfaces (APIs) and in particular, by way of example but not limitation, to designing APIs that are easy to use while simultaneously providing control and flexibility.

BACKGROUND

Application programming interfaces (APIs) are used by developers to create a wide variety of applications and programs. Developers range from office workers recording macros to low-level device driver authors. These developers rely on different languages and/or different frameworks of differing complexities while programming with different skill sets and/or for different purposes. Traditionally, different APIs have been designed to target different individual levels of skill and different demands for control (e.g., based on different relevant scenarios).

Although this approach can be successful in providing APIs that are optimized for a specific developer, it has significant drawbacks. For example, the multiple framework approach creates situations where developers have difficulty transferring knowledge from one skill level and scenario type to another. When there is a need for them to implement a scenario using a different framework, developers hit a very steep learning curve. And not only is the learning curve very steep, but it generally requires that the code written to a first lower-skill-level framework has to be rewritten from scratch to a second higher-skill-level

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

framework. Moreover, the creation of separate frameworks for different developer skill levels typically results in a situation in which APIs that are targeted for or implemented by one level of developer are unusable by another level of developer.

FIG. 1 illustrates a graph 101 of a traditional API learning curve with regard to two different frameworks. The first framework corresponds to a framework that has a relatively lower level of required skills and/or difficulty and a concomitantly relatively lower capacity for control by a developer. The second framework, on the other hand, corresponds to a framework that has a relatively higher level of required skills and/or difficulty and a concomitantly relatively higher capacity for control by a developer. Such a first framework might be used by a novice or infrequent developer, and such a second framework might be used by an experienced or professional developer. For example, the first framework may correspond to one designed for Visual Basic, and the second framework may correspond to one designed for C++.

In this traditional approach, relatively separate and disparate APIs are designed and employed as part of each framework. A steep but relatively short learning curve is traversed to enable API usage for the first framework at the relatively lower skill level and control capability. Because of the separate and disparate nature of the two API frameworks, the experience with the first framework contributes little if any knowledge toward learning the second API of the second framework. Consequently, an equally steep but even taller learning curve is traversed to enable API usage for the second framework.

In other words, learning an API of the first framework does not provide a stepping stone to learning an API of the second framework. The regressive nature of this disjointed set of API frameworks is indicated by the continuity gap. A

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

13

developer who has learned the API of the first framework is no closer to learning the API of the second framework and must therefore start with the basics of the second framework.

Another problem with traditional frameworks is that they tend to have an overall poor usability in any case. In general, object oriented design/development (OOD) methodologies (e.g. unified modeling language (UML)) are "optimized" for maintainability of the resulting design and not for usability of the resulting frameworks. OOD methodologies are better suited for internal architecture designs and less suited for designs of an API layer of a large reusable library. For example, poor usability can result from OOD methodologies that focus only on distillation to a lowest fundamental block and/or that have an unwavering allegiance to a strict inheritance hierarchy throughout an API design.

Accordingly, there is a need for schemes and/or techniques that can at least ameliorate the regressive continuity gap of a traditional API learning curve and/or that can deliver better overall API usability.

SUMMARY

In a first exemplary method implementation, a method for designing an application programming interface (API) includes: preparing multiple code samples for a core scenario, each respective code sample of the multiple code samples corresponding to a respective programming language of multiple programming languages; and deriving the API from the core scenario responsive to the multiple code samples. In a second exemplary method implementation, a method for designing an API includes: selecting a core scenario for a feature area; writing at least one code sample for the core scenario; and deriving an API for the

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

core scenario responsive to the at least one code sample. In a third exemplary method implementation, a method for designing an API includes: deriving an API for a scenario responsive to at least one code sample written with regard to the scenario; performing one or more usability studies on the API utilizing multiple developers; and revising the API based on the one or more usability studies.

Other method, system, approach, apparatus, device, media, API, procedure, arrangement, etc. implementations are described herein.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like and/or corresponding aspects, features, and components.

FIG. 1 illustrates a graph of a traditional API learning curve with regard to two different frameworks.

FIG. 2 illustrates a graph of an exemplary progressive API learning curve with regard to two different levels of abstraction.

FIG. 3 illustrates exemplary design principles and practices for APIs.

FIG. 4 is a flow diagram that illustrates an exemplary technique for designing APIs per feature area.

FIG. 5 is a block diagram that illustrates an exemplary scheme for designing APIs per core scenario.

FIG. 6 illustrates potential disparity between exemplary component types that are targeted to two different purposes.

FIG. 7 illustrates an exemplary relationship between component types that are designed to be extensible and/or interoperable so as to cover two different purposes.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

FIG. 8 illustrates an exemplary aggregate component (AC) and associated factored types (FTs) for handling two different purposes with a two-layer API.

FIG. 9 illustrates an exemplary aggregate component and associated APIs that can support a create-set-call usage pattern.

FIG. 10 illustrates an exemplary computing (or general device) operating environment that is capable of (wholly or partially) implementing at least one aspect of designing and/or using APIs as described herein.

DETAILED DESCRIPTION

FIG. 2 illustrates a graph 200 of an exemplary progressive API learning curve with regard to two different levels of abstraction. The two different illustrated levels of abstraction are a relatively high level of abstraction and a relatively low level of abstraction. The high level of abstraction corresponds to a development environment that involves a relatively lower level of required skills and/or difficulty and a concomitantly relatively lower capacity for control by a developer. The low level of abstraction, on the other hand, corresponds to a development environment that involves a relatively higher level of required skills and/or difficulty and a concomitantly relatively higher capacity for control by a developer.

A progressive API learning curve is shown rising from a point of lower required skills and concomitant control capability in a relatively smooth manner through the areas for the high and low levels of abstraction to a point of higher required skills and concomitant control capability. The progressive API learning curve exhibits a continuity zone between the areas of the high level of abstraction and the low level of abstraction. An integrated API framework enables a gradual

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

learning curve. Because of the integrated nature of the API framework, experience with the high level of abstraction contributes to knowledge toward learning API usage for the low level of abstraction as well as for scenarios demanding greater control.

In other words, learning the API for higher levels of abstraction provides a stepping stone to learning and/or extending the API into lower levels of abstraction. This is indicated by the two-layer (API) framework shape encompassing both the high and the low level of abstraction areas. The progressive nature of certain APIs, as described herein below, enable developers to use simple APIs initially and to gradually (and partially) begin using more complicated API components. Thus, developers who have learned the APIs targeting the higher levels of abstraction can move to using the APIs targeting the lower levels of abstraction as their experience warrants and/or as the complexity of the scenarios that they are facing demand.

A progressive API can be easily usable (especially during early learning phases) and highly powerful (especially as the API is explored over time). A usable API may include one or more of the following exemplary attributes: a small number of concepts and/or classes are required to get started, a few lines of code can implement simple scenarios, classes/methods have intuitive names, a natural and/or obvious starting point is apparent, and there is a clear (e.g., discoverable) progression to additional required and/or relevant concepts/classes.

A progressive API can also enable an incremental advancement from developing at a point of lower difficulty and concomitant control capability to a point of higher difficulty and concomitant control capability. Exemplary

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

paradigms for designing progressive APIs, as well as generally highly usable APIs, are described herein below.

FIG. 3 illustrates exemplary design principles and practices for APIs in a table 300. Table 300 indicates general design principles and practices for four exemplary categories 302-308. Specifically, the following four categories are addressed: scenario driven design 302, component oriented design 304, customizable defaults 306, and self documenting object model 308.

When designing a given API, the design principles and practices for any one or more of the indicated categories 302-308 may be employed. Furthermore, within any given category 302-308, one or more of the illustrated design principles and practices may be implemented. In other words, neither every category nor every design principle and practice thereof need be employed or implemented for a given API design.

Scenario driven design category 302 illustrates four exemplary design principles and practices. Firstly, core scenarios for selected features or technological areas are defined. Secondly, code samples corresponding to the core scenarios are written first, and the API is designed responsive thereto second. Thirdly, a progressive API, as introduced above and described further herein below, is designed. Fourthly, utilizing the defined core scenarios is made easy while utilizing other scenarios is made possible. Scenario driven design 302 is described further below in the section entitled "Scenario Driven Design".

Component oriented design category 304 illustrates three exemplary design principles and practices. Firstly, aggregate components (ACs) are created. Generally, aggregate components are directed toward core scenarios, are relatively simple and easy to use, and are built on top of factored types (FTs). Factored

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

types are more fundamental and are decomposed to lower logical levels. This results in a two-layer API design. Secondly, these aggregate components are interrelated with the factored types. Thirdly, a create-set-call usage pattern is supported, especially for aggregate components. Component oriented design 304 is described further below in the section entitled "Component Oriented Design".

Customizable defaults category 306 illustrates two exemplary design principles and practices. Firstly, required initializations to use at least aggregate components are reduced. Defaults are used to reduce required initializations. Secondly, defaults are selected that are appropriate for the defined core scenarios. Customizable defaults 306 is described further below in the section entitled "Customizable Defaults".

Self documenting object model category 308 illustrates four exemplary design principles and practices. Firstly, simple and intuitive names are reserved for core scenarios. Secondly, names are selected based on the intended use or purpose of the component type, instead of a hidebound adherence to the inheritance hierarchy. Thirdly, actionable exceptions are thrown so that a developer receives instructions indicating how to fix an error from the exception message. Fourthly, clean namespaces are produced by placing types that are rarely used into sub-namespaces to avoid cluttering the main namespaces. Self documenting object model 308 is described further below in the section entitled "Self Documenting Object Model".

Scenario Driven Design

In a described implementation, API specifications are driven by scenarios. Accordingly, API designers first write the code that the users of the API will have to write in core (e.g., main) scenarios. API designers then design an object model

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

to support these code samples. This approach contrasts with starting a design of an object model (using various design methodologies) and then writing code samples based on the resulting API.

In other words, especially for public API design, API designers start with a list of scenarios for each feature or technology area and code samples therefor and produce a header-style object model description based thereon. Examples of feature areas include: file I/O, networking, messaging, console, diagnostics, database access, web pages, graphical user interface (GUI) programming, and so forth.

FIG. 4 is a flow diagram 400 that illustrates an exemplary technique for designing APIs per feature area. At block 402, core scenarios are selected for the given feature area. For example, for a given technology area, the top 5-10 scenarios may be selected. They may be selected based on the most commonly used functions (e.g., most common tasks) or the most frequently pursued goals for the given technology area. For instance, exemplary scenarios for a file I/O technology feature area are reading from a file and writing to a file.

At block 404, code samples for a core scenario are written in multiple (e.g., two or more) languages. For example, code samples associated with a selected core scenario may be written in three different languages. The code samples may implement the current selected core scenario in the three languages. Such languages include, for example, VB, C#, MC++, a markup language, and so forth; however, other languages may also be used. As indicated by the asterisk, it should be understood that a code sample (or even more than one code sample) may be written for the core scenario in a single language when designing a usable and powerful API for a single language.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1743US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

20

Writing code samples in multiple languages may be performed because sometimes code written in different languages differs significantly. In a described implementation, the code samples for the current selected core scenario are written using different coding styles that are common among users of the particular language (e.g., using language-specific features or traits, using the practices/habits of developers, etc.) in which a particular code sample is written. For example, the samples may be written using language-specific casing. For instance, VB is case-insensitive, so code samples written in VB reflect that variability. Code samples written in C#, on the other hand, follow the standard casing therefor.

Another example relates to a statement called "using", which C# supports. For instance, the "using" call encapsulates a try/finally block. However, VB does not support this feature, and writing code samples can indicate that utilizing this feature in a try/finally statement is awkward for VB users. Yet another example relates to assignments in a conditional clause, which C# supports. In a file I/O instance: "if ((text = reader.ReadLine() != null)" works in C#. However, the assignment statement cannot be used within the "if" clause in VB; instead, the code is broken into multiple statements. Still yet another example relates to the tendency of C# developers to utilize parameterized constructors while VB developers usually do not. For instance, a C# coding may be "MyClass x = new MyClass("value")" while a corresponding VB coding is "Dim x As MyClass" and "x.Property = "value"."

At block 406, an API is derived from the current core scenario responsive to the code samples written in the multiple languages. For example, factors gleaned from the code samples written in each of the multiple languages may be incorporated into the API. Such factors may include similarities across the

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

different code samples, differences between/among two or more code samples, and so forth. Such factors, as well as other aspects of blocks 404 and 406, are described further below with reference to FIG. 5.

Similarly, when designing an API for a single language, the API is derived from the current core scenario responsive to the code sample(s) written in the single language. Thus, factors gleaned from the code sample(s) written in the single language may be incorporated into the API. As an additional API design factor example for single or multiple language situations, an API design factor may include compatibility with tools that are oriented toward the language or languages for which the code sample(s) are written.

At block 408, it is determined if the API is too complex. For example, the API may be reviewed by the API designer(s) to determine if the API is or is not too complex. In other words, an initial check may be performed to consider whether the API can be used without significant understanding of multiple other specific APIs, without undue experimentation, and so forth. Such an initial check may also verify that the derived API is actually workable in the current core scenario in every relevant language. If the API is too complex, then the API is refined by the designers with reference to the current core scenario and responsive to the code samples written in the multiple languages at block 406.

If, on the other hand, it is determined that the API is not too complex (at block 408), then at block 410 usability studies with typical developers are performed. For example, one or more usability studies may be performed using a development environment akin to that which the typical developer normally uses. Such a normal development environment likely includes intellisense, editors,

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

language, and a documentation set that is most widely used by the targeted developer group.

Usability Studies

Usability studies that target a wide range of developers facilitate scenario-driven design, especially when designing general public APIs. The code samples written by the API designer(s) for the core scenarios probably appear simple to them, but the code samples might not be equally simple to certain groups of developers that are in fact targeted (e.g., especially novice and/or occasional developers). Additionally, the understanding, which is garnered through usability studies, regarding the manner in which developers approach each core scenario can provide powerful insight into the design of the API and how well it meets the needs of all of the targeted developers.

Generally, usability studies may be conducted early in the product cycle and again after any major redesign of the object model. Although this is a costly design practice, it can actually save resources in the long run. The cost of fixing an unusable or merely defective API without introducing breaking changes is enormous.

At block 412, it is ascertained whether typical developers are able to use the API without significant problem(s). For example, most subjects should be able to write code for the current selected scenario without major problems. If they cannot, the API is revised (as described below with reference to block 414).

The interpretation of significant/major problems hinges on a desired level of usability for a given targeted developer group. For example, frequent and/or extensive reference to detailed API documentation for the current core scenario by test subjects may constitute significant problems. Generally, if the majority of test

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

developers cannot implement the current core scenario, or if the approach they take is significantly different from what was expected, the API should be evaluated for possible revisions (up to and including a full redesign).

If it is ascertained that typical developers are unable to use the API without major problems (at block 412), then at block 414 the API is revised based on lessons from the usability studies. For example, a default may be changed, another property may be added, one or more attributes may be exposed instead of encapsulated, and so forth.

If, on the other hand, it is ascertained that typical developers are able to use the API without major problems (at block 412), then at block 416 the process is repeated for each core scenario. For example, another core scenario of the selected core scenarios for the given feature becomes the current core scenario for which code samples are written (at block 404). Another exemplary technique for designing APIs, which focuses more on two-layer API design, is described further below in conjunction with FIG. 8.

FIG. 5 is a block diagram 404/406 that illustrates an exemplary scheme for designing APIs per core scenario. The illustrated exemplary scheme corresponds to blocks 404 and 406 of FIG. 4 for a multiple language implementation. A code sample 502(1) for a first language, a code sample 502(2) for a second language, and a code sample 502(3) for a third language is shown. Each of the three code samples 502(1, 2, 3) are directed to a given current core scenario. Although three code samples 502(1, 2, 3) corresponding to three languages are shown, two or more code samples 502 of any arbitrary number of targeted languages may alternatively be used in this exemplary multiple-language implementation.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

In a described implementation, factors 506 are gleaned from code samples 502(1, 2, 3) that are written in each of the three languages. These factors 506 are incorporated into the API 504. Specifically, API 504 is designed to support the three code samples 502(1, 2, 3) that are written in the three respective corresponding languages. However, it should be understood that factors 506 may also be applicable to single-language implementations.

Some exemplary factors 506 are described above with reference to blocks 404 and 406 of FIG. 4, and other exemplary factors 506 are indicated in block diagram 404/406 of FIG. 5. Such factors 506 include language-specific mandates that are revealed by a review of code samples 502(1, 2, 3). An example of a language-specific constraint is described with regard to the following sample line of code: "Foo f = new Foo();". A progressive API that is designed to support this sample line has to include a default constructor; otherwise, the code sample does not compile correctly.

Factors 506 also include developer expectations that are inspired by both language peculiarities and the different skill/experience levels of typical developers that naturally gravitate toward the different languages. Factors 506 further include commonalities of code and coding practices across the different languages as discoverable by a review of code samples 502(1, 2, 3).

While considering factors 506 that directly relate to different languages, other factors 506, as described herein, continue to be considered. For example, the following factors 506 are also pertinent to progressive APIs targeted to single-language environments as well as multiple-language environments. First, the number of different component types that are required to complete a scenario is a factor. Generally, the more component types that are required, the harder it is to

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

learn. A second factor is the connection between succeeding lines of code. To the extent that usage of one component type leads a developer towards usage of the next required component type, the easier the API is to use.

Third, consistency in the naming of identifiers is another factor. A fourth factor involves the appropriate usage of properties, methods, and events. A fifth factor relates to possible similarities to one or more existing APIs. Sixth, another factor involves compliance with overall design guidelines for an API. A seventh factor relates to whether the APIs overlap with other component types of the framework. Eighth, compatibility with tools that are oriented toward a particular language is yet another factor. For instance, VB developers typically want parameter-less constructors and property setters. Other factors 506 may alternatively be considered.

Still other factors 506 that relate to interrelationships of aggregate components and factored types are described below, especially with reference to FIG. 8. Although the method and scheme of FIGS. 4 and 5 may be applied to designing APIs in general, they are particularly applicable to designing two-layer APIs. A two-layer API paradigm (e.g., with aggregate components and factored types) is described below with reference to FIGS. 6-8 in the section entitled "Component Oriented Design".

Component Oriented Design

FIG. 6 illustrates potential disparity between exemplary component types 602 that are targeted to two different purposes along a continuum 600. Continuum 600 extends from a high usability range on the left side to a high controllability range on the right side. Multiple component types 602 are spread across continuum 600.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Generally, component types 602 that are illustrated as being relatively larger represent types that are simpler and therefore easier to use. Conversely, component types 602 that are illustrated as being relatively smaller represent types that are more complex and therefore more difficult to use. Simple and complex in this context refer to how easy or how difficult the particular component types 602 are to use when implementing a specified scenario.

The component types 602 that are illustrated as being relatively smaller are generally more difficult to use for a number of exemplary reasons as follows: First, developers have more choices as to which component types 602 they should use. In the illustrated example, there are 14 "choices" for the smaller component types 602 as compared to three "choices" for the larger component types 602. More specifically, a developer has to know or discern, from among the various component types 602(HC), which component type or types to use. This involves understanding each of the (e.g., 14) multiple component types 602(HC) as well as how they interrelate, which contrasts with starting with a single component type 602(HU) from among the fewer (e.g., 3) component types 602(HU). Differences between component types 602(HU) and component types 602(HC) are described further below.

By way of an exemplary analogy, the smaller component types 602 are like the individual components of a stereo system; hence, a user has to know which components are needed and how to hook them together. Without hooking them together, they are generally not useful. The larger component types 602 are like all-in-one stereos that are easily usable but likely to be less powerful as well as less flexible. A second reason that smaller component types 602 are harder to use is that there are potentially more "starting points". Third, there are generally more

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

concepts to understand. Fourth, a developer has to understand how individual components types 602 relate to other component types 602.

In a described implementation, component types 602 are divided into those with a high usability purpose 602(HU) and those with a high controllability purpose 602(HC). High usability component types 602(HU) are simpler and easier to use, but they tend to be more inflexible, limiting, and/or constraining. They can generally be used without extensive knowledge of an overall API. High usability component types 602(HU) are usually capable of implementing a limited number of scenarios or at most a limited number of approaches to each scenario of interest.

High controllability component types 602(HC), on the other hand, are complex to use, but they provide a greater degree of control to developers. They are relatively powerful and enable developers to effectuate low-level tweaking and tuning. However, developing with high controllability component types 602(HC) entails a fuller understanding of many component types 602 to enable the instantiation of multiple high controllability component types 602(HC) that are correctly interlinked to implement even relatively straight-forward scenarios.

Typically, high usability component types 602(HU) are present in introductory languages such as VB, and high controllability component types 602(HC) are present in advanced professional-programmer-type languages such as C++. The potential disparity between high usability component types 602(HU) and high controllability component types 602(HC) that is illustrated in FIG. 6 is at least partly ameliorated by component types 702 of FIG. 7. Specifically, an interrelationship between high usability component types 602(HU) and high controllability component types 602(HC) is established by a progressive API.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

FIG. 7 illustrates an exemplary relationship between component types 702 that are designed to be extensible and/or interoperable so as to cover at least two different purposes along a continuum 700. In a described implementation, component types with a high usability purpose are realized as aggregate components 702(AC), and component types with a high controllability purpose are realized as factored types 702(FT). Although component types 702 are divided into only two purposes, they may alternatively be separated into three or more purposes (or other categories).

A key 704 indicates that a solid line represents a relationship for exposed factored types and that a dashed line represents a relationship for encapsulated factored types. As illustrated, aggregate component 702(AC)(1) has a relationship with three factored types 702(FT). Specifically, factored type 702(FT)(1) has an exposed factored type relationship with aggregate component 702(AC)(1), and factored types 702(FT)(2) and 702(FT)(3) have an encapsulated factored type relationship with aggregate component 702(AC)(1).

Although not so illustrated, two or more aggregate components 702(AC) may have an encapsulated and/or exposed relationship with the same factored type 702(FT). Exposed and encapsulated factored types 702(FT) and aggregate components 702(AC), as well as relationships therebetween, are described further below, including with reference to FIG. 8.

Component oriented design relates to offering a single object per user concept as opposed to requiring multiple objects per logical concept. Aggregate components therefore usually correspond to a user concept and are simpler from a usability perspective. Aggregate components are layered on top of factored types. By way of an exemplary comparison, aggregate components may model a thing

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

such as a file, and factored types may model a state of a thing such as a view on the file. Together, aggregate components and factored types provide a progressive and gradual learning curve for new developers, especially with respect to a particular given API.

Component Oriented Design for Aggregate Components

Many feature areas may benefit from façade types that act as simplified views over a more complex but well-factored remainder of the feature area APIs. In a described implementation, the façade covers the top 5-10 scenarios in a given feature area and optionally other high-level operations. Aggregate components 702(AC) can serve as such façade types, and factored types 702(FT) can provide a remaining well-factored complex API landscape.

Each aggregate component ties multiple lower level factored classes into a higher-level component to support the top core scenarios. For example, a mail aggregate component may tie together SMTP protocol, sockets, encodings, and so forth. Generally, each aggregate component provides a higher abstraction level rather than just a different way of doing things. Providing simplified high-level operations is helpful for those developers who do not want to learn the whole extent of the functionality provided by a feature area and merely wish to accomplish their often very simple tasks without significant study or API exploration.

Generally, component oriented design is a design based on constructors, properties, methods, and events. Using aggregate components is a relatively extreme application of component oriented design. An exemplary set of parameters for component oriented design of aggregate components is provided below:

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Constructors: aggregate components have default (parameter-less) constructors.

Constructors: optional constructor parameters correspond to properties.

Properties: most properties have getters and setters.

Properties: properties have sensible defaults.

Methods: methods do not take parameters if the parameters specify options that stay constant across method calls (in the selected core scenarios). Such options may be specified using properties.

Events: methods do not take delegates as parameters. Callbacks are implemented in terms of events.

Component oriented design entails considering how the API is used instead of focusing on the mere inclusions of the methods, properties, and events in the object model. An exemplary usage model for component oriented design involves a pattern of instantiating a type with a default or relatively simple constructor, setting some properties on the instance, and then calling simple methods. This pattern is termed a Create-Set-Call usage pattern. A general example follows:

```
' VB
' Instantiate
Dim T As New T()

' Set properties/options.
T.P1 = V1
T.P2 = V2
T.P3 = V3
```

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

```
' Call methods; optionally change options between calls.
T.M1()
T.P3 = V4
T.M2()
```

When aggregate components support this Create-Set-Call usage pattern, the aggregate components comport with the expectations of the main users of aggregate components. Moreover, tools, such as intellisense and designers, are optimized for this usage pattern. A concrete code example showing the Create-Set-Call usage pattern follows:

```
' VB
' Instantiate
Dim File As New FileObject()

' Set properties.
File.Filename = "c:\foo.txt"
File.Encoding = Encoding.Ascii

' Call methods.
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
File.Close()
```

With an exemplary aggregate component that is part of a progressive API, setting the "File.Encoding" property is optional. The API has a default for a pre-selected

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

file encoding if one is not specified. Similarly, with regard to "File.Open()", specifying an "OpenMode.Write" is optional. If it is not specified, a default "OpenMode" as pre-selected by the API is employed.

An issue with component oriented design is that it results in types that can have modes and invalid states. For example, a default constructor allows users to instantiate a "FileObject" without specifying a "FileName". Attempting to call Open() without first setting the "FileName" results in an exception because the "FileObject" is in an invalid state with respect to being opened (e.g., no file name has yet been specified). Another issue is that properties, which can be set optionally and independently, do not enforce consistent and atomic changes to the state of the object. Furthermore, such "modal" properties inhibit sharing of an object instance between consumers because a first user has to check a previously-set value before reusing it in case a second user has changed the value in the interim. However, the usability of aggregate components outweighs these issues for a vast multitude of developers.

When users call methods that are not valid in the current state of the object, an "InvalidOperationException" is thrown. The exception's message can clearly explain what properties need to be changed to get the object into a valid state. These clear exception messages partially overcome the invalid state issue and result in an object model that is more self-documenting.

API designers often try to design types such that objects cannot exist in an invalid state. This is accomplished, for example, by having all required settings as parameters to the constructor, by having get-only properties for settings that cannot be changed after instantiation, and by breaking functionality into separate types so that properties and methods do not overlap. In a described

Eliminado: MSI-1749US PATAPP (2).DOC

Insertado: MSI-1748US PATAPP (2).DOC

Eliminado: MSI-1748US.PATAPP

implementation, this approach is employed with factored types but not with aggregate components. For aggregate components, developers are offered clear exceptions that communicate invalid states to them. These clear exceptions can be thrown when an operation is being performed, instead of when the component is initialized (e.g., when a constructor is called or when a property is set), so as to avoid situations where the invalid state is temporary and gets "fixed" in a subsequent line of code.

Factored Types

As described above, aggregate components provide shortcuts for most common high level operations and are usually implemented as a façade over a set of more complex but also richer types, which are called factored types. In a described implementation, factored types do not have modes and do have very clear lifetimes.

An aggregate component may provide access to its internal factored types through some properties and/or methods. Users access the internal factored types in relatively advanced scenarios or in scenarios where integration with different parts of the system is required. The ability to access factored type(s) that are being used by an aggregate component enables code that has been written using the aggregate component to incrementally add complexity for advanced scenarios, or integrate with other component types, without having to re-write code from the beginning with a focus on using the factored types.

The following example shows an exemplary aggregate component ("FileObject") exposing its exemplary internal factored type ("StreamWriter"):

`VB

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

```
Dim File As New FileObject("c:\foo.txt")
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
AppendMessageToTheWorld(File.StreamWriter)
File.Close()
...

Public Sub AppendMessageToTheWorld(ByVal Writer As
StreamWriter)
   ...
End Sub
```

High Level Operations

In a described implementation, aggregate components, as the upper or higher level APIs (e.g., from a level of abstraction perspective), are implemented such that they appear to "magically" work without the user being aware of the sometimes complicated things happening underneath. For example, an "EventLog" aggregate component hides the fact that a log has both a read handle and a write handle, both of which are opened in order to use it. As far as a developer may be concerned, the aggregate component can be instantiated, properties can be set, and log events can be written without concern for the under-the-hood functioning.

In some situations, a bit more transparency may facilitate some task with the developer. An example is an operation in which the user takes an explicit action as a result of the operation. For instance, implicitly opening a file and then requiring the user to explicitly close it is probably taking the principle of

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

"magically" working too far. Nevertheless, a diligent API designer may often be capable of designing clever solutions that hide even those complexities. For example, reading a file can be implemented as a single operation that opens a file, reads its contents, and closes it; the user is thus shielded from the complexities related to opening and closing the file handles.

Furthermore, using aggregate components does not involve implementing any interfaces, modifying any configuration files, and so forth. Instead, library designers can ship default implementations for interfaces that are declared. Moreover, configuration settings are optional and backed by sensible defaults.

FIG. 8 illustrates an exemplary aggregate component 702(AC) and associated factored types 702(FT) for handling two different purposes with a two-layer API 800. Aggregate component 702(AC) represents a first or higher layer, and factored types 702(FT) represent a second or lower layer. The first layer effectively builds on the second layer with a custom interface.

As illustrated, aggregate component 702(AC) includes multiple aggregate component (AC) members 802. Specifically, aggregate component members 802(1), 802(2), 802(P)(1), 802(P)(2), 802(M)(1), 802(M)(2), and 802(M)(3) are shown. Aggregate component 702(AC) also includes exposed factored types 702(FT-Ex) and encapsulated factored types 702(FT-En). Specifically, exposed factored types 702(FT-Ex)(1) and 702(FT-Ex)(2) and encapsulated factored types 702(FT-En)(1) and 702(FT-En)(2) are shown. Factored types 702(FT) also include factored type (FT) members 804.

In a described implementation, aggregate component 702(AC) includes at least one aggregate component member 802, which may be a method or a property for example. Aggregate component members 802 can therefore include aggregate

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

component methods 802(M) and aggregate component properties 802(P). These aggregate component members 802, such as aggregate component members 802(1) and 802(2), may be specific to the aggregate component 702(AC). In other words, some aggregate component members 802 like aggregate component members 802(1) and 802(2) that are on aggregate component 702(AC) may not rely on any factored types 702(FT). Alternatively, some aggregate component members 802 may be linked to underlying factored types 702(FT).

Factored types 702(FT) may be exposed factored types 702(FT-Ex) or encapsulated factored types 702(FT-En). Exposed factored types 702(FT-Ex) are factored types 702(FT) of a given aggregate component 702(AC) that may be accessible by or to other general component types 702(FT or AC) without using or going through individual aggregate component members 802 of the given aggregate component 702(AC). If a factored type 702(FT-Ex/En) is returned by an aggregate component member 802 (either a method or a property), then that factored type 702(FT-Ex) is exposed. Otherwise, that factored type 702(FT-En) is encapsulated.

In other words, an aggregate component member 802 can expose a factored type member 804, or an aggregate component member 802 can return a factored type instance. The latter can occur with exposed factored types 702(FT-Ex), and the former can occur with encapsulated factored types 702(FT-En). Encapsulated factored types 702(FT-En) are factored types 702(FT) of a given aggregate component 702(AC) that are contained within or internal to the given aggregate component 702(AC). Each factored type 702(FT) may include one or more members 804 (some of which are specifically indicated in FIG. 8) that are methods and/or properties.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

As illustrated, two method members 804 of encapsulated factored type 702(FT-En)(1) are exposed by aggregate component 702(AC) as method member 802(M)(1) and method member 802(M)(2). One method member 804 of encapsulated factored type 702(FT-En)(2) is exposed by aggregate component 702(AC) as method member 802(M)(3).

Exposed factored type 702(FT-Ex)(1) is itself exposed as a property member 802(P)(1) of aggregate component 702(AC). Similarly, exposed factored type 702(FT-Ex)(2) is also exposed as a property member 802(P)(2) of aggregate component 702(AC). As indicated, a factored type (FT) member 804 of exposed factored type 702(FT-Ex)(1) is exposed so as to be separately accessible (i.e., accessible without directly using an individual member 802 of aggregate component 702(AC)). Hence, a factored type member 804 of an exposed factored type 702(FT-Ex) can still be accessed even if it is not individually exposed by an aggregate component member 802 of aggregate component 702(AC).

Thus, the indicated member 804 of exposed factored type 702(FT-Ex)(1) is exposed so as to be accessible by component types 702 that are external to aggregate component 702(AC) without using a member 802 thereof. As indicated by the dashed lines emanating from exposed factored type 702(FT-Ex)(1), exposed factored types 702(FT-Ex) may be "handed off" for use by other component types 702 (especially by other factored types 702(FT)) that are unable to interact with aggregate components 702(AC) or that can better achieve their intended purpose using the handed-off exposed factored type 702(FT-Ex)(1) alone. It should be noted that the object (exposed factored type 702(FT-Ex)(1)) that is "handed off" is not a copy but rather an actual part of the exposing aggregate component 702(AC).

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

As a result, the operations on the handed off object affect aggregate component 702(AC).

Generally, if a factored type 702(FT) is encapsulated, it is not exposed to a consumer; instead, setting properties 802(P) or calling methods 802(M) on an aggregate component 702(AC) may cause factored types 702(FT) to be created, properties 804 to be set, or methods 804 to be called on the underlying factored type 702(FT). These members 802 and 804 may not have a one-to-one correspondence; for example, setting several properties 802(P) on an aggregate component 702(AC) may be cached in the aggregate component 702(AC). Subsequently calling a method 802(M) on the aggregate component 702(AC) may cause a factored type 702(FT) to be created using the previously-specified values of the several properties 802(P) as constructor arguments for the factored type 702(FT).

In a described implementation, aggregate components 702(AC) differ from more-traditional object-oriented components in at least two ways in addition to the exposure of exposed factored types 702(FT-Ex). First, an aggregate component 702(AC) does not necessarily expose every member 804 of all of its factored types 702(FT). In other words, aggregate components 702(AC) are not strictly devoted to an inheritance hierarchy. Second, aggregate components 702(AC) can have modes and thus may periodically have states that result in invalid operations.

As a guideline to component oriented design with respect to factors 506 (of FIG. 5), whether a factored type 702(FT) is exposed or encapsulated within an aggregate component 702(AC) may be based on one or more of a number of factors. First, a particular factored type 702(FT) is exposed as a property member 802(P) of a given aggregate component 702(AC) if the particular factored type

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

702(FT) includes functionality that is not exposed by the given aggregate component 702(AC). Second, a particular factored type 702(FT) is exposed as a property member 802(P) of a given aggregate component 702(AC) if other general component types 702 of the framework may benefit from a handoff for direct consumption of the particular factored type 702(FT). On the other hand, a particular factored type 702(FT) is not exposed (and is therefore encapsulated) when functionality of the particular factored type 702(FT) is completely exposed by a given aggregate component 702(AC) and when the particular factored type 702(FT) is not useful for handing off to other component types 702.

A developer can start with aggregate component 702(AC), especially for implementing simpler and/or core scenarios. When the developer wishes or needs to implement a more complex scenario, the developer can incrementally and gradually begin to directly access and use exposed factored types 702(FT-Ex), including low-level attributes thereof, over time. The original code that relied on the simpler aggregate component 702(AC) does not need to be jettisoned and replaced with more complicated coding that relies solely on factored types 702(FT). The two-layers of the API framework can be used in varying proportions and can co-exist simultaneously.

Designing a two-layer API framework can be accomplished using the following exemplary technique that is described in ten phases: First, a set of core scenarios for a particular feature area is selected. Second, sample codes showing the preferred lines of code for the selected core scenarios are written. Third, aggregate components are derived with the appropriate methods, defaults, abstractions, naming, etc. to support the code samples from the lines of code.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Fourth, the code samples from the second phase are refined as appropriate according to the derived aggregate components. Fifth, the refined code samples are evaluated for whether or not they are sufficiently simple. If not, the technique continues again at the third phase. If so, then at the sixth phase it is determined whether additional scenarios, usages, interactions with other components, and/or other requirements exist. Seventh, the API designer decides if any of the additional requirements discovered in the sixth phase can be added to the aggregate components without adding undue complexity to the selected core scenarios.

Eighth, if the additional requirements cannot be added to the aggregate components, an ideal factoring (e.g., based on object-oriented or other analytical methodologies) of a full set of functionality for the factored types is defined based on the seventh phase. Ninth, it is determined how and whether the aggregate components encapsulate or expose the functionality from the factored types that are defined in the eighth phase. Tenth, the factored types are refined as appropriate to support the aggregate components as well as the additional requirements. Using this exemplary technique, a two-layer API framework having aggregate components 702(AC) and factored types 702(FT) may be designed.

FIG. 9 illustrates an exemplary aggregate component 702(AC) and associated APIs 902, 802(P), 802(M), and 904 that can support a create-set-call usage pattern. The following exemplary API groups are illustrated: constructors 902, properties 802(P), methods 802(M), and events 904. With a create, set, call usage pattern, an instance of aggregate component 702(AC) is initially created by a developer with reliance on default (e.g., parameter-less) constructors 902.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Secondly, the developer sets any properties 802(P) for which the default values are inappropriate and/or non-preferred for the intended use of the object. Thirdly, desired methods 802(M) are called by the developer. Callbacks are then implemented in terms of events 904.

Customizable Defaults

Customizable defaults relates to having defaults whenever practicable for at least aggregate components. When designing an API for example with multiple code samples corresponding to multiple languages, an identical value that is passed in each of the code samples can instead be set as a default for the aggregate component. The customizable defaults may be changed by setting one or more properties on the aggregate component.

Many developers prefer to code by trial and error as opposed to taking the time to read the documentation and fully understand a feature area prior to beginning a project. This is particularly true for novice and occasional developers, such as those that code with VB. These developers often try to experiment with an API to discover what it does and how it works, and then they adjust their code slowly and incrementally until the API implementation achieves their goal. The popularity of the editing and continuing approach to development is a manifestation of this preference.

Some API designs lend themselves to "coding by experimentation" and some do not. There are multiple aspects that affect the level of success a developer is likely to have when using a coding by experimentation approach. These aspects include: (i) how easy it is to locate the right API for the task at hand; (ii) how easy it is to start using an API, regardless of whether it (initially) does what the developer wants it to do or not; (iii) how easy it is to discover what

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

the points of customization are for an API; (iv) how easy it is to discover the correct customization for a given scenario; and (v) so forth.

In a described implementation, APIs are designed to require little if any initialization (e.g., a minimal amount of initialization). For example, an API can be designed so that a default constructor or a constructor with one simple parameter is sufficient to start working with a type. When initialization is necessary, an exception that results from not performing the initialization clearly explain what needs to be done and/or changed in order to remove or prevent the exception. For example, an exception may stipulate what or which property needs to be set.

By way of example but not limitation, a rule of thumb is that the simplest constructor has three or fewer parameters (with an upper limit of five). In addition, the simplest constructors should avoid complex types as any of the parameters, where complex types may be other factored types or aggregate components. Another rule of thumb is that the simplest constructors rely on primitive types like enumerations, strings, integers, and so forth. Types may also implement more complex constructor overloads to support more complex scenarios.

In short, an API's customizability can be simplified by providing properties with good defaults for all customization points. (However, developers should generally be able to add new code to the existing code when customizing their scenarios; rewriting the entire code from scratch using a different API should be optional.) For example, a system messaging queue aggregate component enables the sending of messages after passing a path string to the constructor and calling a

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

send method. Message properties, such as message priority and encryption algorithms, can be customized by adding code to the core scenario.

Self Documenting Object Model

Self documenting object model relates to designing an API framework in which a developer can look at objects and members thereof to learn about them as well as be able to use them. For example, names can be based on how a type is expected to be used instead of devotion to an inheritance hierarchy that many developers do not wish to study. In short, a self documenting object model facilitates discoverability by would-be developers.

As noted above, some developers prefer to code by trial and error and resort to reading documentation only when their intuition fails to implement their intended scenario. Thus, a self documenting object model should avoid requiring that developers read documentation every time they want to perform even simple tasks. An exemplary set of principles and practices to help in producing intuitive APIs that are relatively self documenting for a described implementation are presented below. Any one or more of them may be utilized in a given API and/or API design implementation.

Naming

A first guiding principle is to reserve simple and intuitive names for types that users need to use (e.g., instantiate) in the most common scenarios. Designers often "squander" the best names for abstractions, with which most users do not have to be concerned. For example, naming an abstract base class "File" and then providing a concrete type "XYZFile" works well if the expectation is that all users will have to understand the inheritance hierarchy before they can start using the APIs. However, if users do not understand the hierarchy, the first thing they will

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

likely try to use, most often unsuccessfully, is the "File" type. More specifically, the most common or expected names are reserved for aggregate components targeting the top core scenarios with less common or familiar names being used on concepts and abstractions.

A second guiding principle is to use descriptive identifier names that clearly state what each method does and what each type and parameter represents. For example, API designers should not hesitate to be rather verbose when choosing identifier names. For instance, "EventLog.DeleteEventSource(string source, string machineName)" may be seen as rather verbose, but it arguably has a net positive usability value. Moreover, type and parameter names state what a type or a parameter represents, instead of what it does. Method names state what the method does. Of course, accurate verbose method names are easier for methods that have simple and clear semantics, which is another reason why avoiding complex semantics is a good general design principle to follow.

A guiding design practice is to include a discussion about naming choices as a significant part of API specification reviews and/or tests. Exemplary considerations and questions include: What are the types most scenarios start with? What are the names most people think of first when trying to implement a given scenario? Are the names of the common types what users think of first? For example, since "File" is the name most people think of when dealing with file I/O scenarios, the aggregate component for accessing files can be named "File". Additionally, the most commonly used methods of the most commonly used types and their parameters are reviewed and tested. For example, can anybody familiar with the technology, but not the specific API design under consideration, recognize and call those methods quickly, correctly, and easily?

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Exceptions

As indicated above, exceptions can facilitate self-documenting APIs. In other words, APIs should lead the user to do the next required thing, and exceptions are capable of and good for communicating what is required next. For example, the following sample code throws an exception with a message "The 'FileName' property needs to be set before attempting to open the 'FileObject'.":

```
'VB
'Instantiate
Dim File As New FileObject()
'The file name is not set.

File.Open()
```

Strong Typing

Another guiding principle for facilitating intuitive APIs is strong typing. For example, calling "Customer.Name" is easier than calling "Customer.Properties['Name']". Furthermore, having such a "Name" property return the name as a string is more usable than if the property returned an object.

There are cases where property bags with a string based accessor, late bind calls, and other not strongly types APIs are desired, but they are relegated to rarity and are not the rule. Moreover, API designers can provide strongly typed helpers for the more common operations that the user performs on the non-strongly typed API layer. For example, a customer type may have a property bag, but it may additionally provide strongly typed APIs for more common properties like "name", "address", and so forth.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Vectoring toward Simplicity

Yet another guiding principle is to strive for simplicity, especially for core scenarios. Standard design methodologies are aimed at producing designs that are optimized for maintainability, such as by using abstractions. Consequently, modern design methodologies produce a lot of abstractions. An issue is that such design methodologies operate on an assumption that users of the resulting designs will become experts in that design before starting to implement even simple scenarios. However, that is often not the cases in the real world.

In a described implementation, for at least simple scenarios, API designers ensure that object model hierarchies are sufficiently simple so that they can be used without having to understand how the entire feature area fits together or interoperates. A resulting well-designed API may require that the developer understand the core scenario being implemented, but it does not require a full understanding of the design of the library being used to implement it.

Generally, core scenario APIs are directed or correspond to physical or well-known logical parts of the system instead of abstractions. Types that correspond to abstractions are usually difficult to use without understanding how all the parts of the feature area fit together and interoperate; they are therefore more relevant when cross-feature integration is required.

Another guiding practice is to use standard design methodologies (e.g., UML) when designing internal architectures and some of the factored types, but not when designing the APIs for the core or common scenarios (e.g., those with aggregate components). When designing aggregate components for core scenarios, scenario driven design together with prototyping, usability studies, and iteration (as described herein above) is employed instead.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Clean Namespaces

Yet another guiding principle is that types that are (very) rarely used are placed in sub-namespaces to avoid clutter of the main namespaces. For example, the following two groups of types may be separated from their main namespaces: permission types and design types. For instance, permission types can reside in a ".Permissions" sub-namespace, and design-time-only types can reside in a ".Design" sub-namespace.

The actions, aspects, features, components, etc. of FIGS. 1-9 are illustrated in diagrams that are divided into multiple blocks. However, the order, interconnections, interrelationships, layout, etc. in which FIGS. 1-9 are described and/or shown is not intended to be construed as a limitation, and any number of the blocks can be modified, combined, rearranged, augmented, omitted, etc. in any manner to implement one or more systems, methods, devices, procedures, media, APIs, apparatuses, arrangements, etc. for designing APIs. Furthermore, although the description herein includes references to specific implementations (and the exemplary operating environment of FIG. 10), the illustrated and/or described implementations can be implemented in any suitable hardware, software, firmware, or combination thereof and using any suitable software architecture(s), coding language(s), scenario definitions(s), usability study format(s), and so forth.

Exemplary Operating Environment for Computer or Other Device

FIG. 10 illustrates an exemplary computing (or general device) operating environment 1000 that is capable of (fully or partially) implementing at least one system, device, apparatus, component, arrangement, protocol, approach, method, procedure, media, API, some combination thereof, etc. for designing APIs as

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

described herein. Operating environment 1000 may be utilized in the computer and network architectures described below.

Exemplary operating environment 1000 is only one example of an environment and is not intended to suggest any limitation as to the scope of use or functionality of the applicable device (including computer, network node, entertainment device, mobile appliance, general electronic device, etc.) architectures. Neither should operating environment 1000 (or the devices thereof) be interpreted as having any dependency or requirement relating to any one or to any combination of components as illustrated in FIG. 10.

Additionally, designing APIs and/or the APIs resulting therefrom may be implemented with numerous other general purpose or special purpose device (including computing system) environments or configurations. Examples of well known devices, systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, thin clients, thick clients, personal digital assistants (PDAs) or mobile telephones, watches, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set-top boxes, programmable consumer electronics, video game machines, game consoles, portable or handheld gaming units, network PCs, minicomputers, mainframe computers, network nodes, distributed or multi-processing computing environments that include any of the above systems or devices, some combination thereof, and so forth.

Implementations for the design of APIs and/or the APIs resulting therefrom may be described in the general context of processor-executable instructions. Generally, processor-executable instructions include routines, programs, modules, protocols, objects, interfaces, components, data structures, etc. that perform and/or

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

enable particular tasks and/or implement particular abstract data types. Designing APIs and/or the APIs resulting therefrom, as described in certain implementations herein, may also be practiced and/or present in distributed processing environments where tasks are performed by remotely-linked processing devices that are connected through a communications link and/or network. Especially but not exclusively in a distributed computing environment, processor-executable instructions may be located in separate storage media, executed by different processors, and/or propagated over transmission media.

Exemplary operating environment 1000 includes a general-purpose computing device in the form of a computer 1002, which may comprise any (e.g., electronic) device with computing/processing capabilities. The components of computer 1002 may include, but are not limited to, one or more processors or processing units 1004, a system memory 1006, and a system bus 1008 that couples various system components including processor 1004 to system memory 1006.

Processors 1004 are not limited by the materials from which they are formed or the processing mechanisms employed therein. For example, processors 1004 may be comprised of semiconductor(s) and/or transistors (e.g., electronic integrated circuits (ICs)). In such a context, processor-executable instructions may be electronically-executable instructions. Alternatively, the mechanisms of or for processors 1004, and thus of or for computer 1002, may include, but are not limited to, quantum computing, optical computing, mechanical computing (e.g., using nanotechnology), and so forth.

System bus 1008 represents one or more of any of many types of wired or wireless bus structures, including a memory bus or memory controller, a point-to-point connection, a switching fabric, a peripheral bus, an accelerated graphics port,

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

and a processor or local bus using any of a variety of bus architectures. By way of example, such architectures may 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, a Peripheral Component Interconnects (PCI) bus also known as a Mezzanine bus, some combination thereof, and so forth.

Computer 1002 typically includes a variety of processor-accessible media. Such media may be any available media that is accessible by computer 1002 or another (e.g., electronic) device, and it includes both volatile and non-volatile media, removable and non-removable media, and storage and transmission media.

System memory 1006 includes processor-accessible storage media in the form of volatile memory, such as random access memory (RAM) 1040, and/or non-volatile memory, such as read only memory (ROM) 1012. A basic input/output system (BIOS) 1014, containing the basic routines that help to transfer information between elements within computer 1002, such as during start-up, is typically stored in ROM 1012. RAM 1010 typically contains data and/or program modules/instructions that are immediately accessible to and/or being presently operated on by processing unit 1004.

Computer 1002 may also include other removable/non-removable and/or volatile/non-volatile storage media. By way of example, FIG. 10 illustrates a hard disk drive or disk drive array 1016 for reading from and writing to a (typically) non-removable, non-volatile magnetic media (not separately shown); a magnetic disk drive 1018 for reading from and writing to a (typically) removable, non-volatile magnetic disk 1020 (e.g., a "floppy disk"); and an optical disk drive 1022 for reading from and/or writing to a (typically) removable, non-volatile optical

Eliminado: MS1-1748US PATAPP (J).DOC
Insertado: MS1-1748US PATAPP (J).DOC
Eliminado: MS1-1748US.PATAPP

51

disk 1024 such as a CD, DVD, or other optical media. Hard disk drive 1016, magnetic disk drive 1018, and optical disk drive 1022 are each connected to system bus 1008 by one or more storage media interfaces 1026. Alternatively, hard disk drive 1016, magnetic disk drive 1018, and optical disk drive 1022 may be connected to system bus 1008 by one or more other separate or combined interfaces (not shown).

The disk drives and their associated processor-accessible media provide non-volatile storage of processor-executable instructions, such as data structures, program modules, and other data for computer 1002. Although exemplary computer 1002 illustrates a hard disk 1016, a removable magnetic disk 1020, and a removable optical disk 1024, it is to be appreciated that other types of processor-accessible media may store instructions that are accessible by a device, such as magnetic cassettes or other magnetic storage devices, flash memory, compact disks (CDs), digital versatile disks (DVDs) or other optical storage, RAM, ROM, electrically-erasable programmable read-only memories (EEPROM), and so forth. Such media may also include so-called special purpose or hard-wired IC chips. In other words, any processor-accessible media may be utilized to realize the storage media of the exemplary operating environment 1000.

Any number of program modules (or other units or sets of instructions/code, including an API framework and/or objects based thereon) may be stored on hard disk 1016, magnetic disk 1020, optical disk 1024, ROM 1012, and/or RAM 1040, including by way of general example, an operating system 1028, one or more application programs 1030, other program modules 1032, and program data 1034.

Eliminado: MSI-1748US PATAPP (2).DOC

Insertado: MSI-1748US PATAPP (2).DOC

Eliminado: MSI-1748US.PATAPP

A user may enter commands and/or information into computer 1002 via input devices such as a keyboard 1036 and a pointing device 1038 (e.g., a "mouse"). Other input devices 1040 (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 processing unit 1004 via input/output interfaces 1042 that are coupled to system bus 1008. However, input devices and/or output devices may instead be connected by other interface and bus structures, such as a parallel port, a game port, a universal serial bus (USB) port, an infrared port, an IEEE 1394 ("Firewire") interface, an IEEE 802.11 wireless interface, a Bluetooth® wireless interface, and so forth.

A monitor/view screen 1044 or other type of display device may also be connected to system bus 1008 via an interface, such as a video adapter 1046. Video adapter 1046 (or another component) may be or may include a graphics card for processing graphics-intensive calculations and for handling demanding display requirements. Typically, a graphics card includes a graphics processing unit (GPU), video RAM (VRAM), etc. to facilitate the expeditious display of graphics and performance of graphics operations. In addition to monitor 1044, other output peripheral devices may include components such as speakers (not shown) and a printer 1048, which may be connected to computer 1002 via input/output interfaces 1042.

Computer 1002 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computing device 1050. By way of example, remote computing device 1050 may be a personal computer, a portable computer (e.g., laptop computer, tablet computer, PDA, mobile station, etc.), a palm or pocket-sized computer, a watch, a gaming device, a

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

server, a router, a network computer, a peer device, another network node, or another device type as listed above, and so forth. However, remote computing device 1050 is illustrated as a portable computer that may include many or all of the elements and features described herein with respect to computer 1002.

Logical connections between computer 1002 and remote computer 1050 are depicted as a local area network (LAN) 1052 and a general wide area network (WAN) 1054. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, the Internet, fixed and mobile telephone networks, ad-hoc and infrastructure wireless networks, other wireless networks, gaming networks, some combination thereof, and so forth. Such networks and communications connections are examples of transmission media.

When implemented in a LAN networking environment, computer 1002 is usually connected to LAN 1052 via a network interface or adapter 1056. When implemented in a WAN networking environment, computer 1002 typically includes a modem 1058 or other component for establishing communications over WAN 1054. Modem 1058, which may be internal or external to computer 1002, may be connected to system bus 1008 via input/output interfaces 1042 or any other appropriate mechanism(s). It is to be appreciated that the illustrated network connections are exemplary and that other manners for establishing communication link(s) between computers 1002 and 1050 may be employed.

In a networked environment, such as that illustrated with operating environment 1000, program modules or other instructions that are depicted relative to computer 1002, or portions thereof, may be fully or partially stored in a remote media storage device. By way of example, remote application programs 1060 reside on a memory component of remote computer 1050 but may be usable

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

or otherwise accessible via computer 1002. Also, for purposes of illustration, application programs 1030 and other processor-executable instructions such as operating system 1028 are illustrated herein as discrete blocks, but it is recognized that such programs, components, and other instructions reside at various times in different storage components of computing device 1002 (and/or remote computing device 1050) and are executed by processor(s) 1004 of computer 1002 (and/or those of remote computing device 1050).

Although systems, media, devices, methods, procedures, apparatuses, techniques, APIs, schemes, approaches, procedures, arrangements, and other implementations have been described in language specific to structural, logical, algorithmic, functional, and action-based features and/or diagrams, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or diagrams described. Rather, the specific features and diagrams are disclosed as exemplary forms of implementing the claimed invention.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

CLAIMS

1. A method for designing an application programming interface (API), the method comprising:

preparing a plurality of code samples for a core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages; and

deriving the API from the core scenario responsive to the plurality of code samples.

2. The method as recited in claim 1, further comprising:

determining, by an API designer, if the derived API is too complex.

3. The method as recited in claim 2, further comprising:

if the derived API is determined to be too complex, then

refining, by the API designer, the derived API to produce a refined API.

4. The method as recited in claim 3, further comprising:

determining, by the API designer, if the refined API is too complex.

5. The method as recited in claim 1, further comprising:

performing one or more usability studies on the API utilizing a plurality of developers.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

6. The method as recited in claim 5, wherein the performing comprises:

performing the one or more usability studies on the API utilizing the plurality of developers wherein the plurality of developers are typical for the plurality of programming languages.

7. The method as recited in claim 5, further comprising:

ascertaining whether the plurality of developers are able to use the API without significant problems.

8. The method as recited in claim 7, further comprising:

if the plurality of developers are not ascertained to be able to use the API without significant problems, then revising the API.

9. The method as recited in claim 8, wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies.

10. The method as recited in claim 1, wherein the deriving comprises:

deriving the API to support the plurality of code samples that correspond respectively to the plurality of programming languages.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

11. The method as recited in claim 1, wherein the deriving comprises:

gleaning language-specific mandates from the plurality of code samples; and

incorporating the language-specific mandates into the API.

12. The method as recited in claim 1, wherein the deriving comprises:

gleaning language-inspired developer expectations from the plurality of code samples; and

incorporating the language-inspired developer expectations into the API.

13. The method as recited in claim 1, wherein the deriving comprises:

gleaning commonalities from the plurality of code samples; and

incorporating the commonalities into the API.

14. The method as recited in claim 1, wherein the deriving comprises:

deriving the API to have an aggregate component that ties a plurality of lower-level factored types together to support the core scenario.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

15. A method for designing an application programming interface (API), the method comprising:

selecting a core scenario for a feature area;

writing at least one code sample for the core scenario; and

deriving an API for the core scenario responsive to the at least one code sample.

16. The method as recited in claim 15, wherein the selecting comprises:

selecting a plurality of core scenarios for the feature area.

17. The method as recited in claim 16, further comprising:

repeating the writing and the deriving for each core scenario of the plurality of core scenarios that are selected for the feature area.

18. The method as recited in claim 15, wherein the writing comprises:

writing a plurality of code samples for the core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages.

19. The method as recited in claim 18, wherein the deriving comprises:

deriving the API for the core scenario responsive to the plurality of code samples.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

20. The method as recited in claim 15, further comprising:

performing one or more usability studies on the API utilizing a plurality of developers;

ascertaining whether the plurality of developers are able to use the API without significant problems; and

if the plurality of developers are not ascertained to be able to use the API without significant problems, then revising the API.

21. The method as recited in claim 20, wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies to produce a revised API.

22. The method as recited in claim 21, further comprising:

repeating the performing and the ascertaining with respect to the revised API.

23. The method as recited in claim 15, wherein the deriving comprises:

deriving the API to support the at least one code sample written for the core scenario by producing a two-layer API that includes an aggregate component and a plurality of underlying factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-.7481S PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

24. The method as recited in claim 15, wherein the deriving comprises:

gleaning one or more language-specific mandates from the at least one code sample; and

incorporating the one or more language-specific mandates into the API.

25. The method as recited in claim 15, wherein the deriving comprises:

encapsulating a particular factored type into an aggregate component that is associated with the core scenario if all members of the particular factored type are exposed by the aggregate component.

26. The method as recited in claim 15, wherein the deriving comprises:

encapsulating a particular factored type into an aggregate component that is associated with the core scenario if the particular factored type is independently uninteresting to other component types.

27. The method as recited in claim 15, wherein the deriving comprises:

exposing a particular factored type from an aggregate component that is associated with the core scenario if at least one member of the particular factored type is not exposed by the aggregate component.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

28. The method as recited in claim 15, wherein the deriving comprises:

exposing a particular factored type from an aggregate component that is associated with the core scenario if the particular factored type can be beneficially used independently of the aggregate component by another component type.

29. The method as recited in claim 15, wherein the deriving comprises:

producing a two-layer framework that includes component types targeting a relatively higher level of abstraction and component types targeting a relatively lower level of abstraction.

30. The method as recited in claim 29, wherein the component types targeting the relatively higher level of abstraction are directed to core scenarios.

31. The method as recited in claim 29, wherein the component types targeting the relatively lower level of abstraction provide a relatively greater amount of control to developers as compared to the component types targeting the relatively higher level of abstraction.

32. The method as recited in claim 15, wherein the deriving comprises:

deriving the API so as to enable a developer to implement a create-set-call usage pattern for the core scenario.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

33. The method as recited in claim 32, wherein the deriving comprises:

producing the API with pre-selected parameters that are appropriate for the core scenario.

34. A method for designing an application programming interface (API), the method comprising:

deriving an API for a scenario responsive to at least one code sample written with regard to the scenario;

performing one or more usability studies on the API utilizing a plurality of developers; and

revising the API based on the one or more usability studies.

35. The method as recited in claim 34, further comprising:

writing a plurality of code samples with regard to the scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages;

wherein the deriving comprises:

deriving the API for the scenario responsive to the plurality of code samples.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

36. The method as recited in claim 34, further comprising, prior to the performing one or more usability studies on the API:

determining, by an API designer, if the derived API is too complex;

if the derived API is determined to be too complex, then

refining, by the API designer, the derived API to produce a refined API; and

determining, by the API designer, if the refined API is too complex.

37. The method as recited in claim 34, further comprising:

ascertaining whether the plurality of developers are able to use the API without significant problems; and

when the plurality of developers are not ascertained to be able to use the API without significant problems, then implementing the revising;

wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies to produce a revised API.

38. The method as recited in claim 37, further comprising:

repeating at least the performing and the ascertaining with respect to the revised API.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

39. The method as recited in claim 37, wherein the ascertaining comprises:

ascertaining whether the plurality of developers are able to use the API without significant problems with regard to a desired level of usability for at least one targeted developer group, wherein the desired level of usability includes considerations with respect to (i) frequent and/or extensive reference to detailed API documentation by the plurality of developers, (ii) a failure of a majority of the plurality of developers to implement the scenario, and (iii) whether the plurality of developers take an approach that is significantly different from what is expected by an API designer.

40. The method as recited in claim 34, further comprising:

selecting a plurality of core scenarios for a feature area;

repeating the deriving, the performing, and the revising for each core scenario of the plurality of core scenarios.

41. The method as recited in claim 40, wherein the deriving comprises:

producing an aggregate component for each core scenario of the plurality of core scenarios.

42. The method as recited in claim 34, wherein the deriving comprises:

producing an aggregate component that has a respective relationship with each respective factored type of a plurality of factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

43. The method as recited in claim 42, wherein the producing comprises:

producing the aggregate component to support the scenario for which the at least one code sample is written.

44. The method as recited in claim 42, wherein the producing comprises:

producing the aggregate component to have an exposed relationship with at least one factored type of the plurality of factored types and an encapsulated relationship with at least one other factored type of the plurality of factored types.

45. The method as recited in claim 44, wherein the at least one other factored type of the plurality of factored types that has the encapsulated relationship with the aggregate component can be handed off by the aggregate component for direct interaction with another component type.

46. The method as recited in claim 42, wherein the plurality of factored types are designed using an object-oriented methodology.

Eliminado: MS1-1748US PATAPP (:).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

47. A method for designing an application programming interface (API), the method comprising:

preparing a plurality of code samples for a core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages;

deriving the API for the core scenario responsive to the plurality of code samples;

performing one or more usability studies on the API utilizing a plurality of developers; and

revising the API based on the one or more usability studies.

48. A method for designing an application programming interface (API), the method comprising:

writing at least one code sample for a scenario; and

deriving an API for the scenario responsive to the at least one code sample, the API including (i) an aggregate component that is adapted to facilitate implementation of the scenario and (ii) a plurality of factored types that provide underlying functionality for the aggregate component, the API enabling a gradual progression from using the aggregate component in simpler situations to using an increasing portion of the plurality of factored types in increasingly complex situations.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

49. A method for designing an application programming interface (API), the method comprising:

deriving at least one aggregate component to support at least one code sample for at least one scenario;

determining additional requirements with respect to the at least one scenario;

deciding if the additional requirements can be added to the at least one aggregate component without adding undue complexity to the at least one scenario; and

if not, defining a plurality of factored types responsive to the deciding.

50. The method as recited in claim 49, further comprising:

if so, refining the at least one aggregate component to incorporate the additional requirements.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

51. The method as recited in claim 49, further comprising:

selecting a plurality of core scenarios for a feature area, the plurality of core scenarios including the at least one scenario; and

writing a plurality of code samples showing preferred lines of code for the plurality of core scenarios, the plurality of code samples including the at least one code sample;

wherein the deriving comprises:

deriving a plurality of aggregate components, which include the at least one aggregate component, to support the plurality of code samples for the plurality of core scenarios.

52. The method as recited in claim 49, wherein the deriving comprises:

deriving the at least one aggregate component with appropriate methods, defaults, and abstractions to support the at least one code sample for the at least one scenario.

53. The method as recited in claim 49, further comprising:

refining the at least one code sample according to the at least one derived aggregate component; and

evaluating the refined at least one code sample with regard to simplicity; and

repeating the deriving if the refined at least one code sample fails to be sufficiently simple in the evaluating.

Eliminado: MS1-1746US PATAPP (2).DOC

Insertado: MS1-1746US PATAPP (2).DOC

Eliminado: MS1-1746US.PATAPP

54. The method as recited in claim 49, wherein the determining comprises:

determining additional requirements with respect to the at least one scenario, wherein the additional requirements include additional scenarios, additional usages, and additional interactions with other component types.

55. The method as recited in claim 49, wherein the deciding comprises:

considering whether adding the additional requirements to the at least one aggregate component hinders a create-set-call usage pattern.

56. The method as recited in claim 49, wherein the defining comprises:

defining the plurality of factored types responsive to the deciding with an ideal factoring of a full set of functionality.

57. The method as recited in claim 49, wherein the defining comprises:

defining the plurality of factored types responsive to the deciding using one or more object-oriented methodologies.

58. The method as recited in claim 49, further comprising:

determining whether the at least one aggregate component is to encapsulate or expose the functionality of each factored type of the plurality of factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

59. The method as recited in claim 49, further comprising:

refining the plurality of factored types to support the at least one aggregate component and the additional requirements.

Eliminado: MSI-1748US PATAPP (2).DOC

Insertado: MSI-1748US PATAPP (2).DOC

Eliminado: MSI-1748US.PATAPP

ABSTRACT

A first exemplary method implementation for designing an application programming interface (API) includes: preparing multiple code samples for a core scenario, each respective code sample of the multiple code samples corresponding to a respective programming language of multiple programming languages; and deriving the API from the core scenario responsive to the multiple code samples. A second exemplary method for designing an API includes: selecting a core scenario for a feature area; writing at least one code sample for the core scenario; and deriving an API for the core scenario responsive to the at least one code sample. A third exemplary method for designing an API includes: deriving an API for a scenario responsive to at least one code sample written with regard to the scenario; performing one or more usability studies on the API utilizing multiple developers; and revising the API based on the one or more usability studies.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

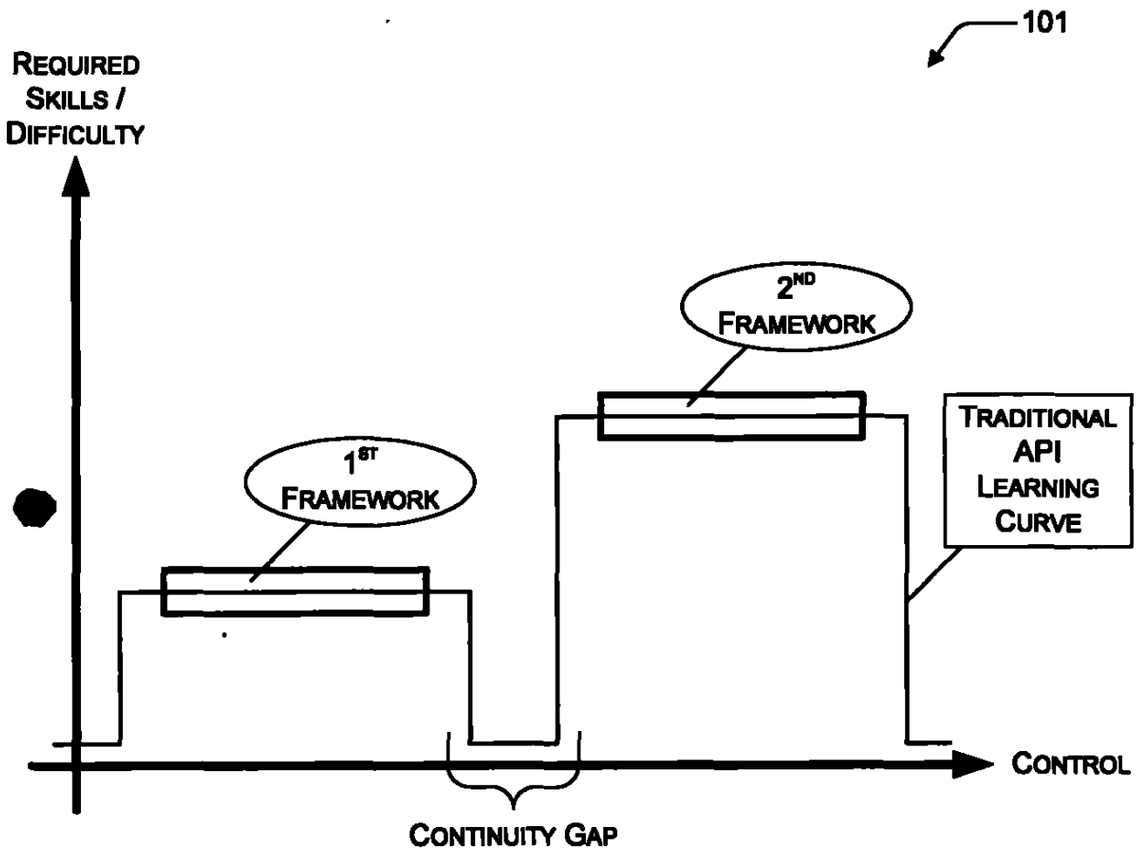


FIG. 1 Prior Art

72

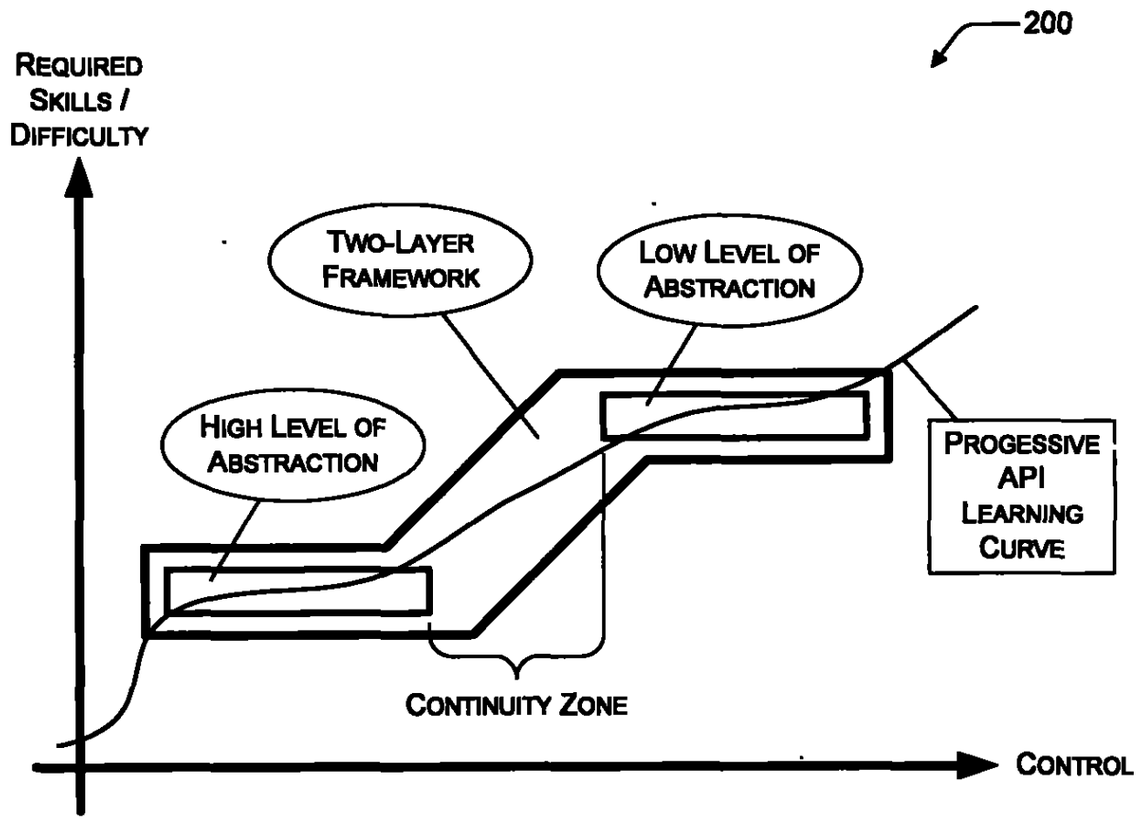

200
REQUIRED SKILLS / DIFFICULTY
TWO-LAYER FRAMEWORK
LOW LEVEL OF ABSTRACTION
HIGH LEVEL OF ABSTRACTION
PROGESSIVE API LEARNING CURVE
CONTINUITY ZONE
CONTROL

FIG. 2

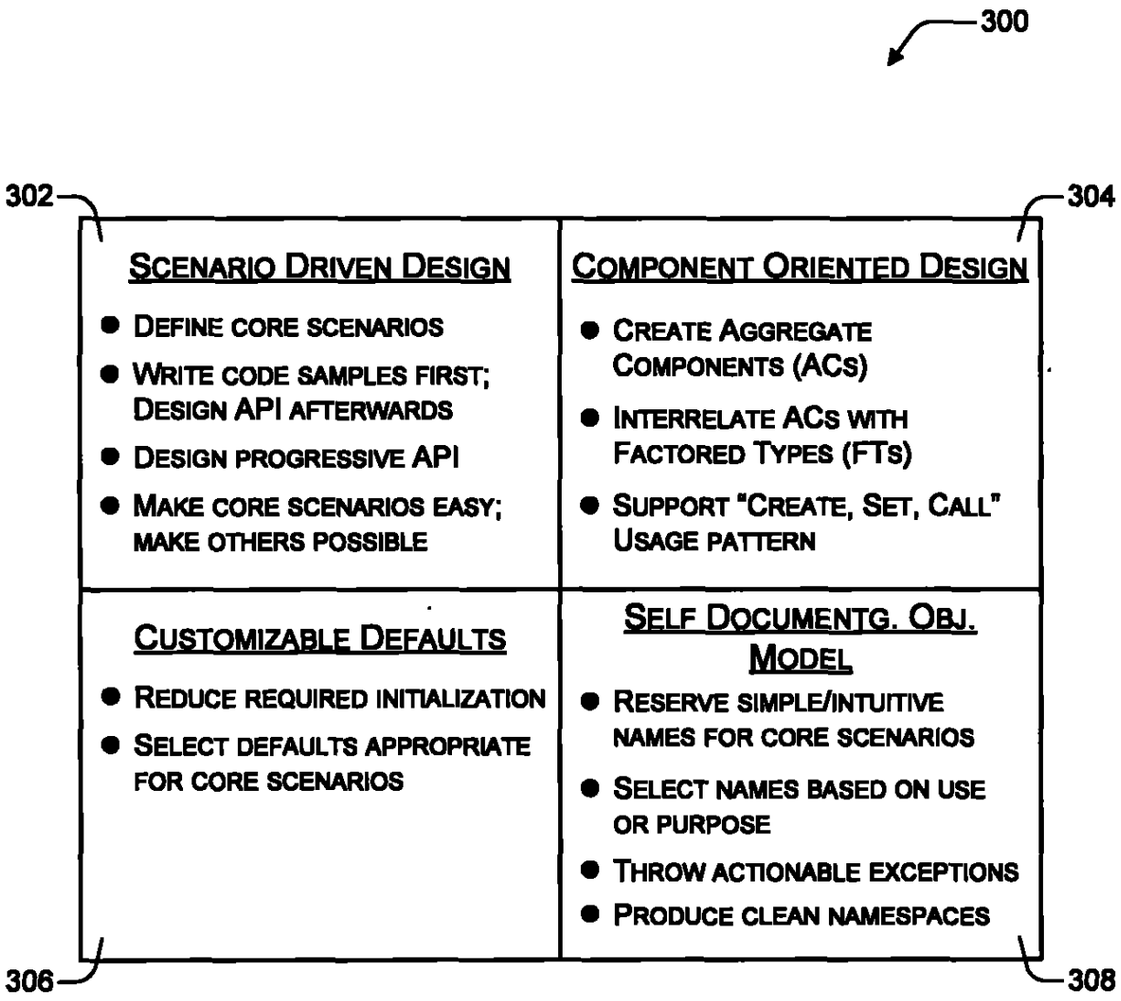

300
302
304
SCENARIO DRIVEN DESIGN
DEFINE CORE SCENARIOS
WRITE CODE SAMPLES FIRST; DESIGN API AFTERWARDS
DESIGN PROGRESSIVE API
MAKE CORE SCENARIOS EASY; MAKE OTHERS POSSIBLE
COMPONENT ORIENTED DESIGN
CREATE AGGREGATE COMPONENTS (ACS)
INTERRELATE ACS WITH FACTORED TYPES (FTS)
SUPPORT "CREATE, SET, CALL" USAGE PATTERN
CUSTOMIZABLE DEFAULTS
REDUCE REQUIRED INITIALIZATION
SELECT DEFAULTS APPROPRIATE FOR CORE SCENARIOS
SELF DOCUMENTG. OBJ. MODEL
RESERVE SIMPLE/INTUITIVE NAMES FOR CORE SCENARIOS
SELECT NAMES BASED ON USE OR PURPOSE
THROW ACTIONABLE EXCEPTIONS
PRODUCE CLEAN NAMESPACES
306
308

FIG. 3

74

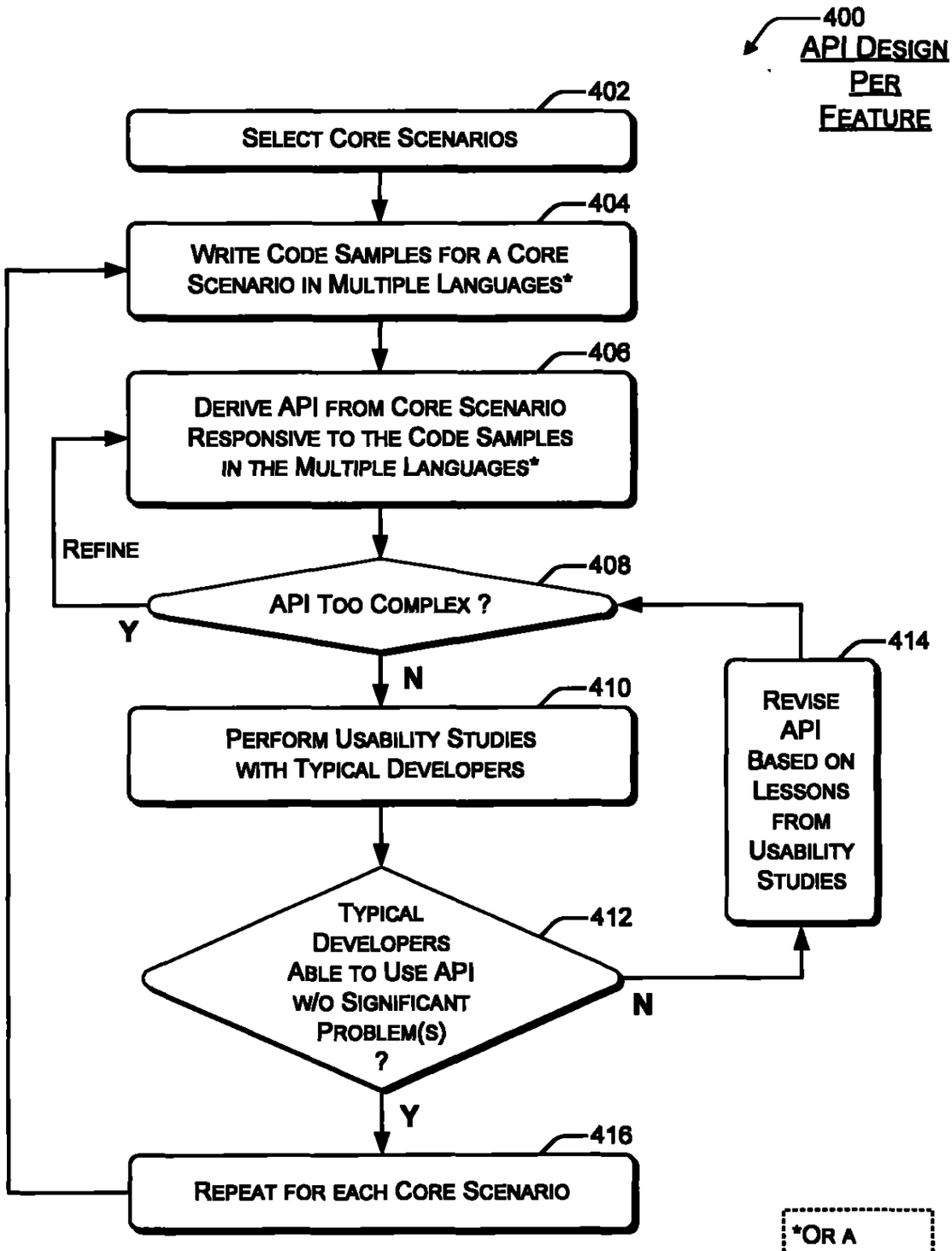

400
API DESIGN PER FEATURE
402
SELECT CORE SCENARIOS
404
WRITE CODE SAMPLES FOR A CORE SCENARIO IN MULTIPLE LANGUAGES*
406
DERIVE API FROM CORE SCENARIO RESPONSIVE TO THE CODE SAMPLES IN THE MULTIPLE LANGUAGES*
REFINE
408
API TOO COMPLEX ?
Y
N
410
PERFORM USABILITY STUDIES WITH TYPICAL DEVELOPERS
414
REVISE API BASED ON LESSONS FROM USABILITY STUDIES
412
TYPICAL DEVELOPERS ABLE TO USE API W/O SIGNIFICANT PROBLEM(S) ?
N
Y
416
REPEAT FOR EACH CORE SCENARIO
*OR A SINGLE LANGUAGE

FIG. 4

175

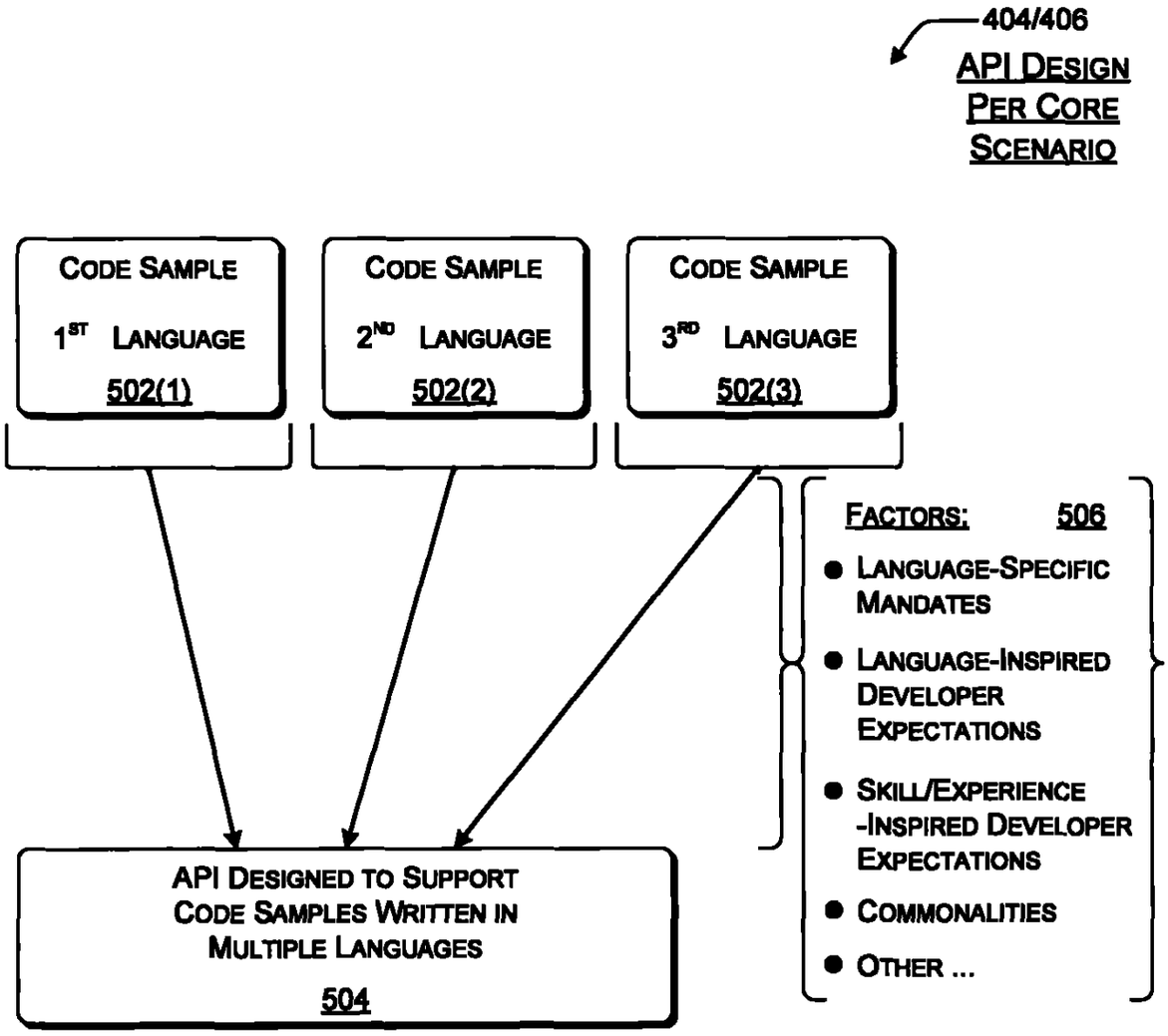

404/406
API DESIGN PER CORE SCENARIO
CODE SAMPLE 1ST LANGUAGE 502(1)
CODE SAMPLE 2ND LANGUAGE 502(2)
CODE SAMPLE 3RD LANGUAGE 502(3)
FACTORS: 506
LANGUAGE-SPECIFIC MANDATES
LANGUAGE-INSPIRED DEVELOPER EXPECTATIONS
SKILL/EXPERIENCE -INSPIRED DEVELOPER EXPECTATIONS
COMMONALITIES
OTHER ...
API DESIGNED TO SUPPORT CODE SAMPLES WRITTEN IN MULTIPLE LANGUAGES 504

FIG. 5

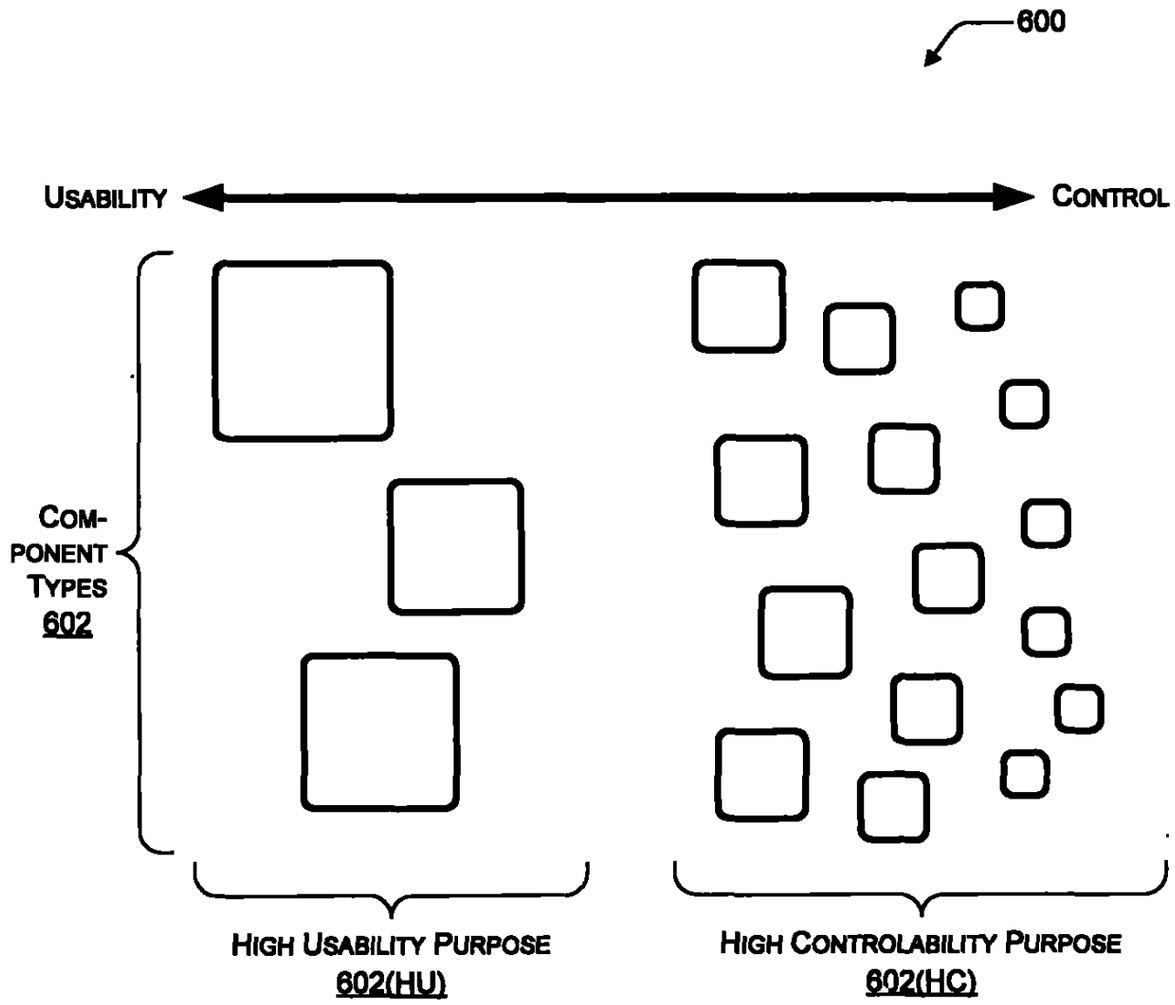

600
USABILITY
CONTROL
COM-
PONENT
TYPES
602
HIGH USABILITY PURPOSE
602(HU)
HIGH CONTROLABILITY PURPOSE
602(HC)

FIG. 6

77

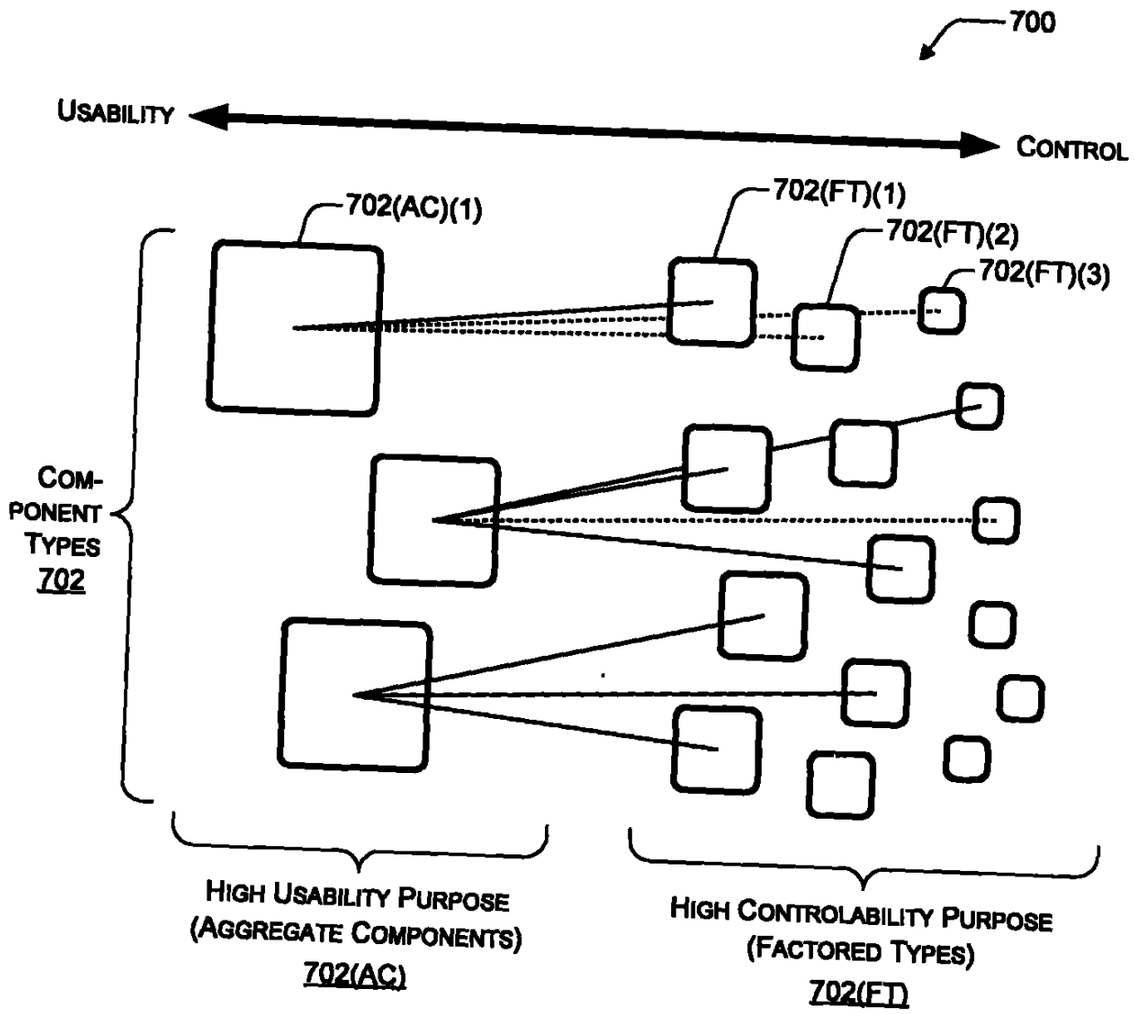
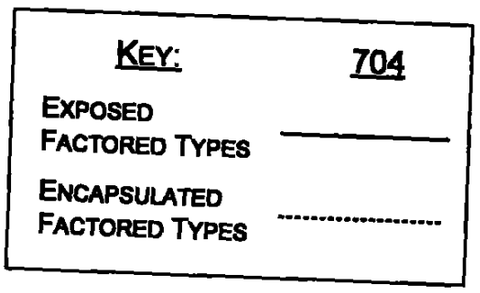

700
USABILITY
CONTROL
702(AC)(1)
702(FT)(1)
702(FT)(2)
702(FT)(3)
COM-
PONENT
TYPES
702
HIGH USABILITY PURPOSE
(AGGREGATE COMPONENTS)
702(AC)
HIGH CONTROLABILITY PURPOSE
(FACTORED TYPES)
702(FT)

FIG. 7


KEY:
704
EXPOSED
FACTORED TYPES
ENCAPSULATED
FACTORED TYPES

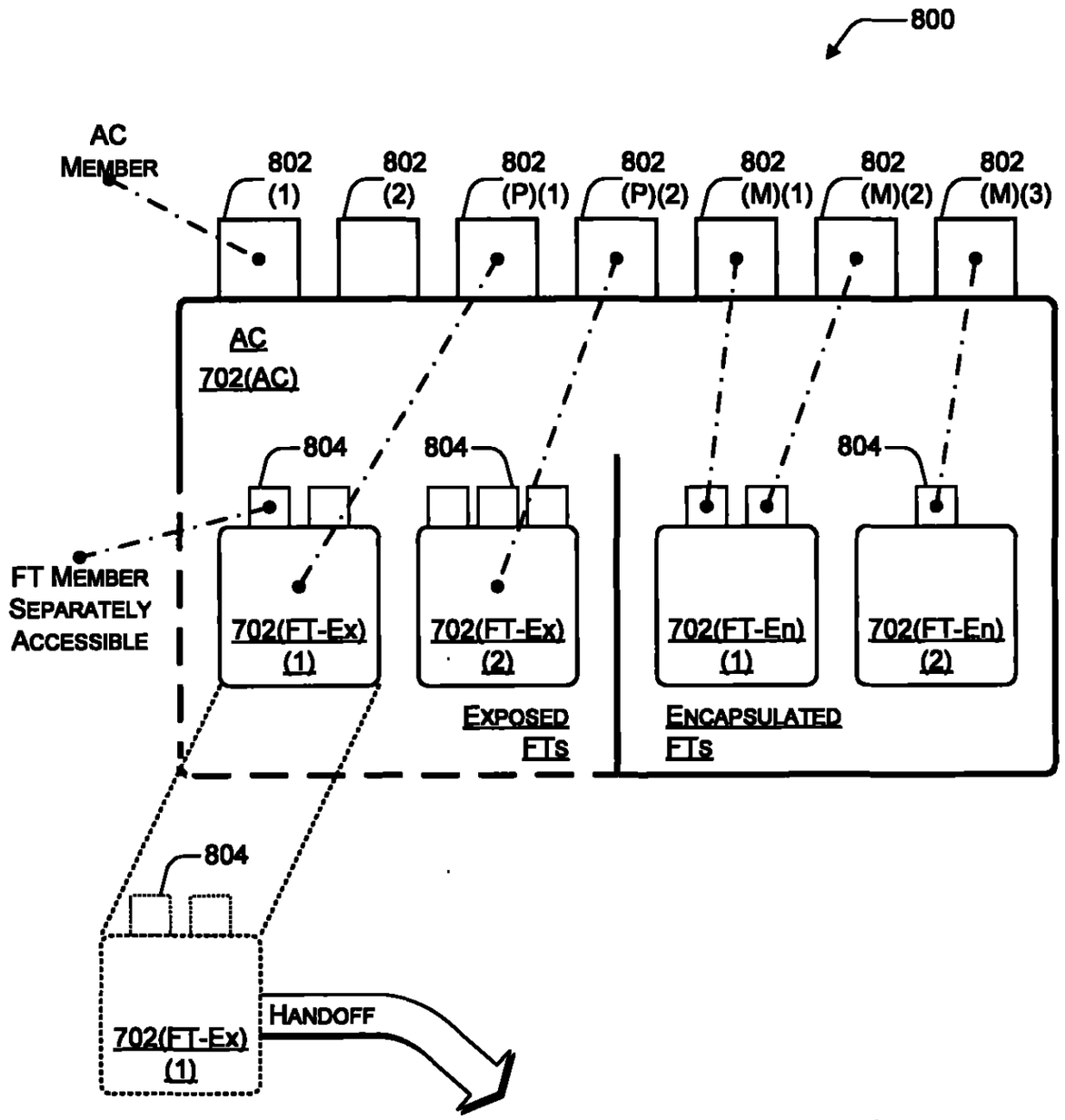

800
AC MEMBER
802 (1)
802 (2)
802 (P)(1)
802 (P)(2)
802 (M)(1)
802 (M)(2)
802 (M)(3)
AC 702(AC)
804
804
804
FT MEMBER SEPARATELY ACCESSIBLE
702(FT-Ex) (1)
702(FT-Ex) (2)
702(FT-En) (1)
702(FT-En) (2)
EXPOSED FTs
ENCAPSULATED FTs
804
HANDOFF
702(FT-Ex) (1)

FIG. 8

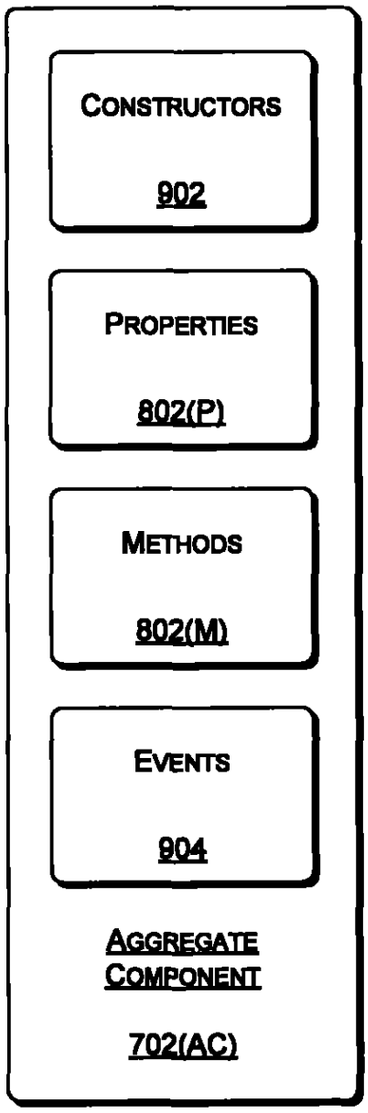
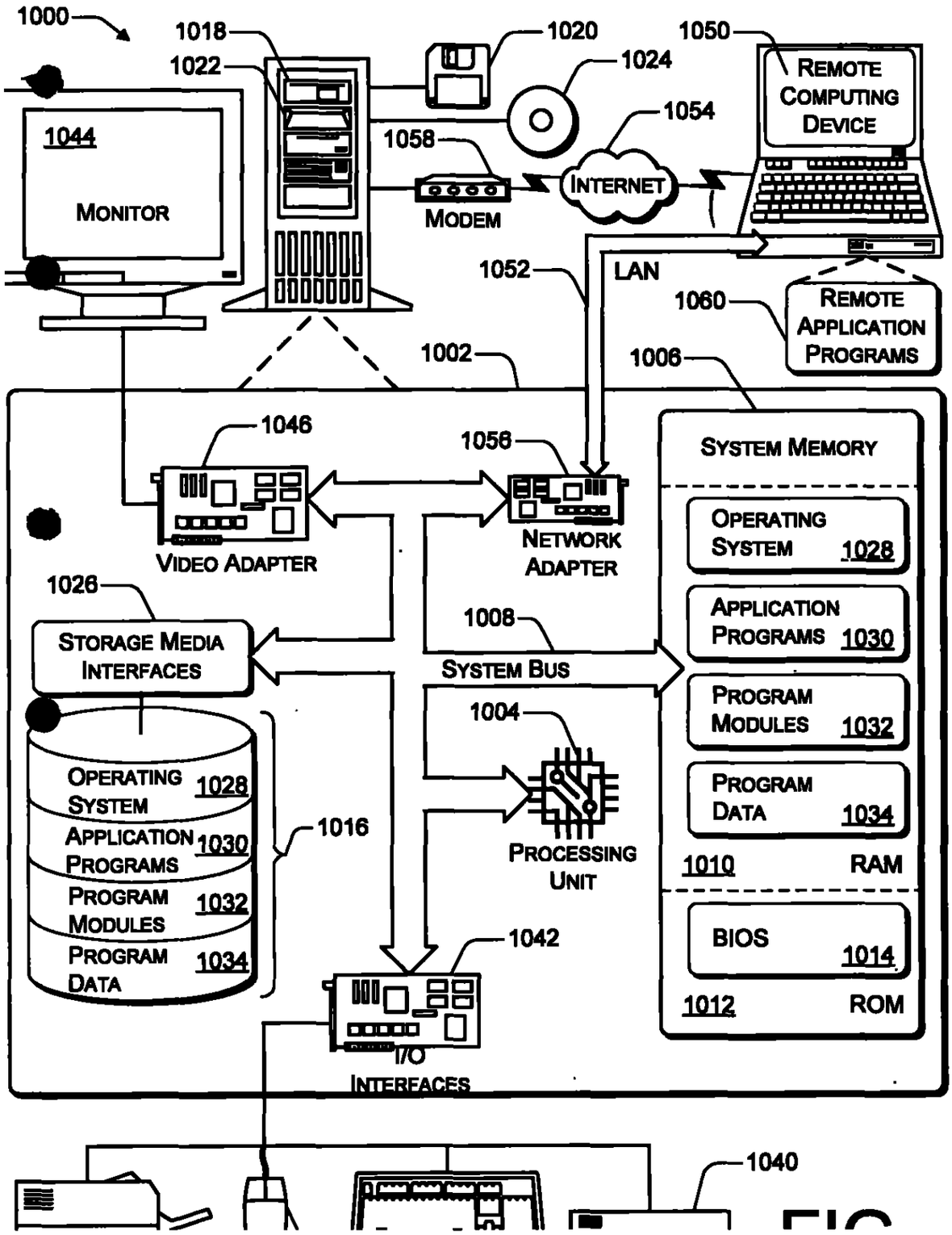

CONSTRUCTORS
902
PROPERTIES
802(P)
METHODS
802(M)
EVENTS
904
AGGREGATE COMPONENT
702(AC)

FIG. 9


1000
1018
1022
1020
1024
1050
REMOTE COMPUTING DEVICE
1044
MONITOR
1058
MODEM
1054
INTERNET
1052
LAN
1060
REMOTE APPLICATION PROGRAMS
1002
1006
1046
1056
SYSTEM MEMORY
VIDEO ADAPTER
NETWORK ADAPTER
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1026
STORAGE MEDIA INTERFACES
1008
SYSTEM BUS
1004
PROCESSING UNIT
1010 RAM
BIOS 1014
1012 ROM
1016
1042
I/O INTERFACES
1040

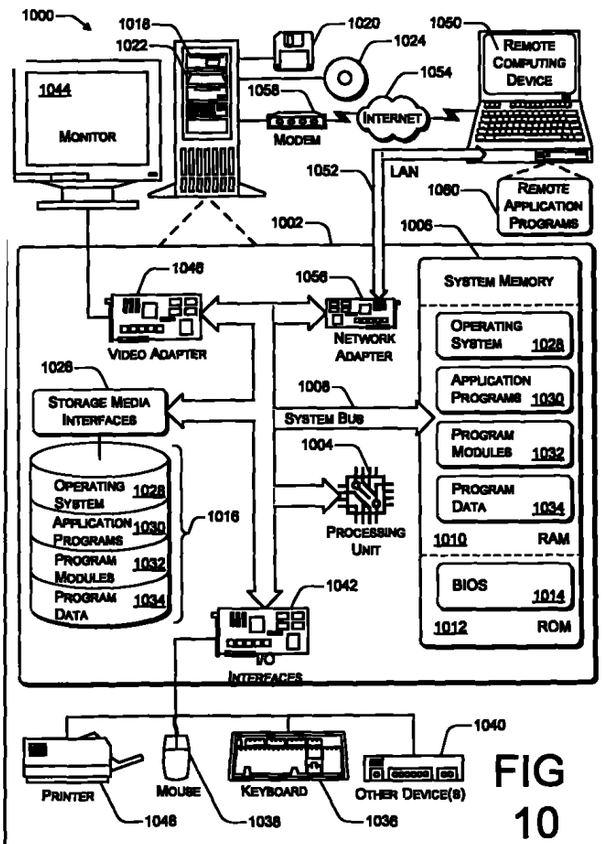

1000
1018
1022
1020
1024
1050
REMOTE COMPUTING DEVICE
1044
MONITOR
1058
MODEM
1054
INTERNET
1052
LAN
1080
REMOTE APPLICATION PROGRAMS
1002
1006
SYSTEM MEMORY
1046
VIDEO ADAPTER
1056
NETWORK ADAPTER
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1010 RAM
BIOS 1014
1012 ROM
1026
STORAGE MEDIA INTERFACES
1008
SYSTEM BUS
1004
PROCESSING UNIT
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1016
1042
I/O INTERFACES
1040
PRINTER
MOUSE
KEYBOARD
OTHER DEVICE(S)
1048
1038
1036

FIG 10

HOJA DE DETALLES

Información de la Solicitud

Número de Archivo MS: 305796.01

Título de la Invención: Diseño de Programación de Aplicaciones (APIs, por sus siglas en inglés: "*Design of Application Programming Interfaces*")

Información de la Prioridad:

País: Estados Unidos de América
Número de Serie: 10/692320
Fecha de Presentación de la Prioridad: 10/23/2003
Dependencias adicionales, si las hay: ver la familia de patente a continuación

Inventores:

Favor usar las direcciones de Microsoft Corporation como la dirección del (los) inventor(es) a continuación.

Inventores/Participantes		
Nombre del Inventor	**Tipo**	**Ciudadanía**
Anthony J. Moore	Inventor	Estados Unidos de América
Bradley M. Abrams	Inventor	Estados Unidos de América
Christopher L. Anderson	Inventor	Estados Unidos de América
Krzysztof J. Cwalina	Inventor	Estados Unidos de América
Michael J. Pizzo	Inventor	Estados Unidos de América
Robert A. Brigham	Inventor	Estados Unidos de América

Solicitante: Microsoft es el único solicitante a menos haya anotaciones a continuación respecto de co-cesionarios

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
Estados Unidos de América

Estado de Constitución: Washington

Documento de Solicitud:

Archivos Adjuntos		
Tipo de Documento	**Adjunto**	**Título del Documento**
Documento de Solicitud	305796.1 – Diseño del Programa de Aplicación	(en blanco)
Dibujos de la Solicitud	305796.1 – Figuras.vsd	

Figuras para publicación: __4___

Instrucciones Específicas

Por favor suministre una copia de los documentos de solicitud presentados para este caso, así como una copia de toda la correspondencia presentada ante y expedida por la oficina de patentes a través del proceso de esta solicitud. Si usted requiere cualquier documentación adicional en este momento, por favor háganoslo saber de inmediato. A falta de instrucciones, por favor lleve a cabo todas las acciones necesarias para mantener esta solicitud.. Esta solicitud no deberá ser abandonada ni se podrá permitir que se venza sin nuestro consentimiento previo.

Cuando reenvía un Acto Oficial emitido por la Oficina de Patentes, les solicitamos que nos envíen un borrador de respuesta apropiado, junto con nuestra revisión, ya que nosotros nos basamos en su experiencia y conocimiento, en relación con las leyes y reglamentos locales sobre patentes.

Gracias por su ayuda en relación con este asunto.. **Por favor enviarnos sin demora el acuse de recibo de nuestras instrucciones vía correo electrónico al remitente, con copia a msintl@microsoft.com.**

Información Adicional

La siguiente información se suministra como información adicional, para su uso, en el momento en que ustedes asignen sus recursos internos, y para ayudarles a entender la estrategia / relaciones que esta invención tiene, para con y dentro Microsoft. No tenemos instrucciones específicas en cuanto al uso de esta información.

Familia de la Patente: Lista de todas las demás aplicaciones que tienen una relación padre / hijo con las instrucciones que se envían.

Familia de Patente							
Archivo #	**Archivo Padre #**	**Archivo de Prioridad**	**Padre Adicional#**	**Tipo de Caso**	**Tipo de Asunto**	**Tipo de Archivo**	**País**
305796.01	305796.01			NPR	ORG	NAT	EEUU
305796.02	305796.01			NPR	ORG	EPC	EP
305796.03	305796.01			NPR	ORG	NAT	JP
305796.04	305796.01			NPR	ORG	NAT	CN
305796.05	305796.01			NPR	ORG	NAT	KR
305796.06	305796.01			NPR	ORG	NAT	IN
305796.07	305796.01			NPR	ORG	NAT	CA
305796.08	305796.01			NPR	ORG	NAT	AU
305796.09	305796.01			NPR	ORG	NAT	BR
305796.10	305796.01			NPR	ORG	NAT	RU
305796.11	305796.01			NPR	ORG	NAT	MX

305796.12	305796.01			NPR	ORG	NAT	ZA
305796.13	305796.01			NPR	ORG	NAT	TH
305796.14	305796.01			NPR	ORG	NAT	MY
305796.15	305796.01			NPR	ORG	NAT	TW
305796.16	305796.02			NPR	ORG	NAT	HK
305796.17	305796.01			NPR	ORG	NAT	NO
305796.18	305796.01			NPR	ORG	NAT	EG
305796.19	305796.01			NPR	ORG	NAT	PH
305796.20	305796.01			NPR	ORG	NAT	ID
305796.21	305796.01			NPR	ORG	NAT	IL
305796.22	305796.01			NPR	ORG	NAT	SG
305796.23	305796.01			NPR	ORG	NAT	CL
305796.24	305796.01			NPR	ORG	NAT	CO
305796.25	305796.01			NPR	ORG	NAT	NZ

Clase ROMP: Lista de clasificación interna de Microsoft. Esta se puede usar cuando asigne el trabajo dentro de su oficina.

ROMP	
Código ROMP	**Descripción ROMP**
18.1	Interfaces
6.5	Modelos de Programación/Patrones de Diseño

División Microsoft: Plataforma de Servidor

Productos: Productos conocidos referentes a esta invención

Productos Relacionados
Productos Seleccionados
Longhorn
- Whidbey

Firma de Abogados de los EE.UU.: Esta firma / apoderado manejó el borrador inicial de la solicitud.

Recurso OC		
Firma de Abogados	**Apoderado**	**Referencia OC**
Lee y Hayes	Keith Saunders	MS1-1748US

Conteo de Páginas: ***estos conteos son específicos de la paginación tal como fue impresa por el Asesor Legal de EE.UU.***
Especificación: 62
Dibujos: 10
Reivindicaciones: 16

Palabras Clave:

Palabras Clave	
Palabras Clave Relacionadas	**Tipo de Palabras Clave**
Tier 3	Country Tiers

86

EN LA OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

SOLICITUD DE PATENTE

DISEÑO DE INTERFACES DE PROGRAMACIÓN DE APLICACIÓN (API)

Inventor(es):

Krzysztof J. Cwalina

Bradley M. Abrams

Anthony J. Moore

Christopher L. Anderson

Michael J. Pizzo

Robert A. Brigham II

REGISTRO DE ABOGADO NO. **MS1-1748US**

87

OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

DEPARTAMENTO DE COMERCIO DE LOS ESTADOS UNIDOS
Oficina de Marcas y Patentes de los Estados Unidos
Dirección: COMISIONADO DE MARCAS Y PATENTES
P.O. BOX 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Número de Solicitud	Fecha de Presentación	Unidad técnica GPR	Tarifa Pres Rec.
10/692,320	10/23/2003	2127	0.00
Número Registro de Abogado	**Dibujos**	**Total reivindicaciones**	**Reivindicaciones independientes**
MS1-1748US	10	59	6

Confirmación No. 8610

Recibo de Presentación

22801
LEE & HAYES PLLC
421 W RIVERSIDE AVENUE SUITE 500
SPOKANE, WA 99201

Fecha de Correspondencia: 01/22/2004

88

Se acusa recibo de esta Solicitud de Patente regular. Será tenida en cuenta en su orden y usted será notificado de los resultados del examen. Asegúrese de suministrar el NÚMERO DE SOLICITUD U.S., FECHA DE PRESENTACIÓN, NOMBRE DEL SOLICITANTE, y TÍTULO DE LA INVENCIÓN cuando se le pregunte por esta solicitud. Las tarifas transferidas por cheque o letra de cambio están sujetas a recaudo. Por favor verifique la exactitud de los datos presentados en este recibo. **Si un error se anota en este Recibo de Presentación, por favor escriba a la oficina de Examen de Patente Inicial presentando el recibo de correcciones, facsimil número 703-746-9195. Por favor suministre una copia de este recibo de presentación con los cambios anotados en él.** Si usted recibe una "Nota para archivar Partes Perdidas" para esta solicitud, por favor presente cualquier corrección a este recibo de presentación con su respuesta a la nota. Cuando los procesos USPTO le responden a la Nota, el USPTO generará otro recibo de presentación que incorpora las correcciones solicitadas (si es apropiado).

Solicitante(s)

Krzysztof J. Cwalina, Residencia no suministrada;

Bradley M. Abrams, Residencia no suministrada;

89

Anthony J. Moore, Residencia no suministrada;

Christopher L. Anderson, Residencia no suministrada;

Michael J. Pizzo, Residencia no suministrada;

Robert A. Brigham II, Residencia no suministrada;

Datos de prioridad doméstica como lo reivindicó el solicitante

Solicitudes Extranjeras

Si se requiere, licencia de presentación extranjera otorgada: **01/21/2004**

Fecha de Publicación Proyectada: **A ser determinada - pendiente de terminación de las partes que faltan**

Solicitud de no publicación: **No**

**(Aparece sello: Departamento de Patents de Microsoft
Enero 30 2004
DPD ND Por: _____**

Solicitud de Publicación anticipada: **No**

Título **Diseño de interfaces de programación de aplicaciones (API)**

Clase Preliminar **719**

90

90

LICENCIA PARA PRESENTACIÓN EXTRANJERA DE ACUERDO CON
El Título 35, Código de los Estados Unidos, Sección 184
Título 37, Código de Regulaciones Federal, 5.11 & 5.15

OTORGADO

Se ha otorgado al solicitante una licencia de acuerdo con el 35 U.S.C. 184, si la frase "se requiere, PRESENTACIÓN DE LICENCIA EXTRANJERA OTORGADA" seguido por un dato aparece en esta forma. Tales licencias se publican en todas las solicitudes donde las condiciones para publicación de una licencia se han presentado, a pesar de si o no una licencia pueda ser solicitada como lo establece el Artículo 737 CFR 5 15. El alcance y limitaciones de esta licencia se establecen en el 37 CFR 1 15(a) a menos que una licencia temprana haya sido publicada de acuerdo con el 37 CFR 5 15(b). La licencia se sujeta a revocación una vez se notifique por escrito. La fecha indicada es la fecha efectiva de la licencia, a menos que una licencia temprana de similar alcance se haya otorgado de acuerdo con código 37 CFR 5.13 o 5.14.

Esta licencia es para ser retenida por el concesionario y puede ser usada en cualquier tiempo sobre o después de la

hoja 14 folio en blanco

fecha efectiva de esta a menos que esta se revoque. Esta licencia se transfiere automáticamente a cualquier solicitud relacionada presentada bajo el 37 CFR 1 53(d). Esta licencia no es retroactiva.

El otorgar una licencia no reduce en ninguna forma la responsabilidad del concesionario por la seguridad de la materia objeto como lo impone cualquier contrato de gobierno o las condiciones de las leyes existentes relacionadas con espionaje y la seguridad nacional o la exportación de datos técnicos. Las licencias se deben tasar de las regulaciones actuales especialmente con respecto a ciertos países, de otras agencias, particularmente la Oficina de Controles de Comercio de Defensa, Departamento de Estado (con respecto a Armas, Municiones e implementos de Guerra (22 CFR 121-128)); la Oficina de Administración de Exportaciones, Departamento de Comercio (15 CFR 370.10 (j)); la Oficina de Control de Activos Exteriores, Departamento del Tesoro (31 CFR Partes 500+) y Departamento de Energía.

92.

No Otorgado

Sin licencia según 35 U.S.C. 184 se ha otorgado en este tiempo, si la frase "SI SE REQUIERE, LICENCIA DE PRESENTACIÓN EXTRANJERA OTORGADA" NO aparece en esta forma. El solicitante puede hacer una petición para una licencia de acuerdo con el 37 CFR 5.12, si una licencia se desea antes de la expiración de 6 meses a partir de la fecha de presentación de la solicitud. Si han pasado 6 meses a partir de la fecha de presentación de esta solicitud y el concesionario no ha recibido ninguna indicación de orden de secreto de acuerdo con el 35 U.S.C. 181, el concesionario puede archivar la solicitud extranjera de acuerdo con el 37 CFR 5.15 (b).

EN LA OFICINA DE MARCAS Y PATENTES DE LOS ESTADOS UNIDOS

SOLICITUD DE PATENTE

DISEÑO DE INTERFACES DE PROGRAMACIÓN DE APLICACIÓN (API)

Inventor(es):

Krzysztof J. Cwalina

Bradley M. Abrams

Anthony J. Moore

Christopher L. Anderson

Michael J. Pizzo

Robert A. Brigham II

REGISTRO DE ABOGADO No. **MS1-1748US**

DISEÑO DE INTERFACES DE PROGRAMACIÓN DE APLICACIÓN (API)

CAMPO TÉCNICO

Esta descripción se relaciona de manera general con interfaces de programación de aplicación (API) y en particular, por vía de ejemplo pero no de limitación, con diseñar API que son fáciles de utilizar aunque suministran simultáneamente control y flexibilidad.

ANTECEDENTES

Las interfaces de programación de aplicación (API) son utilizadas por los desarrolladores para crear una amplia variedad de aplicaciones y programas. Los desarrolladores varían desde trabajadores de oficina que graban macros hasta autores de controladores de dispositivos de bajo nivel. Estos desarrolladores se basan en diferentes lenguajes y/o diferentes marcos de diferente complejidades aunque programando con diferentes conjuntos de habilidades y/o para diferentes propósitos.

Tradicionalmente, los diferentes API se han diseñado para apuntar a diferentes niveles de habilidad individuales y

DISEÑO DE INTERFACES DE PROGRAMACIÓN DE APLICACIÓN (API)

CAMPO TÉCNICO

Esta descripción se relaciona de manera general con interfaces de programación de aplicación (API) y en particular, por vía de ejemplo pero no de limitación, con diseñar API que son fáciles de utilizar aunque suministran simultáneamente control y flexibilidad.

ANTECEDENTES

Las interfaces de programación de aplicación (API) son utilizadas por los desarrolladores para crear una amplia variedad de aplicaciones y programas. Los desarrolladores varían desde trabajadores de oficina que graban macros hasta autores de controladores de dispositivos de bajo nivel. Estos desarrolladores se basan en diferentes lenguajes y/o diferentes marcos de diferente complejidades aunque programando con diferentes conjuntos de habilidades y/o para diferentes propósitos.

Tradicionalmente, los diferentes API se han diseñado para apuntar a diferentes niveles de habilidad individuales y

diferentes demandas de control (por ejemplo basados en diferentes escenarios relevantes).

Aunque esta aproximación puede ser exitosa al suministrar API que son optimizados para un desarrollador específico, esto tiene inconvenientes significativos. Por ejemplo, la aproximación de marco múltiple crea situaciones donde los desarrolladores tienen dificultades de transferir el conocimiento desde un nivel de habilidad y un tipo de escenario a otro. Cuando ellos necesitan implementa un escenario utilizando un marco diferente, los desarrolladores dan con una curva de aprendizaje muy pendiente. Y no solo es la curva de aprendizaje muy pendiente, sino que esta generalmente requiere que el código escrito en un primer marco de más bajo nivel de habilidad tenga que ser reescrito desde borrador a un segundo marco de mayor nivel de habilidad. Más aún, la creación de marcos separados para diferentes niveles de habilidad del desarrollador típicamente dan como resultado una situación en la cual los API que son objetivo o se implementan por un nivel de desarrollador no son utilizables por otro nivel de desarrollador.

La figura 1 ilustra una gráfica 101 de una curva de aprendizaje de API tradicional con relación a dos marcos diferentes. El primer marco corresponde a un marco que tiene un nivel relativamente más bajo de habilidades requeridas y/o dificultad y concomitantemente relativamente menor capacidad de control por un desarrollador. El segundo marco, de otra parte, corresponde a un marco que tiene un relativamente mayor nivel de habilidades requeridas y/o dificultad y una concomitantemente relativamente mayor capacidad de control por el desárrollador. Tal primer marco podría ser utilizado por un novato o un desarrollador no frecuente, y tal segundo marco podría ser utilizado por un desarrollador experimentado o profesional. Por ejemplo, el primer marco puede corresponder a uno diseñado para Visual Basic, y el segundo marco podría corresponder a uno diseñado para C++.

En esta aproximación tradicional, se diseñan y se emplean API relativamente separados y desiguales como parte de cada marco. Esta curva de aprendizaje pendiente pero relativamente corta se atraviesa para posibilitar el uso del API para el primer marco a un nivel y capacidad de control relativamente más bajo. En razón a la naturaleza

separada y desigual de los dos marcos API, la experiencia con el primer marco contribuye poco, si es que existe algún conocimiento, al aprendizaje del segundo API del segundo marco. Consecuentemente, una curva de aprendizaje igualmente pendiente pero aún más alta se atraviesa para posibilitar el uso del API por el segundo marco.

En otras palabras, el aprendizaje del API del primer marco no suministra una piedra escalonada para el aprendizaje de un API del segundo marco. La naturaleza regresiva de este conjunto desarticulado de marcos API se indica mediante la continuidad de espacio. Un desarrollador que ha aprendido los API del primer marco no está cerca de aprender el API del segundo marco y debe por lo tanto iniciar con lo básico del segundo marco.

Otro problema con los marcos tradicionales es que ellos en cualquier caso tienden a tener un pobre uso total. En general, las metodologías de diseño/desarrollo orientadas a objeto (OOD) (por ejemplo leguaje de modelación unificado (UML)) son "optimizadas" para el mantenimiento del diseño resultante y no para el uso de los marcos resultantes. Las metodologías OOD son mejor adecuadas para los diseños de arquitectura internos y menos

adecuados para los diseños de una capa API de una biblioteca reutilizable grande. Por ejemplo, el pobre uso puede dar como resultado metodologías OD que se enfocan solamente en la destilación a un bloque fundamental más bajo y/o que tenga una fidelidad indudable a una jerarquía de herencia estricta en todo un diseño API.

De acuerdo con esto, subsiste la necesidad de esquemas y/o técnicas que puedan al menos mejorar el espacio de continuidad regresivo de una curva de aprendizaje API tradicional y/o puedan suministrar un mejor uso API total.

RESUMEN

En una primera implementación del método de ejemplo, un método para diseñar una interfaz de programación de aplicación (API) incluye: preparar muestras de código múltiples para un escenario principal, cada muestra de código respectiva de las muestras de código múltiple corresponden a un lenguaje de programación respectivo de lenguajes de programación múltiples; y derivar el API del escenario principal que responde a las muestras de código

múltiples. En una segunda implementación del método de ejemplo, un método para diseñar un API incluye:

Seleccionar un escenario principal para un área característica; escribir al menos una muestra de código para el escenario principal; y derivar un API para el escenario principal que responde a al menos una muestra de código. En una tercera implementación de método de ejemplo, un método para diseñar un API incluye: derivar un API para un escenario que responde a al menos una muestra de código escrita con relación al escenario; desarrollar uno o más estudios de utilización en el API que utiliza múltiples desarrolladores; y revisar el API basado en uno o más de los estudios de utilización.

Otro método, sistema, aproximación, aparatos, dispositivos, medios, API, procedimiento, disposición, etc. las implementaciones se describen aquí.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

Los mismos números son utilizados en todos los dibujos con referencia a aspectos similares y/o correspondientes, características y componentes.

La figura 1 ilustra una gráfica de una curva de aprendizaje API tradicional con relación a dos diferentes marcos.

La figura 2 ilustra una gráfica de una curva de aprendizaje de API progresiva de ejemplo con relación a dos diferentes niveles de abstracción.

La figura 3 ilustra principios y prácticas de diseño de ejemplo para los API.

La figura 4 es un diagrama de flujo que ilustra una técnica de ejemplo para diseñar API por área característica.

La figura 5 es un diagrama de bloque que ilustra un esquema de ejemplo para diseñar los API por escenario principal.

La figura 6 ilustra la disparidad potencial entre tipos de componente de ejemplo que son el objetivo a dos propósitos diferentes.

La figura 7 ilustra una relación de ejemplo entre tipos de componente que están diseñados para ser extensibles y/o interoperables con el fin de cubrir dos diferentes propósitos.

La figura 8 es un componente agregado de ejemplo (AC) y tipos factorizados asociados (FT) para manejar dos diferentes propósitos con un API de dos capas.

La figura 9 ilustra un componente agregado de ejemplo y los API asociados que pueden soportar un patrón de uso Create-Set-Call.

La figura 10 ilustra un ambiente operativo de cómputo de ejemplo (o dispositivo general) que es capaz de implementar (completa o parcialmente) al menos un aspecto de diseñar y/o utilizar los API tal como se describen aquí.

DESCRIPCIÓN DETALLADA

La figura 2 ilustra una gráfica 200 de una curva de aprendizaje API progresiva de ejemplo con relación a dos diferente niveles de abstracción. Los dos diferentes

niveles de abstracción son un nivel relativamente alto de abstracción y un nivel relativamente bajo de abstracción. El nivel alto de abstracción corresponde a un ambiente de desarrollo que involucra un nivel relativamente bajo de las habilidades requeridas y/o la dificultad y una capacidad relativamente inferior concomitante para ser controlada por un desarrollador. El nivel bajo de abstracción, de otra parte, corresponde a un ambiente de desarrollo que involucra un nivel relativamente alto de habilidades requeridas y/o la dificultad y una capacidad relativamente mayor concomitante para ser controlada por un desarrollador.

Se muestra una curva de aprendizaje API progresiva que se eleva desde un punto de las habilidades requeridas inferiores y la capacidad de control concomitante de una manera relativamente suave a través de las áreas para los niveles alto y bajo de abstracción a un punto de habilidades requeridas mayores y una capacidad de control concomitante. La curva de aprendiza API progresiva exhibe una zona de continuidad entre las áreas del nivel mayor de abstracción y el nivel bajo de abstracción. Un marco API integrado posibilita una curva de aprendizaje gradual. En razón de la naturaleza integrada del marco

API, la experiencia con el alto nivel de abstracción contribuye al conocimiento hacia el uso del API de aprendizaje para el bajo nivel de abstracción así cómo también para los escenarios que demandan mayor control.

En otras palabras, el aprendizaje de los API para los mayores niveles de abstracción suministra una piedra escalonada para aprender y/o extender el API a los niveles inferiores de abstracción. Esto se indica por la forma de marco de dos capas (API) que comprende tanto las áreas de abstracción de alto y bajo nivel. La naturaleza progresiva de ciertos API, como se describen aquí adelante, le posibilita a los desarrolladores utilizar los API simples inicialmente y gradualmente comenzara utilizar componentes API más complicados. De esta manera, los desarrolladores que han aprendido los API que apuntan a los mayores niveles de abstracción se pueden mover para utilizar los API que apuntan a los niveles inferiores de abstracción en la medida en que su experiencia lo garantice y/o en la medida en que la complejidad de los escenarios que ellos están enfrentando lo demande.

Un API progresivo puede ser fácilmente utilizable (especialmente durante las fases tempranas de

aprendizaje) y altamente poderosas (especialmente en la medida en que el API es explorado durante el tiempo). Un API utilizable pude incluir uno o más de los siguientes atributos de ejemplo: un pequeño número de conceptos y/o las clases que son requeridas para arrancar, una pocas líneas de código pueden implementar escenarios simples, clases/métodos que tienen nombres intuitivos, un punto de partida natural y/o obvio es evidente, y existe una progresión clara (por ejemplo descubrible) de los conceptos/clases requeridos y/o relevantes adicionales.

Un API progresivo también puede posibilitar un avance creciente del desarrollo en un punto de dificultad inferior y una capacidad de control concomitante a un punto de dificultad mayor y una capacidad de control concomitante. Paradigmas de ejemplo para diseñar los API progresivos, así como también generalmente los API altamente utilizables se describe aquí adelante.

La figura 3 ilustra principios y prácticas de diseño de ejemplo para los API en una tabla 300. La tabla 300 indica principios y prácticas de diseño general para las cuatro categorías de ejemplo 302- 308. Específicamente, las siguientes cuatro categorías son las direccionadas:

diseño manejado de escenario 302, diseño orientado de componente 304, Opciones Predeterminadas Personalizables 306, y modelo objeto autodocumentante 308.

Cuando se diseña un API dado, los principios y las prácticas de diseño para una cualquiera o más de las categorías indicadas 302-308 se pueden emplear. Adicionalmente, dentro de cualquier categoría dada 302-308 una o más de los principios y prácticas de diseño ilustrados se pueden implementar. En otras palabras, ni cada categoría ni cada principio de diseño y práctica de este necesita ser empleado o implementado por un diseño de API dado.

La categoría de diseño manejada de escenario 302 ilustra cuatro principios y prácticas de diseño de ejemplo. Primeramente se definen escenarios principales para características o áreas tecnológicas seleccionadas. Segundo, las muestras código que corresponden a los escenarios principales son escritas primero y el API es diseñado en respuesta a este segundo. Tercero, se diseña un API progresivo, según se introdujo anteriormente y se describe adicionalmente aquí adelante. Cuarto, utilizar los escenarios principales definidos se hace fácil aunque

se hace posible utilizar otros escenarios. El diseño manejado de escenario 302 se describe adicionalmente delante de la sección titulada (Diseño Manejado de Escenario".

La categoría de diseño orientada de componente 304 ilustra tres principios y prácticas de diseño de ejemplo. Primero, son creados los componentes agregados (AC). Generalmente, los componentes agregados son dirigidos hacia los escenarios principales, son relativamente simples y fáciles de utilizar y son construidos en la parte superior de los tipos factorizados (FT). Los tipos factorizados son más fundamentales y se descomponen a niveles lógicos inferiores. Esto da como resultado un diseño API de dos capas. Segundo, estos componentes agregados están interrelacionados con los tipos factorizados. Tercero, el patrón de uso Create-Set-Call, es soportado especialmente para los componentes agregados. El diseño orientado a componente 304 se describe adicionalmente adelante en la sección titulada "Diseño Orientado a Componentes".

La categoría de Opciones Predeterminadas Personalizables 306 ilustra dos principios y prácticas de diseño de

ejemplo. Primero, se reducen las utilizaciones requeridas para usar al menos los componentes agregados. Los predeterminados son utilizados para reducir las utilizaciones requeridas. Segundo, los predeterminados se seleccionan de los que son apropiados para los escenarios principales definidos. Las Opciones Predeterminadas Personalizables 306 se describen además adelante en la sección titulada "Opciones Predeterminadas Personalizables".

La categoría de modelo de objeto de autodocumentación 308 ilustra cuatro prácticas y principios de diseño de ejemplo. Primero, los nombres simples e intuitivos son reservaos para escenarios principales. Segundo, los nombres son seleccionados con base en el uso o propósito pretendido en el tipo de componente, en lugar de una adherencia obstinada a la jerarquía de herencia. Tercero, las excepciones accionables son lanzadas de tal forma que un desarrollador recibe instrucciones que indican cómo arreglar un error del mensaje de excepción. Cuarto, los espacios de nombres limpios son producidos mediante tipos de ubicación que son simplemente utilizados en subespacios de nombres para evitar el apiñamiento de los espacios de nombre principales. El modelo de objeto

autodocumentante 308 se describe adicionalmente adelante en la sección titulada "Modelo de Objeto Autodocumentante".

Diseño Manejado de Escenario

En una implementación descrita, las especificaciones API son manejadas por escenarios. De acuerdo con esto, los diseñadores API primero escriben el código que los usuarios del API tendrán que escribir en los escenarios principales (por ejemplo principal). Los diseñadores API luego diseñan un modelo de objeto para soportar estas muestras de código. Esta aproximación contrasta con iniciar un diseño de un modelo de objeto (utilizando varias metodologías de diseño) y luego escribir las muestras de código basadas en el API resultante.

En otras palabras, especialmente para el diseño API público, los diseñadores API inician con una lista de escenarios para cada característica por área de tecnología y muestras de código para estos y producen una descripción del modelo objeto con estilo de encabezamiento basado en este. Los ejemplos de las áreas características incluyen: archivo y/o, red, mensajes,

consola, diagnósticos, accesos a la base de datos, páginas web, programación con interfaz gráfica de usuario (GUI), y asi sucesivamente.

La figura 4 es un diagrama de flujo 400 que ilustra una técnica de ejemplo para diseñar los API por área característica. En el bloque 402, son seleccionados los escenarios principales por el área característica da. Por ejemplo, para un área tecnológica dada, se pueden seleccionar los 5-10 escenarios superiores. Ellos se pueden seleccionar con base en las funciones más comúnmente utilizadas (por ejemplo tareas más comunes) o las metas más frecuentemente perseguidas para el área tecnológica dada. Por ejemplo, los escenarios de ejemplo para un área de característica de tecnología del archivo y/o son leídos de un archivo y escritos a un archivo.

En el bloque 404, las muestras de código para un escenario principal son escritas en lenguajes múltiples (por ejemplo dos o más). Por ejemplo, las muestras de código asociada con el escenario principal asociado pueden ser escritas en tres diferentes lenguajes. Las muestras código pueden implementar el escenario principal seleccionado habitual en los tres lenguajes. Tales

lenguajes incluyen, por ejemplo, VB, C#, MC++, un lenguaje de composición, y así sucesivamente; sin embargo, también se pueden utilizar otros lenguajes. Como se indica por el asterisco, se debe entender que una muestra de código (o aún más de una muestra de código) se puede escribir para el escenario principal en un lenguaje simple cuando se diseña API utilizable y potente para un lenguaje simple.

Se puede efectuar la escritura de muestras de código en lenguajes múltiples porque algunas veces los códigos escritos en diferentes lenguajes difieren significativamente. En una implementación descrita, las muestras de código para el escenario principal seleccionado habitual son escritas utilizando diferentes estilos de código que son comunes entre los usuarios del lenguaje particular (por ejemplo, utilizando características específicas del lenguaje por rasgos, utilizando las prácticas/hábitos de los desarrolladores, etc.) en los cuales se escribe una muestra de código particular. Por ejemplo, las muestras pueden ser escritas utilizando tamaño de letra específico para el lenguaje. Por ejemplo, VB es insensible a minúsculas/mayúsculas, de tal forma que las muestras de código escritas en VB

reflejan esa variabilidad. Las muestras de código escritas en C#, de otra parte, siguen el estándar de mayúsculas/minúsculas para éstas.

Otro ejemplo se relaciona con una declaración denominada "usar", que soporta C#. Por ejemplo, la llamada "usar" encapsula un bloque de ensayo/finalmente. Sin embargo, el VB no soporta esta característica, y escribir muestras de código puede indicar que utilizar esta característica en una declaración ensayo/finalmente es torpe para los usuarios VB. Aún otro ejemplo se relaciona con las asignaciones en una cláusula condicional, que soporta C#. En un ejemplo de archivo E/SO: "if ((text = reader.ReadLine() != null)" trabaja en C#. Sin embargo, la declaración de asignación no se puede utilizar dentro de la cláusula "if" en VB; en su lugar, el código es roto en declaraciones múltiples. Aún otro ejemplo relaciona la tendencia de los desarrolladores C# a utilizar constructores parametrizados aunque los desarrolladores VB usualmente no lo hacen. Por ejemplo, una codificación C# puede ser "MyClass x = new MyClass("value")" mientras una codificación VB correspondiente es "Dim x As MyClass" y "x.Property = "value"".

En el bloque 406, el API se deriva del escenario principal habitual que responde a las muestras de código escritas en lenguajes múltiples. Por ejemplo, los factores recogidos de las muestras de código escritas en cada uno de los lenguajes múltiples se pueden incorporar dentro del API. Tales factores pueden incluir similaridades a través de las diferentes muestras de código, diferencias entre dos o más muestras de código, y así sucesivamente. Tales factores, así como también otros aspectos de los bloques 404 y 406, se describen adicionalmente adelante con referencia a la FIG. 5.

Similarmente, cuando se diseña un API para un lenguaje simple, el API se deriva del escenario principal habitual que responde al o las muestras de código escritas en el lenguaje simple. Así, los factores recogidos de la o las muestras de código escritas en el lenguaje simple se pueden incorporar dentro del API. Como un ejemplo de un factor de diseño API adicional para situaciones de lenguaje simple o múltiple, un factor de diseño API puede incluir la compatibilidad con herramientas que sean orientadas hacia el lenguaje o los lenguajes para los cuales están escritas las muestras de código.

En el bloque 408, se determina si el API es demasiado complejo. Por ejemplo, el API puede ser revisado por el o los diseñadores API para determinar si el API es o no es demasiado complejo. En otras palabras, una revisión inicial se puede desarrollar para considerar si el API se puede utilizar sin entendimiento significativo de otros múltiples API específicos, sin experimentación indebida, y así sucesivamente. Tal revisión inicial también puede verificar que el API derivado sea de hecho trabajable en el escenario principal habitual en cada lenguaje relevante. Si el API es demasiado complejo, entonces el API es refinado por los diseñadores con referencia al escenario principal habitual y responde a las muestras de código escritas en los lenguajes múltiples en el bloque 406.

Si, de otra parte, se determina que el API no es demasiado complejo (en el bloque 408), entonces en el bloque 410 son desarrollados los estudios de utilidad con desarrolladores típicos. Por ejemplo, uno o más estudios de utilidad se pueden desarrollar utilizando un ambiente de desarrollo análogo a aquel que utiliza normalmente el desarrollador típico. Tal ambiente de desarrollo normal incluye similarmente intelisentido, editores, lenguaje, y

un conjunto de documentación que es más ampliamente utilizado por el grupo desarrollador objetivo.

Estudios de Utilidad

Los estudios de utilidad que apuntan a un rango amplio de desarrolladores facilitan el diseño manejado por escenario, especialmente cuando se diseñan los API para público en general. Las muestras de código escritas por el o los diseñadores API para los escenarios principales probablemente parecen simples para ellos, pero las muestras código podrían no ser igualmente simples a ciertos grupos de desarrolladores que son de hecho objetivo (por ejemplo, especialmente desarrolladores novatos y/o ocasionales). Adicionalmente, el entendimiento, que es recopilado a través de los estudios de utilidad, con relación a la manera en la cual los desarrolladores se aproximan a cada uno de los escenarios principales puede suministrar discernimiento potente en el diseño del API y que también este cumple las necesidades de todos los desarrolladores objetivo.

De manera general, los estudios de utilidad se pueden conducir tempranamente en el ciclo de producto y de nuevo

después de cualquier rediseño mayor del modelo objeto. Aunque esta es una práctica de diseño costoso, puede de hecho ahorrar recursos a largo plazo. El costo de fijar un API no utilizable o simplemente defectuoso sin introducir los cambios de rompimiento es enorme.

En el bloque 412, se averigua si los desarrolladores típicos son capaces de utilizar el API sin problemas significativos. Por ejemplo, la mayoría de los sujetos deben ser capaces de escribir códigos para el escenario seleccionado habitual sin mayores problemas. Si ellos no pueden, el API se revisa (como se describe adelante con referencia al bloque 414).

La interpretación de los problemas significativos/principales gira sobre un nivel deseado de utilidad para un grupo desarrollador dado objetivo. Por ejemplo, la referencia frecuente y/o extensiva a la documentación API detallada para el escenario principal habitual por los sujetos de prueba puede constituir problemas significativos. De manera general, si la mayoría de los desarrolladores de prueba no pueden implementar el escenario principal habitual, o si la aproximación que ellos toman es significativamente

diferente de aquella que se espera, el API se debe evaluar para posibles revisiones (hasta e incluyendo un rediseño completo).

Si se averigua que los desarrolladores típicos son incapaces de utilizar el API sin mayores problemas (en el bloque 412), entonces en el bloque 414 el API se revisa con base en las lecciones de los estudios de utilidad. Por ejemplo, se puede cambiar un predeterminado, se puede agregar otra propiedad, uno o más atributos se pueden exponer en lugar de encapsular, y así sucesivamente.

Si, de otra parte, se averigua que los desarrolladores típicos son capaces de utilizar el API sin mayores problemas (en el bloque 412), entonces en el bloque 416 el proceso se repite para cada escenario principal. Por ejemplo, otro escenario principal de los escenarios principales seleccionados para la característica dada se vuelve el escenario principal habitual para el cual son escritas las muestras código (en el bloque 404). Otra técnica de ejemplo para diseñar los API, que se enfoca más en el diseño API de dos capas, se describe adicionalmente adelante en conjunto con la FIG. 8.

La FIG. 5 es un diagrama de bloques 404/406 que ilustra un esquema de ejemplo para diseñar los API por escenario principal. El esquema de ejemplo ilustrado corresponde a los bloques 404 a 406 de la FIG. 4 para una implementación de lenguaje múltiple. Se muestra una muestra de código 502(1) para un primer lenguaje, una muestra de código 502(2) para un segundo lenguaje, y una muestra de código 502(3) para un tercer lenguaje. Cada una de las tres muestras de código 502(1, 2, 3) están dirigidas a un escenario principal habitual dado. Aunque se muestran tres muestras de código 502(1, 2, 3) que corresponden a tres lenguajes, se pueden utilizar alternativamente dos o más muestras de código 502 para cualquier número arbitrario de lenguajes objetivo en este ejemplo de implementación de lenguaje múltiple.

En una implementación descrita, los factores 506 son recogidos de muestras de código 502(1, 2, 3) que son escritas en cada uno de los tres lenguajes. Estos factores 506 son incorporados en el API 504. Específicamente, el API 504 es diseñado para soportar las tres muestras de código 502(1, 2, 3) que son escritas en los tres lenguajes respectivos correspondientes. Sin embargo, se debe entender que los factores 506 también

pueden ser aplicables a implementaciones de lenguaje simple.

Algunos factores de ejemplo 506 son descritos anteriormente con referencia a los bloques 404 y 406 de la FIG. 4, y otros factores de ejemplo 506 se indican en el diagrama de bloque 404/406 de la FIG. 5. Tales factores 506 incluyen mandatos específicos de lenguaje que son revelados mediante una revisión de muestras de código 502(1, 2, 3). Un ejemplo de una restricción especifica de lenguaje se describe con relación a la siguiente línea de ejemplo de código: "Foo f = new Food();". Un API progresivo que se diseña para soportar esta línea de muestra tiene que incluir un constructor por defecto; de otra manera, la muestra de código no compilará correctamente.

Los factores 506 también incluyen las expectativas del desarrollador que son inspiradas tanto por las peculiaridades del lenguaje como por los diferentes niveles de habilidad/experiencia de los desarrolladores típicos que gravitan naturalmente hacia diferentes lenguajes. Los factores 506 incluyen además comunidades de código y prácticas de código a través de diferentes

lenguajes como se descubre mediante una revisión de las muestras de código 502(1, 2, 3).

Aunque considerando los factores 506 que directamente se relacionan con lenguajes diferentes, otros factores 506, como se describen aquí, continúan siendo considerados. Por ejemplo, los siguientes factores 506 también son pertinentes a los API progresivos objetivo a unos ambientes de lenguaje simple así como también a ambientes de lenguaje múltiple. Primero, un factor es el número diferentes de tipos de componentes que son requeridos para completar un escenario. Generalmente, entre más tipos de componentes se requieran, más difícil es aprender. Un segundo factor es la conexión entre las líneas de éxito del código. En la medida en que ese uso de un tipo de componente conduce a un desarrollador hacia el uso del siguiente tipo de componente requerido, más fácil es utilizar el API.

Tercero, la consistencia en el nombramiento de los identificadores es otro factor. Un cuarto factor involucra el uso apropiado de propiedades, métodos, y eventos. Un quinto factor se relaciona con las similaridades posibles a uno o más de los API existentes.

Sexto, otro factor involucra el cumplimiento con las guías de diseño completas para un API. Un séptimo factor se relaciona con si los API se traslapan con otros tipos de componentes de la estructura. Octavo, la compatibilidad con las herramientas que están orientadas hacia un lenguaje particular es aún otro factor. Por ejemplo, los desarrolladores VB típicamente requieren constructores con menos parámetros y asignadores de propiedad. Otros factores 506 pueden ser considerados alternativamente.

Aún otros factores 506 que se relacionan con las interrelaciones de los componentes agregados y los tipos factorizados son descritos adelante, especialmente con referencia a la FIG. 8. Aunque el método y el esquema de las figuras 4 y 5 se pueden aplicar al diseño de los API en general, ellos son particularmente aplicables al diseño de los API de dos capas. Un paradigma del API de dos capas (por ejemplo, con componentes agregados y tipos factorizados) es el descrito adelante con referencia a las FIGS. 6-8 en la sección titulada "Diseño Orientado a Componente".

Diseño Orientado a Componente

La FIG. 6 ilustra la disparidad potencial entre los tipos de componente de ejemplo 602 que son el objetivo de dos diferentes propósitos junto con un continuo 600. El continuo 600 se extiende desde un rango de alta utilidad al lado izquierdo hasta un alto rango de control al lado derecho. Los tipos de componente múltiple 602 se esparcen a través del continuo 600.

Generalmente, los tipos de componente 602 que son ilustrados por ser relativamente mayores representan tipos que son más simples y por lo tanto más fáciles de utilizar. Al contrario, los tipos de componentes 602 que son ilustrados por ser relativamente más pequeños representan tipos que son más complejos y por lo tanto más difíciles de utilizar. Simple y complejo en este contexto se refiere a que tan fácil o que tan difícil son de utilizar los tipos de componente particular 602 para cuando se implementa un escenario específico.

Los tipos de componente 602 que se ilustran por ser relativamente más pequeños son generalmente más difíciles de utilizar por un número de razones de ejemplo como

sigue: Primero, los desarrolladores tienen más elecciones de cuales tipos de componente 602 ellos deben utilizar. En el ejemplo ilustrado, existen 14 "elecciones" para los tipos de componente más pequeños 602 comparados con las tres "elecciones" para los tipos de componente mayores 602. Más específicamente, un desarrollador tiene que saber o discernir, entre los varios tipos de componente 602(HC), qué tipo o tipos de componente utilizar. Esto involucra el entendimiento de cada uno de los tipos de componente múltiple (por ejemplo, 14) 602(HC) así como también cómo se interrelacionan ellos, lo cual contrasta con iniciar con un tipo de componente simple 602(HU) de entre los tipos de componentes más pocos (por ejemplo, 3) 602(HU). Las diferencias entre los tipos de componente 602(HU) y los tipos de componente 602(HC) se describen adicionalmente adelante.

Por vía de una analogía de ejemplo, los tipos de componente más pequeños 602 son similares a los componentes individuales de un sistema estéreo; de esta manera, un usuario tiene que saber qué componentes son necesarios y cómo engancharlos juntos. Sin engancharlos juntos, ellos generalmente no son útiles. Los tipos de componente mayores 602 son como los estéreos de todo en

uno que son fácilmente utilizables pero probablemente menos potentes así como también menos flexibles. Una segunda razón por la cual los tipos de componente más pequeños 602 son más difíciles de utilizar es que existen potencialmente más "puntos de partida". Tercero, existen generalmente más conceptos que entender. Cuarto, el desarrollador tiene que cómo se relacionan los tipos de componente individuales 602 con los otros tipos de componente 602.

En una implementación descrita, los tipos de componente 602 se dividen en aquellos con un alto propósito de utilidad 602(HU) y aquellos con un alto propósito de controlabilidad 602(HC). Los tipos de componente de alta utilidad 602(HU) son más simples y más fáciles de utilizar, pero ellos tienden a ser más inflexibles, limitados, y/o contrastantes. Ellos pueden generalmente ser utilizados sin conocimiento extensivo de un API total. Los tipos de componente de alta utilidad 602(HU) son usualmente capaces de implementar un número limitado de escenarios o lo máximo un número limitado de aproximaciones a cada escenario de interés.

Los tipos de componente de alta controlabilidad 602(HC), de otra parte, son complejos de utilizar, pero ellos suministran un mayor grado de control a los desarrolladores. Ellos son relativamente potentes y le posibilitan a los desarrolladores efectuar un ajustamiento y puesta a punto de dos niveles. Sin embargo, el desarrollo con los tipos de componente de alta controlabilidad 602(HC) implica un más completo entendimiento de muchos tipos de componente 602 para posibilitar la instanciación de múltiples tipos de componente de alta controlabilidad 602(HC) que son correctamente interligados para implementar aún escenarios relativamente directos hacia delante.

Típicamente, los tipos de componente de alta utilidad 602(HU) están presentes en los lenguajes introductorias tales como el VB, y los tipos de componente de alta controlabilidad 602(HC) están presentes en lenguajes tipo programador profesional avanzado tal como el C++. La disparidad potencial entre tipos de componente de alta utilidad 602(HU) y tipos de componente de alta controlabilidad 602(HC) que se ilustran en la FIG. 6 es al menos parcialmente mejorado por los tipos de componente 702 de la FIG. 7. Específicamente, una

interrelación entre los tipos de componente de alta utilidad 602(HU) y los tipos de componente de alta controlabilidad 602(HC) se establece en un API progresivo.

La FIG. 7 ilustra una relación de ejemplo entre los tipos de componente 702 que son diseñados para ser extensibles y/o interoperables con el fin de cubrir al menos dos diferentes propósitos junto con un continuo 700. En una implementación descrita, los tipos de componente con un alto propósito de utilidad se concretan como componentes agregados 702(AC), y tipos de componente con un alto propósito de controlabilidad se concretan como tipo de factores 702(FT). Aunque los tipos de componente 702 se dividen en solamente dos propósitos, ellos pueden alternativamente ser separados en tres o más propósitos (u otras categorías).

Una clave 704 indica que una línea sólida representa una relación para los tipos factorizados expuestos y que la línea punteada representa una relación para los tipos factorizados encapsulados. Como se ilustra, el componente agregado 702(AC)(1) tiene una relación con tres tipos de factor 702(FT). Específicamente, el tipo factorizado

702(FT)(1) tiene una relación de tipo factorizado expuesta con componente agregado 702(AC)(1), y los tipos factorizados 702(FT)(2) y 702(FT)(3) tienen una relación tipo factorizada encapsulada con componente agregado 702(AC)(1).

Aunque no se ilustra así, dos o más componentes agregados 702(AC) pueden tener una relación encapsulada y/o expuesta con el mismo tipo factorizado 702(FT). Los tipos factorizados expuestos y encapsulados 702(FT) y los componentes agregados 702(AC), así como también las relaciones entre ellos, son descritos adicionalmente adelante, incluyendo con referencia a la FIG. 8.

Los diseños orientados a componente se relacionan con el ofrecimiento de un objeto simple por concepto de usuario opuesto a requerir objetos múltiples por concepto lógico. Los componentes agregados por lo tanto usualmente corresponden a un concepto de usuario y son más simples desde la perspectiva de uso. Los componentes agregados tienen capas en la parte superior de los tipos factorizados. Por vía de comparación de ejemplo, los componentes agregados pueden modelar una cosa tal como un archivo, y los tipos factorizados pueden modelar un

estado de una cosa tal como una visión de un archivo. Juntos, los componentes agregados y los tipos factorizados suministran una curva de aprendizaje progresiva y gradual para los nuevos desarrolladores, especialmente con respecto a un API particular dado.

Diseño Orientado a Componente para Componentes Agregados

Muchas áreas características pueden beneficiarse de los tipos *façade* que actúan como vistas simplificadas sobre un recordatorio más complejo pero bien factorizado de los API de área característica. En una implementación descrita, la *façade* cubre los 5-10 escenarios superiores en un área característica dad y opcionalmente otras operaciones de alto nivel. Los componentes agregados 702(AC) pueden servir como tales tipos *façade*, y los tipos factorizados 702(FT) pueden suministrar un paisaje API restante complejo bien factorizado.

Cada componente agregado amarra múltiples clases factorizadas de más bajo nivel en un componente de mayor nivel para sostener los escenarios principales superiores. Por ejemplo, un componente agregado de correo puede amarrar el protocolo SMTP, los sockets,

codificaciones, y así sucesivamente. De manera general, cada componente agregado suministra un mayor nivel de abstracción en lugar de simplemente un diferente camino de hacer las cosas. Suministrar operaciones simplificadas de alto nivel es de ayuda para aquellos desarrolladores que no quieren aprender la extensión completa de la funcionalidad suministrada por un área característica y simplemente desean lograr sus a menudo muy simples tareas sin un estudio significativo de la exploración del API.

Generalmente, el diseño orientado a componente es un diseño basado en constructores, propiedades, métodos, y eventos. Utilizar los componentes agregados es una aplicación relativamente extrema de un diseño orientado a componente. Un conjunto de ejemplo de parámetros para diseño orientado a componente de componentes agregados se suministra adelante:

Constructores: los componentes agregados tienen constructores predeterminados (parámetro-menos).

Constructores: los parámetros constructores opcionales corresponden a propiedades.

Propiedades: la mayoría de las propiedades tienen obtención y asignación.

Propiedades: las propiedades tienen predeterminados sensibles.

Métodos: los métodos no toman parámetros si los parámetros especifican opciones que permanecen constantes a través de las llamadas de método (en los escenarios principales seleccionados). Tales opciones se pueden especificar utilizando propiedades.

Eventos: los métodos no toman delegados como parámetros. Las retrollamadas son implementadas en términos de eventos.

El diseño orientado a componente implica considerar como se utiliza el API en lugar de enfocar la mera discusión de los métodos, propiedades, y eventos en el modelo objeto. Un modelo de uso de ejemplo para un diseño orientado a componente involucra un patrón de iniciación de un tipo con un predeterminado o un constructor relativamente simple, asignando algunas propiedades al caso, y luego llamando métodos simples. Este patrón es

denominado un patrón de uso Create-Set-Call. Un ejemplo general sigue:

```
'VB
'Instantiate
Dim T As New T()

'Set properties/options.
T.P1 = V1
T.P2 = V2
T.P3 = V3
'Call methods; opcionalmente cambia opciones entre llamadas.
T.M1()
T.P3 = V4
T.M2()
```

Cuando los componentes agregados soportan este patrón de uso Create-Set-Call, los componentes agregados se comportan con las expectativas de los usuarios principales de los componentes agregados. Más aún, las herramientas, tal como el intelisentido y los diseñadores, son optimizados para este patrón de uso. Un

ejemplo de código concreto que muestra el patrón de uso Create-Set-Call sigue:

```
'VB
'Instantiate
Dim File As New FileObject()

'Set properties.
File.Filename = "c:\foo,txt"
File.Encoding = Encoding.Ascii

'Call methods.
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
File.Close()
```

Con un componente agregado de ejemplo que es parte de un API progresivo, asignar la propiedad "File.Encoding" es opcional. El API tiene un predeterminado para un archivo preseleccionado que codifica si uno no es especificado. Similarmente, con relación a "File.Open()", especificar un "OpenMode.Write" es opcional. Si no se especifica, se emplea un predeterminado "OpenMode" como preseleccionado.

Un tema con el diseño orientado a componente es que este resulta en tipos que pueden tener modos y estados inválidos. Por ejemplo, un constructor predeterminado le permite a los usuarios instancionar un "FileObject" sin especificar un "FileName". Intentar llamar un Open() sin primero asignar el "FileName" da como resultado una excepción porque el "FileObject" es un estado inválido con respecto a ser abierto (por ejemplo, no se ha especificado aún un nombre de archivo). Otro tema es que las propiedades, que se pueden ajustar opcionalmente e independientemente, no se ponen en práctica de manera consistente y los cambios al estado del objeto. Adicionalmente, tales propiedades "modales" inhiben el compartir de una instancia de objeto entre los consumidores porque el primer usuario tiene que revisar un valor brevemente ajustado antes de reutilizarlo en caso de que un segundo usuario haya cambiado el valor. Sin embargo, la utilización de los componentes agregados hace pesar más estos temas para una vasta multitud de desarrolladores.

Cuando los métodos de llamada de usuario no son válidos en el estado habitual del objeto, un "InvalidOperationException" es disparado. El mensaje de

excepción puede claramente explicar que propiedades necesitan ser cambiadas para conseguir el objeto en un estado válido. Estos mensajes de excepción clara solucionan parcialmente el tema del estado inválido y dan como resultado un modelo de objeto que es más autodocumental.

Los diseñadores API a menudo intentan diseñar tipos tal que los objetos no puedan existir en un estado inválido. Esto se logra, por ejemplo, al tener todas las asignaciones requeridas como parámetros para el constructor, al tener propiedades get-only para configuraciones que no puedan ser cambiadas después de la instanciación, y al romper la funcionalidad en tipos separados de tal forma que las propiedades y los métodos no se traslapen. En una implementación descrita, esta aproximación se emplea con tipos factorizados pero no con componentes agregados. Para los componentes agregados, a los desarrolladores le son ofrecidas excepciones claras que comunican los estados inválidos a ellas. Estas excepciones claras se pueden lanzar cuando las operaciones están siendo ejecutadas, en lugar de cuando el componente es inicializado (por ejemplo, cuando un constructor es llamado o cuando se ajusta un propiedad),

con el fin de evitar situaciones donde el estado inválido es temporal y se consigue "fijo" en una línea subsecuente de código.

Tipos Factorizados

Como se describió anteriormente, los componentes agregados suministran accesos directos para la mayoría de las operaciones comunes de alto nivel y son usualmente implementadas como una *façade* sobre un conjunto de tipos más complejos pero también más ricos, que son llamados tipos factorizados. En la implementación descrita, los tipos factorizados no tienen modos y no tienen tiempos de vida muy claros.

Un componente agregado puede suministrar acceso a sus tipos factorizados internos a través de algunas propiedades y/o métodos. El acceso de los usuarios a los tipos factorizados internos es en escenarios relativamente avanzados o en escenarios donde la integración con diferentes partes del sistema se requiere. La capacidad de acceso a el o los tipos factorizados que están siendo utilizados por un componente agregado le posibilita al código que ha sido

escrito utilizando el componente agregado agregar incrementalmente complejidad para escenarios avanzados, o integrarse con otros tipos de componente, sin tener que reescribir el código desde el inicio con un enfoque sobre la utilización de los tipos factorizados.

El siguiente ejemplo muestra un componente agregado de ejemplo ("FileObject") que expone su tipo factorizado interno de ejemplo ("StreamWriter"):

```
Dim File As New FileObject("c:\foo.txt")
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
AppendMessageToTheWorld(File.StreamWriter)
File.Close()
...

Public Sub AppendMessageToTheWorld(ByVal Writer As
StreamWriter)
...
End Sub
```

Operaciones de Alto Nivel

En una implementación descrita, los componentes agregados, tales como los API superior o más alto nivel (por ejemplo, desde un nivel de perspectiva de abstracción), son implementados de tal forma que ellos parecen trabajar "mágicamente" sin que el usuario se entere algunas veces de las complicadas cosas que ocurren de manera soterrada. Por ejemplo, un componente agregado "EventLog" oculta el hecho de que un *log* tiene tanto manejo de lectura como manejo de escritura, los cuales son abiertos con el fin de utilizarlo. En tanto tiene que ver con el desarrollador, el componente agregado puede ser instancionado, las propiedades se pueden ajustar, y los eventos *log* se pueden escribir sin preocupación del funcionamiento *under-the-hood*.

En algunas situaciones, un poco más de transparencia puede facilitar algunas tareas con el desarrollador. Un ejemplo es una operación en la cual el usuario toma una acción explícita como resultado de la operación. Por ejemplo, un archivo implícitamente abierto y que luego requiera que el usuario explícitamente lo cierre está llevando el principio de "mágicamente" trabajar demasiado

lejos. Sin embargo, un diseñador API diligente puede a menudo ser capaz de diseñar soluciones expertas que ocultan aún aquellas complejidades. Por ejemplo, leer un archivo se puede implementar en una operación simple que abre un archivo, lee su contenido, y se cierra; el usuario está así protegido de las complejidades relacionadas con la abertura y cierre de los manejos de archivo.

Adicionalmente, utilizar componentes agregados no involucra implementar ninguna interfaz, modificar ningún archivo de configuración, y así sucesivamente. En su lugar, los diseñadores de bibliotecas pueden embarcar las implementaciones predeterminadas para interfaces que sean declaradas. Más aún, los valores de la configuración son opcionales y mantenidos por predeterminados sensibles.

La FIG. 8 ilustra un componente agregado de ejemplo 702(AC) y tipos factorizados asociados 702(FT) para manejar dos diferentes propósitos con dos capas API. 800. El componente agregado 702(AC) representa una capa primera o mayor, y tipos factorizados 702(FT) representan una capa segunda o inferior. La primera capa

efectivamente se construye sobre la segunda capa con una interfaz de costumbre.

Como se ilustra, el componente agregado 702(AC) incluye miembros de componente múltiples agregados (AC) 802. Específicamente, los miembros de componente agregado 802(1), 802(2), 802(P)(1), 802(P)(2), 802(M)(1), 802(M)(2), y 802(M)(3) son mostrados. El componente agregado 702(AC) también incluye tipos factorizados expuestos 702(FT-Ex) y tipos factorizados encapsulados 702(FT-En). Específicamente, los tipos factorizados expuestos 702(FT-Ex)(1) y 702(FT-Ex)(2) y los tipos factorizados encapsulados 702(FT-En)(1) y 702(FT-En)(2) son mostrados. Los tipos factorizados 702(FT) también incluyen miembros de tipos factorizados (FT) 804.

En una implementación descrita, el componente agregado 702(AC) incluye al menos un miembro componente agregado 802, el cual puede ser un método o una propiedad para ejemplo. Los miembros de componente agregado 802 pueden por lo tanto incluir métodos de componente agregado 802(M) y propiedades de componente agregado 802(P). Estos miembros de componente agregado 802, tal como los miembros de componente agregado 802(1) y 802(2), pueden

ser específicos al componente agregado 702(AC). En otras palabras, algunos miembros de componente agregado 802 como los miembros de componente agregado 802(1) y 802(2) que están sobre el componente agregado 702(AC) pueden no confiar ningunos tipos factorizados 702(FT). Alternativamente, algunos miembros de componente agregado 802 pueden estar enlazados a tipos factorizados subyacentes 702(FT).

Los tipos factorizados 702(FT) pueden ser tipos factorizados expuestos 702(FT-Ex) o tipos factorizados encapsulados 702(FT-En). Los tipos factorizados expuestos 702(FT-Ex) son tipos factorizados 702(FT) de un componente agregado dado 702(AC) que puede ser accesible mediante o a otros tipos de componente general 702(FT o AC) sin utilizar o ir a través de miembros de componente agregado individuales 802 del componente agregado dado 702(AC). Si un tipo factorizado 702(FT-Ex/En) es regresado por un miembro componente agregado 802 (o un método o una propiedad), entonces el tipo factorizado 702(FT-Ex) es expuesto. De otra manera, el tipo factorizado 702(FT-En) es encapsulado.

En otras palabras, un miembro componente agregado 802 puede exponer un miembro tipo factorizado 804, o un miembro componente agregado 802 puede regresar una instancia tipo factorizada. Lo último puede ocurrir con los tipos factorizados expuestos 702(FT-Ex), y el anterior puede ocurrir con los tipos factorizados encapsulados 702(FT-En). Los tipos factorizados encapsulados 702(FT-En) son tipos factorizados 702(FT) de un componente agregado dado 702(AC) que están contenidos dentro o internos al componente agregado dado 702(AC). Cada uno de los tipos factorizados 702(FT) pueden incluir uno o más miembros 804 (algunos de los cuales son específicamente indicados en la FIG. 8) que son métodos y/o propiedades.

Como se ilustra, dos miembros de método 804 del tipo factorizado encapsulado 702(FT-En)(1) son expuestos mediante el componente agregado 702(AC) como un miembro método 802(M)(1) y el miembro método 802(M)(2). Un miembro método 804 del tipo factorizado encapsulado 702(FT-En)(2) es expuesto por el componente agregado 702(AC) como miembro método 802(M)(3).

El tipo factorizado expuesto 702(FT-Ex)(1) es en si mismo expuesto a un miembro propietario 802(P)(1) del componente agregado 702(AC). Similarmente, el tipo factorizado expuesto 702(FT-Ex)(2) es también expuesto como un miembro propietario 802(P)(2) del componente agregado 702(AC). Como se indica, un miembro tipo factorizado (FT) 804 del tipo factorizado expuesto 702(FT-Ex)(1) es expuesto con el fin de ser separadamente accesible (es decir, accesible sin utilizar directamente un miembro individual 802 del componente agregado 702(AC)). De esta manera, un miembro tipo factorizado 804 de un tipo factorizado expuesto 702(FT-Ex) puede aún ser accedido aún si este no es individualmente expuesto por un miembro componente agregado 802 del componente agregado 702(AC).

Así, el miembro indicado 804 del tipo factorizado expuesto 702(FT-Ex)(1) es expuesto con el fin de ser accesible por los tipos de componente 702 que son externos al componente agregado 702(AC) sin utilizar un miembro 802 de éste. Como se indicó por las líneas punteadas que emanan del tipo factorizado expuesto 702(FT-Ex)(1), los tipos factorizados expuestos 702(FT-Ex) pueden ser "transferidos" para uso por otros tipos de

componentes 702 (especialmente por otros tipos factorizados 702(FT)) que son incapaces de interactuar con los componentes agregados 702(AC) o que pueden lograr mejor su propósito pretendido utilizando un tipo factorizado expuesto transferido 702(FT-Ex)(1) solo. Se debe notar que el objeto (tipo factorizado expuesto 702(FT-Ex)(1)) que es "transferido" no es una copia sino en su lugar una parte actual del componente agregado expuesto 702(AC). Como resultado, las operaciones del objeto transferido afectan el componente agregado 702(AC).

De manera general, si un tipo factorizado 702(FT) es encapsulado, este no es expuesto a un consumidor; en su lugar, las propiedades de configuración 802(P) o los métodos de llamado 802(M) sobre un componente agregado 702(AC) puede originar tipos factorizados 702(FT) a ser creados, las propiedades 804 a ser establecidas, o los métodos 804 a ser llamados sobre el tipo factorizado subyacente 702(FT). Estos miembros 802 y 804 pueden no tener una correspondencia uno a uno; por ejemplo, configurando varias propiedades 802(P) sobre un componente agregado 702(AC) podría ser cacheado en el componente agregado 702(AC). Subsecuentemente llamar un

método 802(M) sobre el componente agregado 702(AC) puede originar que un tipo factorizado 702(FT) sea creado utilizando los valores previamente especificados de las varias propiedades 802(P) como argumentos de constructor para el tipo factorizado 702(FT).

En una implementación descrita, los componentes agregados 702(AC) difieren de los componentes más tradicionales orientados a objeto en al menos dos formas además de la expuesta de los tipos factorizados expuestos 702(FT-Ex). Primero, un componente agregado 702(AC) no necesariamente expone cada miembro 804 de todos sus tipos factorizados 702(FT). En otras palabras, los componentes agregados 702(AC) no son estrictamente devotos a una jerarquía de herencia. Segundo, los componentes agregados 702(AC) pueden tener modos y así poder periódicamente tener estados que resultan en operaciones inválidas.

Como una guía al diseño orientado a componente con respecto a los factores 506 (de la FIG. 5), si un tipo factorizado 702(FT) se expone o se encapsula dentro de un componente agregado 702(AC) puede estar basado en uno o más de un número de factores. Primero, un tipo factorizado particular 702(FT) se expone como un miembro

de propiedad 802(P) de un componente agregado dado 702(AC) si el tipo factorizado particular 702(FT) incluye la funcionalidad que no está expuesta por el componente agregado dado 702(AC). Segundo, un tipo factorizado particular 702(FT) es expuesto como un miembro de propiedad 802(P) de un componente agregado dado 702(AC) si otros tipos de componente general 702 de la estructura se pueden beneficiar de una transferencia para el consumo directo del tipo factorizado particular 702(FT). De otra parte, un tipo factorizado particular 702(FT) no está expuesto (y es por lo tanto encapsulado) cuando la funcionalidad del tipo factorizado particular 702(FT) es completamente expuesta a un componente agregado dado 702(AC) y cuando el tipo factorizado particular 702(FT) no es útil para la transferencia de los otros tipos de componente 702.

Un desarrollador puede iniciar con el componente agregado 702(AC), especialmente para implementar escenarios más simples y/o principales. Cuando el desarrollador desea o necesita implementar un escenario más complejo, el desarrollador puede incremental y gradualmente iniciar el acceso directamente y utilizar los tipos factorizados expuestos 702(FT-Ex), incluyendo los atributos de bajo

nivel de estos, durante el tiempo. El código original que confía en el componente agregado más simple 702(AC) no necesita ser arrojado y reemplazado con codificación más complicada que descanse solamente sobre los tipos factorizados 702(FT). Las dos capas de la estructura API pueden ser utilizadas en proporciones variantes y pueden coexistir simultáneamente.

Diseñar una estructura API de dos capas se puede lograr utilizando la siguiente técnica de ejemplo que se describe en diez fases: Primero, se selecciona un conjunto de escenarios principales para un área característica particular. Segundo, se escriben códigos de muestra que muestren las líneas preferidas del código para los escenarios principales seleccionados. Tercero, los componentes de agregados son derivados con los métodos apropiados, predeterminados, abstracciones, nombramiento, etc. para soportar las muestras de código de las líneas de código.

Cuarto, las muestras de código de la segunda fase son refinadas según sea adecuado de acuerdo con los componentes agregados derivados. Quinto, las muestras de código refinadas son evaluadas para saber si ellas son o

no suficientemente simples. Si no, la técnica continúa de nuevo en la tercera fase. Si es así, entonces en la sexta fase se determina si existen escenarios adicionales, usos, interacciones con otros componentes, y/o otros requisitos. Séptimo, el diseñador API decide si cualquiera de los requisitos adicionales descubiertos en la fase sexta se pueden agregar a los componentes agregados sin adicionar complejidad indebida a los escenarios principales seleccionados.

Octavo, si los requisitos adicionales no pueden ser adicionados a los componentes agregados, una factorización ideal (por ejemplo, basada en metodologías orientadas a objeto u otras metodologías analíticas) de un conjunto completo de funcionalidad para los tipos factorizados se define con base en la séptima fase. Noveno, se determina cómo y si los componentes agregados encapsulan o exponen la funcionalidad de los tipos factorizados que son definidos en la octava fase. Décimo, los tipos factorizados son refinados como apropiados para soportar los componentes agregados así como también los requisitos adicionales. Utilizando esta técnica de ejemplo, una estructura API de dos capas que tiene

componentes agregados 702(AC) y tipos factorizados 702(FT) se puede diseñar.

La FIG. 9 ilustra un componente agregado de ejemplo 702(AC) y los API asociados 902, 802(P), 802(M), y 904 que pueden soportar un patrón de uso create-set-call. Se ilustran los siguientes grupos API de ejemplo: constructores 902, propiedades 802(P), métodos 802(M), y eventos 904. Con un patrón de uso create, set, call, una instancia de componente agregado 702(AC) es inicialmente creada por el desarrollador con confianza en los constructores predeterminados 902 (por ejemplo, parameter-less).

En segundo lugar, el desarrollador establece cualesquier propiedades 802(P) para las cuales los valores por defecto son inapropiados y/o no preferidos para el uso propuesto del objeto. En tercer lugar, los métodos deseados 802(M) son llamados por el desarrollador. Las respuestas de las llamadas son entonces implementadas en términos de evento 904.

Opciones Predeterminadas Personalizables

Las Opciones Predeterminadas Personalizables se relacionan con tener opciones por defecto siempre que sea practicable para al menos componentes agregados. Cuando se diseña un API por ejemplo con múltiples muestras de código correspondientes a múltiples lenguajes, un valor idéntico que es pasado en cada una de las muestras de código puede en su lugar ser establecido por una opción por defecto para el componente agregado. Las Opciones Predeterminadas Personalizables pueden ser cambiadas estableciendo una o más propiedades en el componente agregado.

Muchos desarrolladores prefieren codificar por ensayo y error en una posición a tomarse el tiempo de leer la documentación y entender completamente un área de rasgo antes de iniciar un proyecto. Esto es particularmente verdad para desarrolladores nuevos y ocasionales, tales como aquellos que codifican con VB. Estos desarrolladores frecuentemente tratan de experimentar con un API para descubrir qué hace y cómo trabaja, y entonces ellos ajustan su código lentamente e incrementalmente hasta que la implementación API logra su objetivo. La popularidad

de editar y continuar acercándose al desarrollo es una manifestación de esta preferencia.

Algunos diseños API se prestan ellos mismos para "codificar por experimentación" y algunos no. Hay múltiples aspectos que afectan el nivel de éxito que un desarrollador pudiera tener cuando usa una codificación por enseñanza por experimentación. Estos aspectos incluyen: (i) qué tan fácil es localizar el API justo para la tarea a mano; (ii) qué tan fácil es iniciar usando un API, sin importar si este (inicialmente) hace lo que el desarrollador quiere que haga o no; (iii) qué tan fácil es descubrir que los puntos de adaptación son para un API; (iv) qué tan fácil es descubrir la adaptación correcta para un escenario dado; y (v) así sucesivamente.

En una implementación descrita, los API están diseñados de modo que requieran poca o ninguna inicialización (por ejemplo, una cantidad mínima de inicialización). Por ejemplo, un API puede ser diseñado de modo que un constructor por defecto o un constructor con un parámetro simple sean suficientes para iniciar el trabajo con un tipo. Cuando la inicialización es necesaria, una

excepción que resulta de no llevar a cabo la iniciación claramente explica qué necesita hacerse y/o cambiarse para poder remover o impedir la excepción. Por ejemplo, una excepción puede estipular que o cual propiedad requiere ser establecida.

Como vía de ejemplo pero no de limitación, la regla del dedo gordo es que el constructor más simple tiene tres o menos parámetro (con un límite superior de cinco). En adición, los constructores más simples deberían evitar tipos complejos como cualquiera de los parámetros, en donde los tipos complejos pueden ser otros tipos factorizados o componentes agregados. Otra regla del dedo gordo es que los constructores más simples descansan en tipos primitivos como enumeraciones, frases, enteros, y etc. los tipos pueden también implementar más sobrecargas de constructor complejo para soportar más escenarios complejos.

En resumen, la adaptabilidad de los API puede ser simplificada suministrando propiedades con buenas opciones predeterminadas para todos los puntos personalizables. (Sin embargo, los desarrolladores deben generalmente ser capaces de agregar nuevo código al

código existente cuando adaptan sus escenarios; reescribir el código entero desde nada usando un API diferente debería ser opcional). Por ejemplo, un componente agregado de un sistema de fila de mensajería permite el envío de mensajes después de pasar por un hilo de trayecto al constructor y llamar un método de envío. Las propiedades de mensaje, tales como prioridad de mensaje y algoritmos de encriptación, pueden ser adaptados agregando código al escenario principal.

Modelo de Objeto Autodocumentado

El modelo de objetos de autodocumentación se relaciona con diseñar una estructura API en la cual un desarrollador puede buscar en objetos y miembros de la misma para aprender a cerca de ellos lo mismo que ser capaz de usarlos. Por ejemplo, los nombres pueden basarse en cómo un tipo es esperado para ser usado en lugar de la devoción a una jerarquía de herencia que muchos desarrolladores no desean estudiar. En resumen, un modelo de objeto de autodocumentación facilita la cualidad de descubrir por desarrolladores futuros.

Como se anotó anteriormente, algunos desarrolladores prefieren codificar por ensayo y error y leer la documentación solamente cuando su intuición falla para implementar su escenario propuesto. Así, un modelo de objeto de autodocumentación debería evitar requerir que los desarrolladores lean documentación cada vez que ellos quieren llevar a cabo cualesquier tales aún simple. Un conjunto ilustrativo de principios y prácticas para ayudar en la producción intuitiva de los API que son relativamente autodocumentados para una implementación descrita se presentan enseguida. Cualquiera o más de ellos pueden ser utilizados en una implementación de diseño API o en un API.

Nominación

Un primer principio de guía es reservar nombres simples e intuitivos para tipos que los usuarios requieren usar (por ejemplo, instanciar) en la mayoría de escenarios comunes. Los diseñadores frecuentemente "malgastan" los mejores nombres para abstracciones, con los cuales la mayoría de usuarios no les interesa o no están ligados a ellos. Por ejemplo, nombrar una base abstracta clase "archivo" y luego suministrar un tipo concreto "Archivo

XYZ" trabaja bien si la expectativa es que todos los usuarios tienen que entender la jerarquía de herencia antes de que ellos puedan iniciar el uso de los API. Sin embargo, si los usuarios no entienden la jerarquía, la primera cosa que ellos tratarán de usar, más frecuentemente sin éxito, es el tipo "archivo". Más específicamente, los nombres más comunes o esperados son reservados para componentes agregados que objetivan la parte superior de los escenarios principales con los nombres menos comunes o familiares usados en conceptos y abstracciones.

Un segundo principio de guía es usar nombres identificadores descriptivos que claramente establecen lo que cada método hace y lo que cada tipo y parámetro representa. Por ejemplo, los diseñadores API no deben dudar de usar bastantes palabras cuando escojan nombres identificadores. Por ejemplo, "EventLog.DeleteEventSource(string source, string machineName)" puede verse con bastantes palabras, pero es argumentable que tiene un valor de uso positivo en la red. Más aún, los nombres de tipo y parámetro establecen lo que un tipo o parámetro representa, en vez de lo que hace. Los nombres de método establecen lo que el método

hace. Por supuesto, nombres de métodos precisos en palabras son más fáciles para los métodos que tener semánticas simples y claras, que es otra razón de por qué evitar semánticas complejas es un principio de diseño general bueno a seguir.

Una práctica de diseño de guía es incluir una discusión acerca de las escogencias de los nombres como una parte significativa de la aplicación API en la revisión y/o los ensayos. Consideraciones ilustrativas y preguntas ilustrativas incluyen: qué son los tipos con los que los escenarios comienzan en la mayor parte? Cuáles son los nombres que la gente más piensa primero cuando tratan de implementar un escenario dado? Son los nombres de los tipos comunes lo que los usuarios piensan primero? Por ejemplo, ya que "file" es el nombre que la gente más piensa cuando trabajan con escenarios de archivos y/o, el componente agregado para accesar los archivos puede llamarse "archivo". Adicionalmente, los métodos usados más comúnmente de los tipos usados más comúnmente y sus parámetros son revisados y ensayados. Por ejemplo, puede cualquiera que sea familiar con una tecnología, pero no con el diseño API específico bajo consideración,

reconocer y llamar aquellos métodos rápidamente, correctamente y fácilmente?

Excepciones

Como se indicó anteriormente, las excepciones pueden facilitar los API de autodocumentación. En otras palabras, los API deben llevar al usuario a hacer la siguiente cosa, y las excepciones son capaces de proveer buena comunicación para lo que se requiere enseguida. Por ejemplo, el siguiente código de muestra arroja una excepción con un mensaje la propiedad "fliname" requiere ser establecida antes de intentar abrir el "FileObject".:

```
'VB
'instanciar
Dim File As New FileObject()
'el nombre del archivo no está establecido.

File.Open()
```

Mecanografiado Fuerte

Otro principio de guía para facilitar los API intuitivos es un mecanografiado fuerte. Por ejemplo, llamar "Customer.Name" es más fácil que llamar "Customer.Properties['Name']". Adicionalmente, teniendo tal una propiedad "Name" que recupere el nombre como palabra es más utilizable que si la propiedad recupera un objeto.

Estos son casos en donde la propiedad se embolsa con un asesor con base en frases, luego liga las llamadas, y otros tipos no muy fuertes de API son deseados, pero ellos están relegados a rarezas y no son la regla. Más aún, los diseñadores API pueden suministrar ayudadores de mecanografiado fuerte para las operaciones más comunes que el usuario lleva acabo sobre capas API con mecanografiado no muy fuerte. Por ejemplo, un tipo de cliente puede tener una propiedad de bolsa, pero puede adicionalmente suministrar API de mecanografiado fuerte para propiedades más comunes como "Nombre", "Dirección", etc.

Vectorización hacia Simplicidad

Todavía otro principio de guía es buscar la simplicidad especialmente para los escenarios principales. Las metodologías de diseño estándares tienen como propósito producir diseños que son optimizados para su mantenimiento de modo tal como usando abstracciones. Por lo tanto, las metodologías de diseño modernas producen una cantidad de abstracciones. Un problema es que tales metodologías de diseño operan sobre la presunción de que los usuarios de los diseños resultantes se volverán expertos en ese diseño antes de iniciar la implementación de escenarios aún simples. Sin embargo, esto no es frecuentemente el caso en el mundo real.

En una implementación descrita, para al menos escenarios simples, los diseñadores API aseguran que las jerarquías del modelo de objeto sean suficientemente simples de modo que ellas puedan ser usadas sin la necesidad de tener que entender como se astuta el área de rasgos completa junta o interoperada. Un API bien diseñado resultante puede requerir que el desarrollador entienda el escenario principal que va a ser implementado, pero no requiere un

entendimiento completo del diseño de la librería que va a ser usada para implementarlo.

Generalmente, los API de escenario principal están dirigidos o corresponden a partes lógicas bien conocidas o físicas del sistema en lugar de abstracciones. Los tipos que corresponden a abstracciones son usualmente difíciles de usar sin un entendimiento de cómo de todas las partes del área de rasgos se ajusta junta o interoperada; ellos por lo tanto son más relevantes cuando se requiere la integración de rasgos cruzados.

Otra práctica de guía es usar metodologías de diseño estándar (por ejemplo, UML) cuando se diseñan arquitecturas internas y algunos de los tipos factorizados, pero no cuando se diseñan los API para los escenarios principales o comunes (por ejemplo, aquellos con componentes agregados). Cuando se diseñan componentes agregados para escenarios principales, el escenario manejado por el diseño junto con la formación de prototipos, los estudios de utilización, y la iteración (como se describe de aquí en adelante) son empleados en su lugar.

Nombres de Espacios Limpios

Todavía otro principio de guía es que tipos que son (muy) raramente usados son colocados en sub espacios de nombres para evitar aglomeración en los espacios de nombres principales. Por ejemplo, los siguientes dos grupos de tipos pueden ser separados de sus nombres de espacio principales: tipos de permiso y tipos de diseño. Por ejemplo, los tipos de permiso pueden residir en un sub espacio de nombre ".Permissions", y los tipos de diseño - tiempo solamente pueden residir en un sub espacio de nombre ".Design".

Las acciones, aspectos, rasgos, componentes, etc. de las figuras 1-9 son ilustrados en diagramas que están divididos en múltiples bloques. Sin embargo, el orden, interconexiones, interrelaciones, diseño, etc. en los cuales las figuras 1 a 9 se describen y/o mostradas no tienen el propósito de ser construidos como una limitación, y cualquier número de bloques pueden ser modificados, combinados, redispuestos, aumentados omitidos, etc. en cualquier manera para implementar uno o más sistemas, métodos, dispositivos, procedimientos, medios, API, aparatos, disposiciones, etc. para diseñar

los API. Más aún, aunque la descripción incluye aquí referencias a implementaciones específicas (y los ambientes operativos ilustrativos de la figura 10), las implementaciones ilustradas y/o descritas pueden ser implementadas en cualquier hardware, software, firmware adecuado, o combinación de ellos y usar cualquier arquitectura de software adecuada, lenguaje de codificación, definiciones de escenario, formatos de estudio de utilización y etc.

Ambiente Operativo Ilustrativo para Computador u otro Dispositivo

La figura 10 ilustra un ambiente operativo de cómputo ilustrativo (o dispositivo general) 1000 que es capaz de (completa o parcialmente) implementar por lo menos un sistema, dispositivo, aparato, componente, disposición, protocolo, enseñanza, método, procedimiento, medio, API, alguna combinación de ellos, etc. para diseñar los API como se describió aquí. El ambiente operativo 1000 puede ser utilizado en las arquitecturas de computadora y red descrita enseguida.

El ambiente operativo ilustrativo 1000 es solamente un ejemplo de un ambiente y no tiene el propósito de sugerir ninguna limitación en cuanto al alcance de uso o funcionalidad del dispositivo aplicable (incluyendo computador, nodo de red, dispositivo de entretenimiento, electrodoméstico móvil, dispositivo electrónico general, etc.) como arquitecturas. Tampoco debería el ambiente operativo 1000 (o los dispositivos del mismo) ser interpretados como que tienen cualesquier dependencia o requerimiento relacionado con una cualquiera o cualquier combinación de los componentes como se ilustró en la figura 10.

Adicionalmente, al diseñar los API y/o los API resultantes de esto puede ser implementado con numerosos otros dispositivos de propósitos general o dispositivos de propósito especial (incluyendo sistemas de cómputo) como ambientes o configuraciones. Ejemplos de dispositivos, sistemas, ambientes, y/o configuraciones bien conocidas que pueden ser utilizadas para este uso incluyen, pero no se limitan a, computadores personales, computadores servidores, clientes delgados, clientes grandes, asistentes digitales personales (PDA) o teléfonos móviles, relojes, dispositivos de mano o

portátiles, sistemas multiprocesadores, sistemas con base en microprocesador, cajas de televisión por cable, electrónicos de consumo programable, máquinas de videojuegos, consolas de juego, unidades de juego portátiles o manuales, redes de PC, minicomputadores, computadores medios, nodos de red, ambientes de cómputo de multiprocesamiento o distribuidos que incluyen cualesquiera de los anteriores sistemas o dispositivos, algunas combinaciones de ellos, etc.

Las implementaciones para el diseño de los API y/o los API resultantes de ello pueden ser descritas en el contexto general de instrucciones ejecutables por procesador. Generalmente, las instrucciones ejecutables por procesador incluyen rutinas, programas, módulos, protocolos, objetos interfaces, componentes, estructuras de datos, etc. que llevan a cabo y/o permiten tareas particulares y/o implementar tipos de datos abstractos particulares. El diseño de los API y/o los API resultantes de ello, como se describió en ciertas implementaciones aquí, pueden también ser practicados y/o estar presentes en ambientes de procesamiento de distribuido en donde las tareas se llevan a cabo por dispositivos de procesamiento enlazados remotamente que

están conectados por medio de un enlace de comunicaciones y/o una red. Especialmente pero no exclusivamente en un ambiente de cómputo distribuido, las instrucciones ejecutables por procesador pueden estar localizadas en medios de almacenamiento separados, ejecutadas por diferentes procesadores, y/o propagadas por medios de transmisión.

El ambiente operativo ilustrativo 1000 incluye un dispositivo de cómputo de propósito general en la forma de un computador 1002, el cual puede comprender cualquier (por ejemplo electrónico) dispositivo con capacidades de cómputo/procesamiento. Los componentes del computador 1002 pueden incluir, pero no se limitan a, uno o más procesadores o unidades de procesamiento 1004, un sistema de memoria 1006, un bus del sistema 1008 que acopla los diversos componentes del sistema incluyendo el procesador 1004 a la memoria del sistema 1006.

Los procesadores 1004 no están limitados por los materiales a partir de los cuales ellos están formados o los mecanismos de procesamiento que emplean. Por ejemplo, los procesadores 1004 pueden comprender semiconductores y/o transistores (por ejemplo circuitos integrados

electrónicos (IC). En tal contexto, las instrucciones ejecutables por procesador pueden ser instrucciones ejecutables electrónicamente. Alternativamente, los mecanismos de o para los procesadores 1004, y por lo tanto, de o para el computador 1002, puede incluir pero no se limitan a, cómputo cuántico, cómputo óptico, cómputo mecánico (por ejemplo usando nanotecnología), etc.

El bus del sistema 1008 representa uno o más de cualquiera de los muchos tipos de estructuras de bus alambradas o inalámbricas, incluyendo un bus de memoria o controlador de memoria, una conexión punto a punto, un cableado plano de conmutación, un bus periférico, un punto gráfico acelerado, y un procesador o bus local usando cualquiera de una variedad de arquitecturas de uso. Por vía de ejemplo, tales arquitecturas pueden incluir una arquitectura de industria estándar (ISA) un bus de arquitectura de microcanal (MCA), un bus de ISA mejorada (EISA), un bus local de la asociación de estándar de video electrónicos (VESA), un bus de interconexión de componentes periféricos (PCI) también conocido como bus mezanine, algunas combinaciones de ellos, etc.

El computador 1002 incluye típicamente una variedad de medios accesibles por procesador. Tales medios pueden ser cualquiera de los medios disponibles que son accesibles por computador 1002 u otro dispositivos (por ejemplo electrónico), e incluye tanto medios volátiles como no volátiles, medios removibles y no removibles y medios de almacenamiento y transmisión.

La memoria del sistema 1006 incluye medios de almacenamiento accesibles por procesador en la forma de memoria volátil, tal como la memoria de acceso aleatorio (RAM) 1040, y/o memoria no volátil, tal como la memoria de solo lectura (ROM) 1012. un sistema básico de entrada/salida (BIOS) 1014, que contiene las rutinas básicas que ayudan a la transferencia de información entre los elementos dentro del computador 1002, tal como durante el inicio, típicamente almacenada en ROM 1012. La RAM 1010, típicamente contiene datos y/o programas módulos/instrucciones que son accesibles inmediatamente a y/o están actualmente bajo operación en o por medio de la unidad de procesamiento 1004.

El computador 1002 puede también incluir otros medios de almacenamiento removibles/no removibles y/o volátiles/no volátiles. Como vía de ejemplo, la figura 10 ilustra un dispositivo de disco duro o arreglo de disco duro 1016 para leer desde y escribir en (típicamente) medios magnético no removible, no volátil (no mostrado separadamente); un dispositivo de disco magnético 1018 para leer desde y escribir en (típicamente) removible, no volátil 1020 (por ejemplo un diskette); y un dispositivo de disco óptico 1022 para leer desde y/o escribir en (típicamente) removible, no volátil disco óptico 1024 tal como un CD, DVD u otro medio óptico. El dispositivo de disco duro 1016, el dispositivo de disco magnético 1018, y el dispositivo de disco óptico 1022 están conectados al bus del sistema 1008 por uno o más interfaces del medio de almacenamiento 1026. Alternativamente el dispositivo de disco duro 1016, el dispositivo de disco magnético 1018, y el dispositivo de disco óptico 1022 pueden estar conectados al bus del sistema 1008 por uno o más interfaces separadas o combinadas (no mostrado).

Los dispositivos de disco y sus medios accesibles por procesador asociados suministran almacenamiento no volátil de instrucciones ejecutables por procesador, tal

como estructuras de datos, módulos de programas u otros datos para el computador 1003. aunque el computador ilustrativo 1002 ilustra un disco duro 1016, un disco magnético removible 1020 y un disco óptico removible 1024, puede apreciarse que otros tipos de medios accesibles por procesador pueden almacenar instrucciones que son accesibles por un dispositivo, tal como casete magnético u otros dispositivos de almacenamiento magnéticos, memorias flash, discos compactos (CD), discos versátiles digitales (DVD) u otros almacenamientos ópticos, memorias RAM, ROM, programables, borrables eléctricamente o memorias de solo lectura (EEPROM), etc. Tales medios pueden también incluir los circuitos integrados · de alambrado fuerte para propósitos especiales. En otras palabras, cualquier medio accesible por procesador puede ser utilizado para realizar el medio de almacenamiento del ambiente operativo ilustrativo 1000.

Cualquier número de módulos de programa (u otras unidades o conjuntos de instrucciones/código, incluyendo una estructura API y/o objetos con base en ella) puede ser almacenada en el disco duro 1016, disco magnético 1020, disco óptico 1024, ROM 1012, y/o RAM 1040, incluyendo por

vía de ejemplo general, un sistema operativo 1028, uno o más programas de aplicación 1030, otros módulos de programa 1032, y datos de programa 1034. Un usuario puede ingresar comandos y/o información dentro del computador 1002 por medio de dispositivos de entrada tales como un teclado 1036 y un dispositivo de apuntamiento 1038 (por ejemplo un "ratón"). Otros dispositivos de entrada 1040 (no mostrados específicamente) pueden incluir un micrófono, joystick, almohadilla de juego, plato de satélite, puerto serial, escáner y/o similares. Estos y otros dispositivos de entrada están conectados a la unidad de procesamiento 1004 por medio de interfaces de entrada/salida 1042 y son acoplados al bus del sistema 1008. Sin embargo, los dispositivos de entrada y/o dispositivos de salida pueden en su lugar ser conectados por otras interfaces y estructuras de bus, tales como puerto paralelo, puerto de juegos, un puerto de bus serial universal (USB), un puerto infrarrojo, una interfaz inalámbrica IEEE 1394 (Firewire), una interfaz inalámbrica IEEE 802.11, una interfaz inalámbrica bluetooth®, etc.

Una pantalla monitor/de visualización 1044 u otro tipo de dispositivo de visualización pude también estar conectado

al bus del sistema 1008 por medio de una interfaz, tal como un adaptador de video 1046. El adaptador de video 1046 (u otro componente) puede ser o puede incluir una tarjeta de gráficos para procesamiento de cálculos intensivos de gráficos y para el manejo de requerimientos de visualización grandes. Típicamente, una tarjeta de gráficos incluye una unidad de procesamiento de gráficos (GPU), RAM de video (VRAM), etc. para facilitar la visualización rápida de gráficos y de las operaciones de los gráficos. En adición al monitor 1044, otros dispositivos periféricos de salida pueden incluir componentes tales como parlantes (no mostrados) y una impresora 1048, que puede estar conectada al computador 1002 por medio de interfaces de entrada/salida 1042.

El computador 1002 puede operar en un ambiente de red usando conexiones lógicas a uno o más computadores remotos, tal como un dispositivo de cómputo remoto 1050. Como vía de ejemplo, el dispositivo de cómputo remoto 1050 puede ser un computador personal, un computador portátil (por ejemplo un computador portátil, un computador de tableta, un PDA, una estación móvil, etc.), un computador de bolsillo o palm, un reloj, un dispositivo de juego, un servidor, un enrutador, un

computador de red, un dispositivo par, un nodo de red u otro tipo de dispositivo como se listó anteriormente, etc. Sin embargo, el dispositivo de cómputo remoto 1050 está ilustrado como un computador portátil que puede incluir muchos o todos los elementos y rasgos descritos aquí con respecto al computador 1002.

Las conexiones lógicas entre el computador 1002 y un computador remoto 1050 están ilustradas como una red de área local (LAN) 1052 y una red de área amplia general (WAN) 1054. Tales ambientes de red son muy comunes en oficinas, redes de computadores para empresas grandes, intranets, el Internet, redes de teléfonos fijos y móviles, redes inalámbricas ad hoc y de infraestructura, y otras redes inalámbricas, redes para juegos, algunas combinaciones de ellos, etc. Tales redes y conexiones de comunicaciones son ejemplos de medios de transmisión.

Cuando se ha implementado en un ambiente de red LAN, el computador 1002 está conectado usualmente a la LAN 1052 por medio de una interfaz o adaptador de red 1056. Cuando se ha implementado en un ambiente de red WAN, el computador 1002 típicamente incluye un módem 1058 u otro componente para establecer comunicaciones sobre la WAN

1054. El Módem 1058, que puede ser interno o externo al computador 1002, puede estar conectado al bus del sistema 1008 por medio de interfaces de entrada/salida 1042 o cualquier otro mecanismo apropiado. Debe apreciarse que las conexiones de red son ilustrativas y que otras maneras para establecer enlaces de comunicación entre los computadores 1002 y 1050 pueden ser empleados.

En un ambiente de red tal como aquel ilustrado con el ambiente operativo 1000, los módulos de programa u otras instrucciones que son ilustradas con relación al computador 1002, o porciones de ella pueden ser almacenadas completamente o parcialmente en un dispositivo de almacenamiento de medio remoto. Como vía de ejemplo, los programas de aplicación remotos 1060 residen en un componente de memoria del computador remoto 1050 pero pueden ser utilizables o de otra manera accesibles por medio del computador 1002. También, para propósitos de ilustración, los programas de aplicación 1030, u otras instrucciones ejecutables por procesador tal como el sistema operativo 1028 son ilustrados aquí como bloques discretos pero debe reconocerse que tales programas, componentes, u otras instrucciones residen en diversos tiempos en diferentes componentes de

almacenamiento del dispositivo de cómputo 1002 (y/o dispositivo de cómputo remoto 1050) y son ejecutados por el procesador 1004 del computador 1002 (y/o aquel del dispositivo de cómputo remoto 1050).

Aunque los sistemas, medios, dispositivos, métodos, procedimientos, aparatos, técnicas, API, esquemas, enseñanzas, procedimientos, disposiciones y otras implementaciones han sido descritas en un lenguaje específico a rasgos y/o diagramas estructurales, lógicos, algorítmicos, funcionales y con base en acción, debe entenderse que la invención definida en las reivindicaciones adjuntas no necesariamente está limitada a rasgos específicos o diagramas específicos descritos. En su lugar, los rasgos específicos y los diagramas están descritos como formas ilustrativas de implementación de la invención reivindicada.

REIVINDICACIONES

1. Un método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Preparar una pluralidad de muestras de código para un escenario principal, cada muestra de código respectiva de una pluralidad de muestras de código correspondientes a un lenguaje de programación respectivo de una pluralidad de lenguajes de programación;

Derivar el API del escenario principal en respuesta a la pluralidad de muestras de código.

2. El método de acuerdo con la reivindicación 1, que comprende adicionalmente:

Determinar, por un diseñador de API, si el API derivado es muy complejo.

3. El método de acuerdo con la reivindicación 2, que comprende adicionalmente:

Si el API derivado está determinado que es complejo entonces

Refinar, por el diseñador del API, el API derivado para producir un API refinado.

4. El método de acuerdo con la reivindicación 3, que comprende adicionalmente:

Determinar, por el diseñador de API, si el API refinado es muy complejo.

5. El método según la reivindicación 1, que comprende adicionalmente:

Llevar a cabo uno o más estudios de utilización en el API utilizando una pluralidad de desarrolladores.

6. El método de acuerdo con la reivindicación 5, en donde el llevar a cabo comprende:

Llevar a cabo uno o más estudios de utilización sobre el API utilizando la pluralidad de desarrolladores en donde la pluralidad de desarrolladores son típicos para la pluralidad de lenguajes de programación.

7. El método según la reivindicación 5, que comprende adicionalmente:

Asegurar si la pluralidad de desarrolladores son capaces de usar el API sin problemas significativos.

8. El método según la reivindicación 7, que comprende adicionalmente:

Si la pluralidad de desarrolladores no está asegurada como capaz de usar el API sin problemas significativos, entonces revisar el API.

9. El método según la reivindicación 8, en donde la revisión comprende:

Revisar el API con base en por lo menos una lección entre el uno o más estudios de utilización.

10. El método según la reivindicación 1, en donde la derivación comprende:

Derivar el API para soportar la pluralidad de muestras de código que corresponden respectivamente a la pluralidad de lenguajes de programación.

11. El método según la reivindicación 1, en donde la derivación comprende:

Recoger los mandatos específicos del lenguaje desde la pluralidad de muestras de código; e

Incorporar los mandatos específicos del lenguaje en el API.

12. El método según la reivindicación 1, en donde la derivación comprende:

Recoger las expectativas del desarrollador inspirado en lenguaje desde una pluralidad de muestras de código; e

Incorporar las expectativas del desarrollador inspiradas en lenguaje en el API.

13. El método según la reivindicación 1, en donde la derivación comprende:

Recoger cosas comunes entre una pluralidad de muestras de código; e

Incorporar las cosas comunes en el API.

14. El método según la reivindicación 1, en donde la reivindicación comprende:

Derivar el API para tener un componente agregado que ate una pluralidad de tipos factorizados de bajo nivel juntos para soportar el escenario principal.

15. Un método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Seleccionar un escenario principal para un área de rasgos;

Escribir por lo menos una muestra de código para el escenario principal; y

Derivar un API para el escenario principal en respuesta al por lo menos una muestra de código.

16. El método según la reivindicación 15, en donde la selección comprende:

Seleccionar una pluralidad de escenarios principales para el área de rasgos.

17. El método según la reivindicación 16, que comprende adicionalmente:

Repetir la escritura y derivar para cada escenario principal de una pluralidad de escenarios principales que son seleccionados para el área de rasgos.

18. El método según la reivindicación 15, en donde la escritura comprende:

Escribir una pluralidad de muestras de código para el escenario principal, cada muestra de código respectiva de la pluralidad de muestras de código corresponde a un lenguaje de programación respectivo entre una pluralidad de lenguajes de programación.

19. El método según la reivindicación 18, en donde la derivación comprende:

Derivar el API del escenario principal en respuesta a la pluralidad de muestras de código.

20. El método según la reivindicación 15, que comprende adicionalmente:

Llevar a cabo uno o más estudios de utilización sobre el API utilizando una pluralidad de desarrolladores;

Hacer seguro si la pluralidad de desarrolladores son capaces de usar el API sin problemas significativos; y

Si la pluralidad de desarrolladores no son asegurados como capaces de usar el API sin problemas significativos, entonces revisar el API.

21. El método según la reivindicación 20, en donde la revisión comprende:

Revisar el API con base en por lo menos una lección entre los uno o más estudios de utilización para producir un API revisado.

22. El método según la reivindicación 21, que comprende adicionalmente:

Repetir el llevar a cabo y el aseguramiento con respecto al API revisado.

23. El método según la reivindicación 15, en donde la derivación comprende:

derivar el API para soportar el por lo menos una muestra de código escrita para el escenario principal por medio de producir un API de dos capas que incluye un componente agregado y una pluralidad de tipos factorizados subyacentes.

24. El método según la reivindicación 15, en donde la derivación comprende:

Recoger uno o más mandatos específicos del lenguaje a partir de por lo menos una muestra de código; y

Incorporar el uno o más mandatos específicos del lenguaje en el API.

25. El método según la reivindicación 15, en donde la derivación comprende:

Encapsular un tipo factorizado particular en un componente agregado que está asociado con el escenario principal si todos los miembros del tipo factorizado particular están expuestos por el componente agregado.

26. El método según la reivindicación 15, en donde la derivación comprende:

Encapsular un tipo factorizado particular dentro de un componente agregado que está asociado con el escenario principal si el tipo factorizado particular no es interesante de manera independiente para otros tipos de componentes.

27. El método según la reivindicación 15, en donde la derivación comprende:

Exponer un tipo factorizado particular desde un componente agregado que está asociado con el escenario principal si por lo menos un miembro

del tipo factorizado particular no está expuesto por el componente agregado.

28. El método según la reivindicación 15, en donde la derivación comprende:

Exponer un tipo factorizado particular a partir de un componente agregado que está asociado con el escenario principal si el tipo factorizado particular puede ser benéficamente usado independientemente del componente agregado por otro tipo de componente.

29. El método según la reivindicación 15, en donde la derivación comprende:

Producir una estructura de dos capas que incluye tipos de componente que objetiva un nivel superior relativamente de abstracción y tipos de componentes que objetivan unible interior relativamente de abstracción.

30. El método según la reivindicación 29, en donde los tipos de componente que objetivan el mayor

nivel relativamente de abstracción están dirigidos a escenarios principales.

31, El método según la reivindicación 29, en donde los tipos de componentes que objetivan el nivel más bajo relativamente de abstracción suministran una cantidad relativamente más grande de control a los desarrolladores en comparación con los tipos de componentes que objetivan el nivel más alto relativamente de abstracción.

32. El método según la reivindicación 15, en donde la reivindicación comprende:

Derivar el API de modo que se le permite a un desarrollador implementar un patrón de crear-establecer-llamar para uso en el escenario principal.

33. El método según la reivindicación 32, en donde la derivación comprende:

Producir el API con parámetros preseleccionados que son apropiados para el escenario principal

34. Un método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Derivar un API de un escenario en respuesta a por lo menos una muestra de código escrita con respecto al escenario:

Llevar a cabo uno o más estudios de utilización sobre el API utilizando una pluralidad de desarrolladores; y

Revisar el API con base en el uno o más estudios de utilización.

35. El método según la reivindicación 34, que comprende adicionalmente:

Escribir una pluralidad de muestras de código con respecto al escenario, cada muestra de código respectiva de la pluralidad de muestras

de código corresponde un lenguaje de programación respectivote una pluralidad de lenguajes de programación;

En donde la derivación comprende:

Derivar el API para el escenario en respuesta a la pluralidad de muestras de código.

36. El método según la reivindicación 34, que comprende adicionalmente, antes de llevar a cabo uno o más estudios de utilización sobre el API:

Determinar, por un diseñador de API, si el API derivado es muy complejo;

Si el API derivado está determinado que es muy complejo, entonces

Refinar, por el diseñador del API, el API derivado para producir un API refinado; y

Determinar, por el diseñador del API, si el API refinado es demasiado complejo.

37. El método según la reivindicación 34, que comprende adicionalmente:

Asegurar si la pluralidad de desarrolladores son capaces de usar el API sin problemas significativos; y

Cuando la pluralidad de desarrolladores no se asegura de ser capaz de usar el API sin problemas significativos, entonces implementar la revisión;

En donde la revisión comprende:

Revisar el API con base en por lo menos una lección entre el uno o más estudios de utilización para producir un API revisado.

38. El método según la reivindicación 37, que comprende adicionalmente:

Repetir por lo menos el llevar a cabo y el aseguramiento con respecto al API revisado.

39. El método según la reivindicación 37, en donde el aseguramiento comprende: asegurar si la pluralidad de desarrolladores son capaces de usar el API sin problemas significativos con respecto a un nivel deseado de utilización para por lo menos un grupo desarrollador objetivado, en donde el nivel deseado de utilización incluye consideraciones con respecto a (i) referencia frecuente y/o extensiva a documentación API detallada por la pluralidad de desarrolladores, (ii) una falla en una mayoría de la pluralidad de desarrolladores para implementar el escenario, y (iii) si la pluralidad de desarrolladores toma un acercamiento que es significativamente diferente de aquel que es esperado por un diseñador API.

40. El método según la reivindicación 34, comprende adicionalmente:

seleccionar una pluralidad de escenarios principales para un área de rasgos;

Repetir la derivación, el llevar a cabo, y la revisión para cada uno de los escenarios de la pluralidad de escenarios principales.

41. El método según la reivindicación 40, en donde la derivación comprende:

Producir un componente agregado para cada escenario de principal de la pluralidad de escenarios principales.

42. El método según la reivindicación 34, en donde la derivación comprende:

Producir un componente agregado que tiene una relación respectiva con cada uno de los tipos factorizados respectivos de una pluralidad de tipos factorizados.

43. El método según la reivindicación 42, en donde el producir comprende:

Producir el componente agregado para soportar el escenario para el cual el por lo menos una muestra de código se escribió.

44. El método según la reivindicación 42, en donde el producir comprende:

Producir el componente agregado para tener una relación expuesta con por lo menos un tipo factorizado de la pluralidad de tipos factorizados y una relación encapsulada con por lo menos otro tipo factorizado de la pluralidad de tipos factorizados.

45. El método según la reivindicación 44, en donde el por lo menos otro tipo factorizado de la pluralidad de tipos factorizados que tiene la relación encapsulada del componente agregado puede ser cedido por el componente agregado para interacción directa con otro tipo de componente.

46. El método según la reivindicación 42, en donde la pluralidad de tipos factorizados están diseñados usando una metodología orientada a objetos.

47. Un método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Preparar una pluralidad de muestras de código para un escenario principal, cada muestra de código respectiva de la pluralidad de muestras de código corresponde a un lenguaje de programación respectivo de entre una pluralidad de lenguajes de programación;

Derivar el API para el escenario principal en respuesta a la pluralidad de muestras de código;

Llevar a cabo uno o más estudios de utilización sobre el API utilizando una pluralidad de desarrolladores; y

Revisar el API con base en el uno o más estudios de utilización.

48. El método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Escribir por lo menos una muestra de código para un escenario; y

Derivar un API para el escenario en respuesta a por lo menos una muestra de código, el API incluye (i) un componente agregado que está adaptado para facilitar la implementación del escenario y (ii) una pluralidad de tipos factorizados que suministran funcionalidad subyacente para el componente agregado, el API permite una progresión gradual de usar el componente agregado en situaciones más simples para usar una porción incremental de la pluralidad de tipos factorizados en situaciones incrementalmente complejas.

49. Un método para diseñar una interfaz de programación de aplicaciones (API), el método comprende:

Derivar por lo menos un componente agregado para soportar por lo menos una muestra de código para por lo menos un escenario;

Determinar requerimientos adicionales con respecto al por lo menos un escenario;

Decidir si los requerimientos adicionales pueden ser agregados al por lo menos un componente agregado son agregar complejidad indebida al por lo menos un escenario; y

Si no, definir una pluralidad de tipos factorizados en respuesta a la decisión.

50. El método según la reivindicación 49, comprende adicionalmente:

Si es así, refinar el por lo menos un componente agregado para incorporar los requerimientos adicionales.

51. El método según la reivindicación 49, comprende adicionalmente:

Seleccionar una pluralidad de escenarios principales para un área de rasgos, la pluralidad de escenarios principales incluye el por lo menos un escenario; y

Escribir una pluralidad de muestras de código que muestran líneas preferidas de código para la pluralidad de escenarios principales, la pluralidad de muestras de código incluyen el por lo menos una muestra de código;

En donde la derivación comprende: derivar una pluralidad de componentes agregados, que incluyen el por lo menos un componente agregado, para soportar la pluralidad de muestras de código para la pluralidad de escenarios principales.

52. El método según la reivindicación 49 en donde la derivación comprende:

Derivar el por lo menos un componente agregado con métodos apropiados, opciones por defecto y abstracciones para soportar el por lo menos una muestra de código para el por lo menos un escenario.

53. El método según la reivindicación 49, comprende adicionalmente:

Refinar el por lo menos una muestra de código de acuerdo con por lo menos un componente agregado; y

Evaluar la por lo menos una muestra de código refinada con respecto a la simplicidad; y

Repetir la derivación si la por lo menos muestra de código refinada falla en cuanto a ser suficientemente simple en la evaluación.

54. El método según la reivindicación 49, en donde la determinación comprende:

Determinar requerimientos adicionales con respecto al por lo menos un escenario, en donde los requerimientos adicionales incluyen escenarios adicionales, usos adicionales, e interacciones adicionales con otros tipos de componentes.

55. El método según la reivindicación 49, en donde la decisión comprende:

Considerar si se agrega los requerimientos adicionales al por lo menos un componente agregado esconde un patrón de uso de crear-establecer-llamar.

56. El método según la reivindicación 49, en donde la definición comprende:

Definir la pluralidad de tipos factorizados en respuesta a la decisión con una factorización

ideal de un conjunto completo de funcionalidades.

57. El método según la reivindicación 49, en donde la definición comprende:

Definir la pluralidad de tipos factorizados en respuesta a la decisión de usar una o más metodologías orientadas a objetos.

58. El método según la reivindicación 49, comprende adicionalmente:

Determinar si el por lo menos un componente agregado es para encapsular o exponer la funcionalidad de cada tipo factorizado de la pluralidad de tipos factorizados.

59. El método según la reivindicación 49, comprende adicionalmente:

Refinar la pluralidad de tipos factorizados para soportar el por lo menos un componente agregado y los requerimientos adicionales.

199

RESUMEN

Una primera implementación de un método ilustrativo para diseñar una interfaz de programación de aplicaciones (API) incluye: preparar mezclas de códigos múltiples para un escenario principal, cada muestra de código respectiva de las muestras de código múltiples corresponde a un lenguaje de programación respectivo de entre lenguajes de programación múltiples; y derivar el API del escenario principal es respuesta a las múltiples muestras de código. Un segundo método ilustrativo para diseñar un API incluye: seleccionar un escenario principal para un área de rasgos; escribir por lo menos una muestra de código para el escenario principal, y derivar un API para el escenario principal en respuesta al por lo menos una muestra de código. Un tercer método ilustrativo para diseñar un API incluye: derivar un API para un escenario en respuesta a por lo menos una muestra de código escrita con respecto al escenario; llevara cabo uno o más estudios de utilización sobre el API utilizando múltiples desarrolladores; y revisar el API con base en el uno o más estudios de utilización.

200

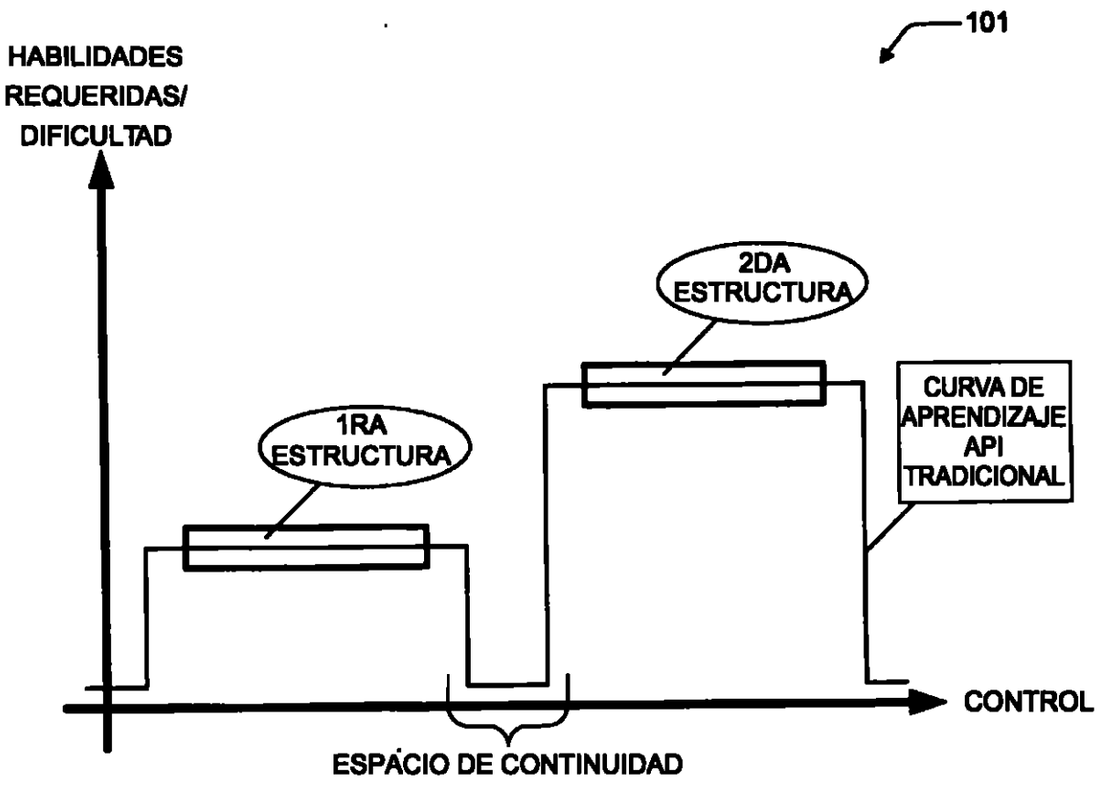


FIG. 1 **Estado de la Técnica**

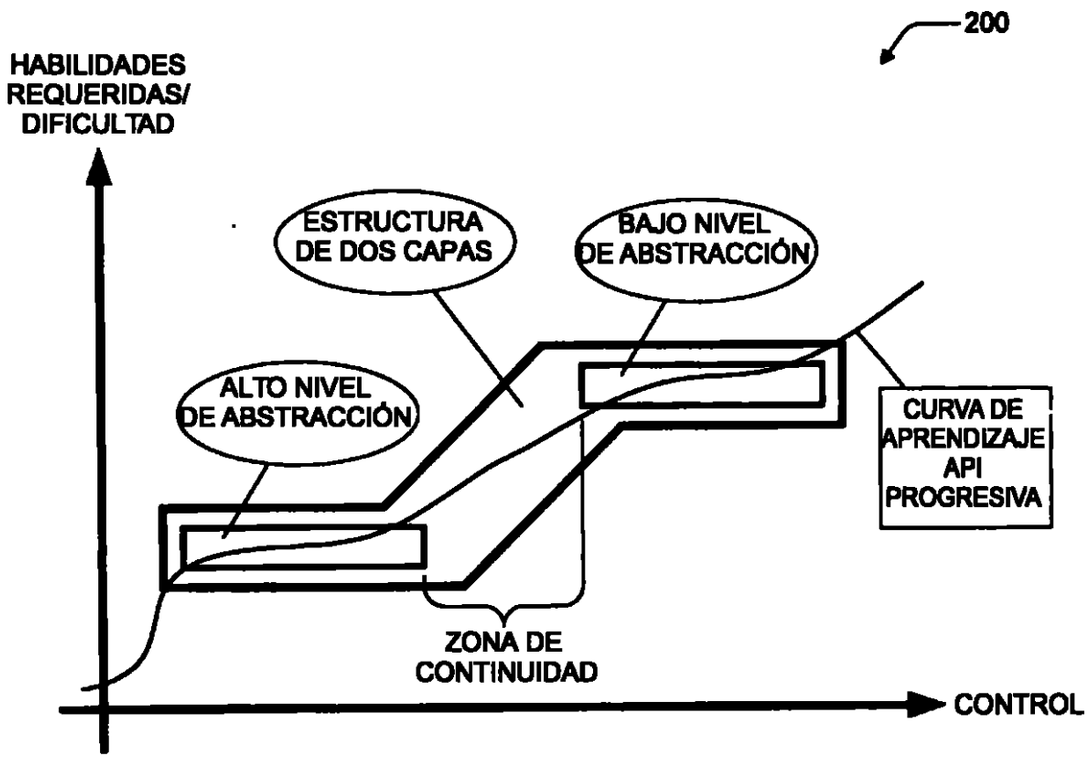

200
HABILIDADES REQUERIDAS/ DIFICULTAD
ESTRUCTURA DE DOS CAPAS
BAJO NIVEL DE ABSTRACCIÓN
ALTO NIVEL DE ABSTRACCIÓN
CURVA DE APRENDIZAJE API PROGRESIVA
ZONA DE CONTINUIDAD
CONTROL

FIG. 2

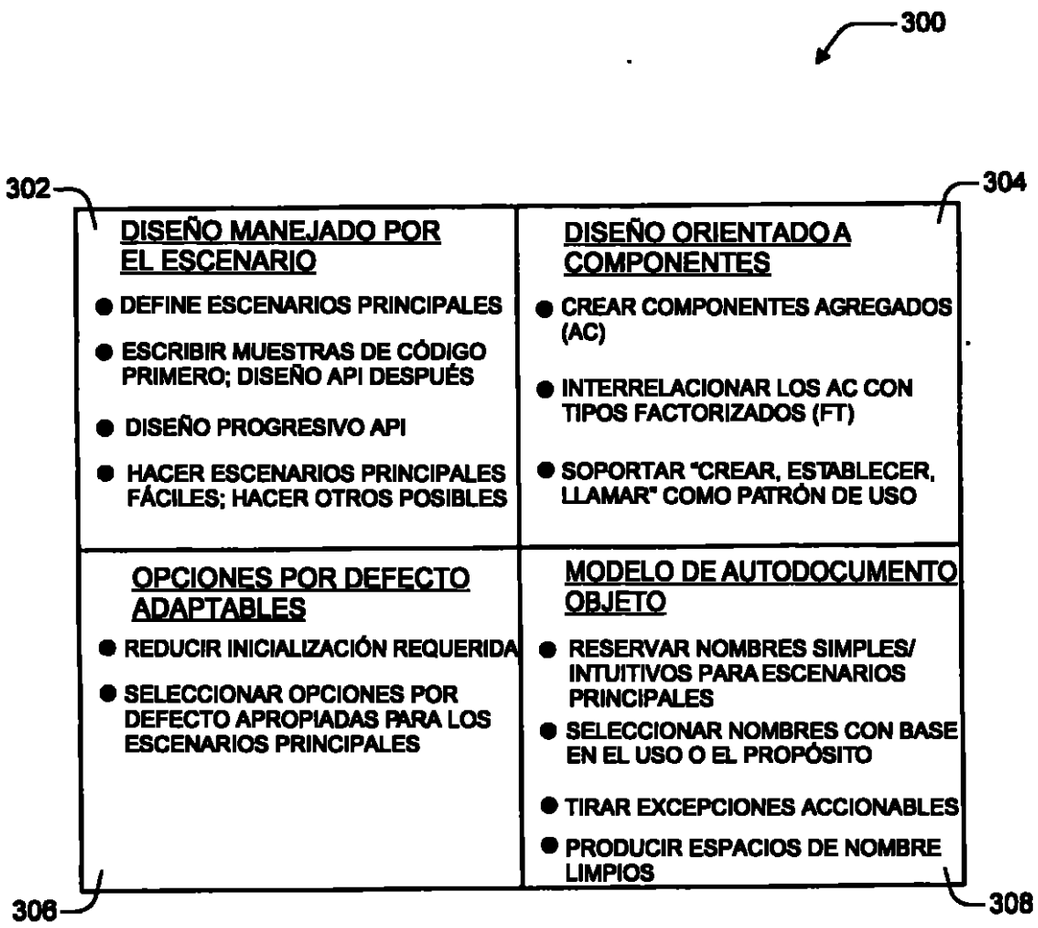

300
302
304
DISEÑO MANEJADO POR EL ESCENARIO
DEFINE ESCENARIOS PRINCIPALES
ESCRIBIR MUESTRAS DE CÓDIGO PRIMERO; DISEÑO API DESPUÉS
DISEÑO PROGRESIVO API
HACER ESCENARIOS PRINCIPALES FÁCILES; HACER OTROS POSIBLES
DISEÑO ORIENTADO A COMPONENTES
CREAR COMPONENTES AGREGADOS (AC)
INTERRELACIONAR LOS AC CON TIPOS FACTORIZADOS (FT)
SOPORTAR "CREAR, ESTABLECER, LLAMAR" COMO PATRÓN DE USO
OPCIONES POR DEFECTO ADAPTABLES
REDUCIR INICIALIZACIÓN REQUERIDA
SELECCIONAR OPCIONES POR DEFECTO APROPIADAS PARA LOS ESCENARIOS PRINCIPALES
MODELO DE AUTODOCUMENTO OBJETO
RESERVAR NOMBRES SIMPLES/ INTUITIVOS PARA ESCENARIOS PRINCIPALES
SELECCIONAR NOMBRES CON BASE EN EL USO O EL PROPÓSITO
TIRAR EXCEPCIONES ACCIONABLES
PRODUCIR ESPACIOS DE NOMBRE LIMPIOS
306
308

FIG. 3

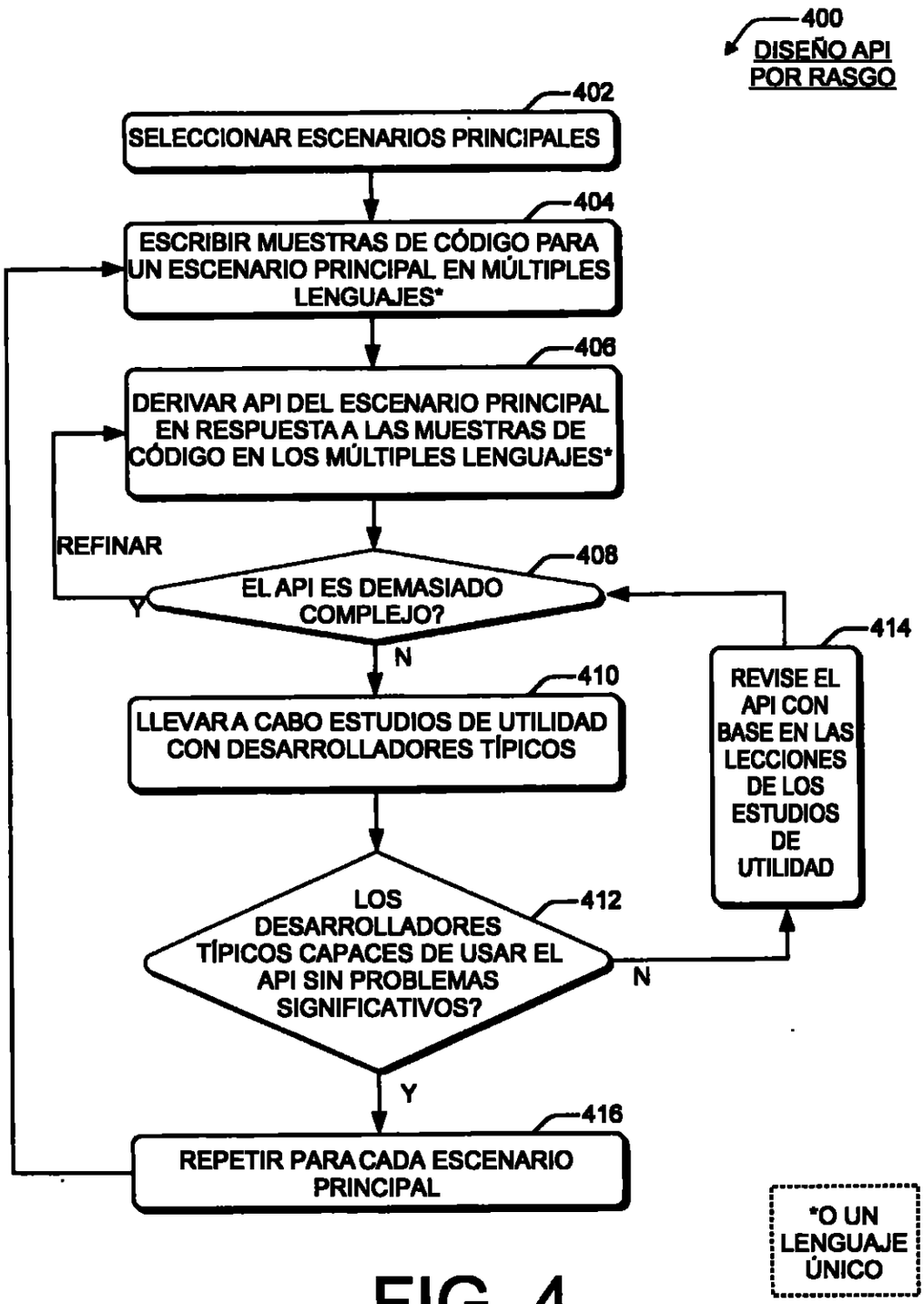

400
DISEÑO API POR RASGO
402
SELECCIONAR ESCENARIOS PRINCIPALES
404
ESCRIBIR MUESTRAS DE CÓDIGO PARA UN ESCENARIO PRINCIPAL EN MÚLTIPLES LENGUAJES*
406
DERIVAR API DEL ESCENARIO PRINCIPAL EN RESPUESTA A LAS MUESTRAS DE CÓDIGO EN LOS MÚLTIPLES LENGUAJES*
REFINAR
408
EL API ES DEMASIADO COMPLEJO?
Y
N
410
LLEVAR A CABO ESTUDIOS DE UTILIDAD CON DESARROLLADORES TÍPICOS
414
REVISE EL API CON BASE EN LAS LECCIONES DE LOS ESTUDIOS DE UTILIDAD
412
LOS DESARROLLADORES TÍPICOS CAPACES DE USAR EL API SIN PROBLEMAS SIGNIFICATIVOS?
N
Y
416
REPETIR PARA CADA ESCENARIO PRINCIPAL
*O UN LENGUAJE ÚNICO

FIG. 4

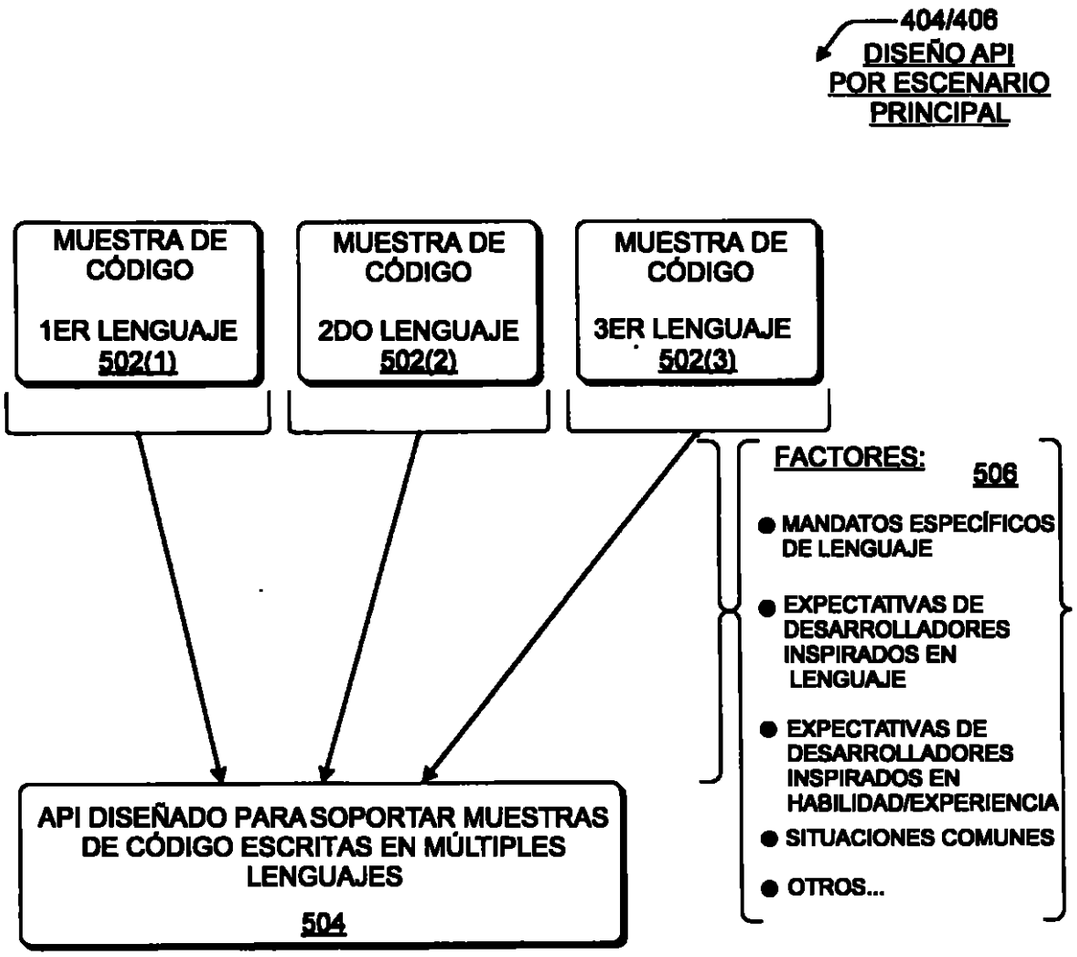

404/406
DISEÑO API
POR ESCENARIO
PRINCIPAL
MUESTRA DE CÓDIGO
1ER LENGUAJE
502(1)
MUESTRA DE CÓDIGO
2DO LENGUAJE
502(2)
MUESTRA DE CÓDIGO
3ER LENGUAJE
502(3)
FACTORES:
506
MANDATOS ESPECÍFICOS DE LENGUAJE
EXPECTATIVAS DE DESARROLLADORES INSPIRADOS EN LENGUAJE
EXPECTATIVAS DE DESARROLLADORES INSPIRADOS EN HABILIDAD/EXPERIENCIA
SITUACIONES COMUNES
OTROS...
API DISEÑADO PARA SOPORTAR MUESTRAS DE CÓDIGO ESCRITAS EN MÚLTIPLES LENGUAJES
504

FIG. 5

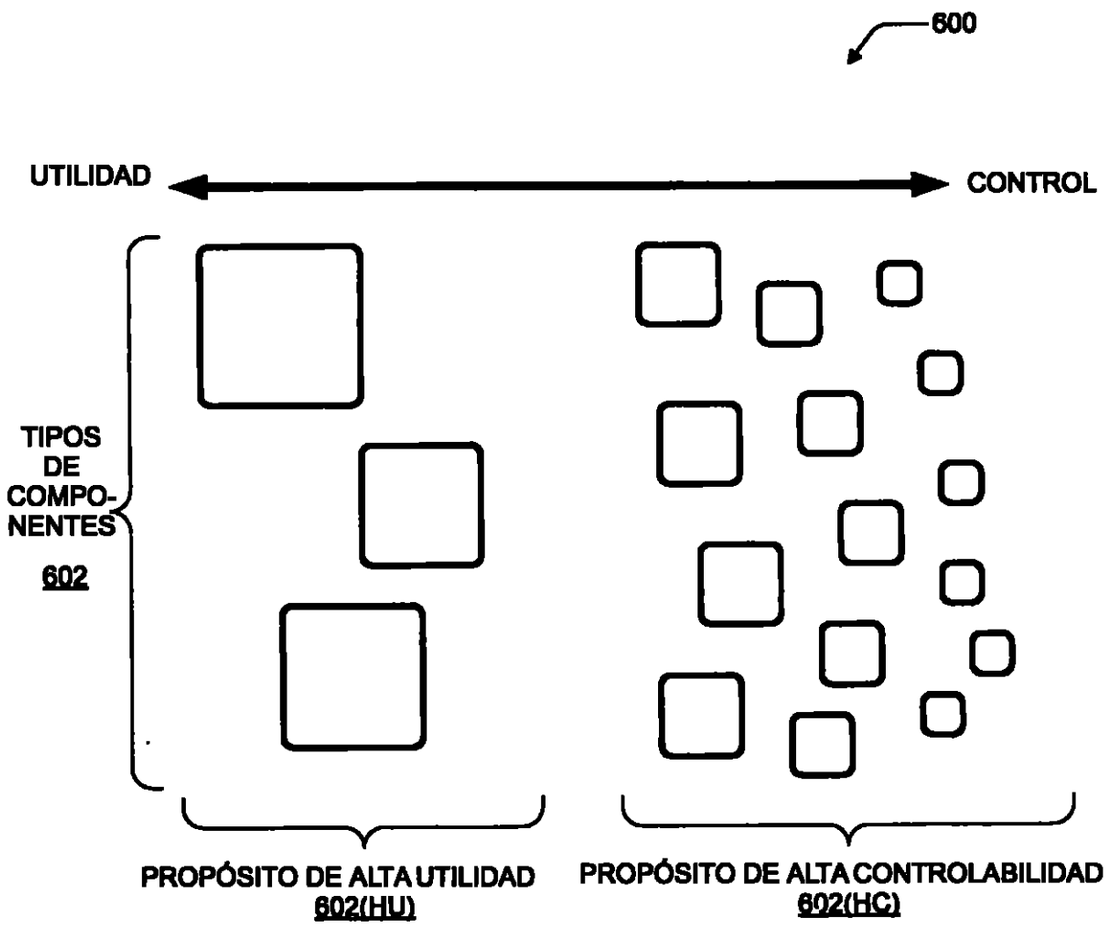

600
UTILIDAD
CONTROL
TIPOS DE COMPO-NENTES
602
PROPÓSITO DE ALTA UTILIDAD
602(HU)
PROPÓSITO DE ALTA CONTROLABILIDAD
602(HC)

FIG. 6

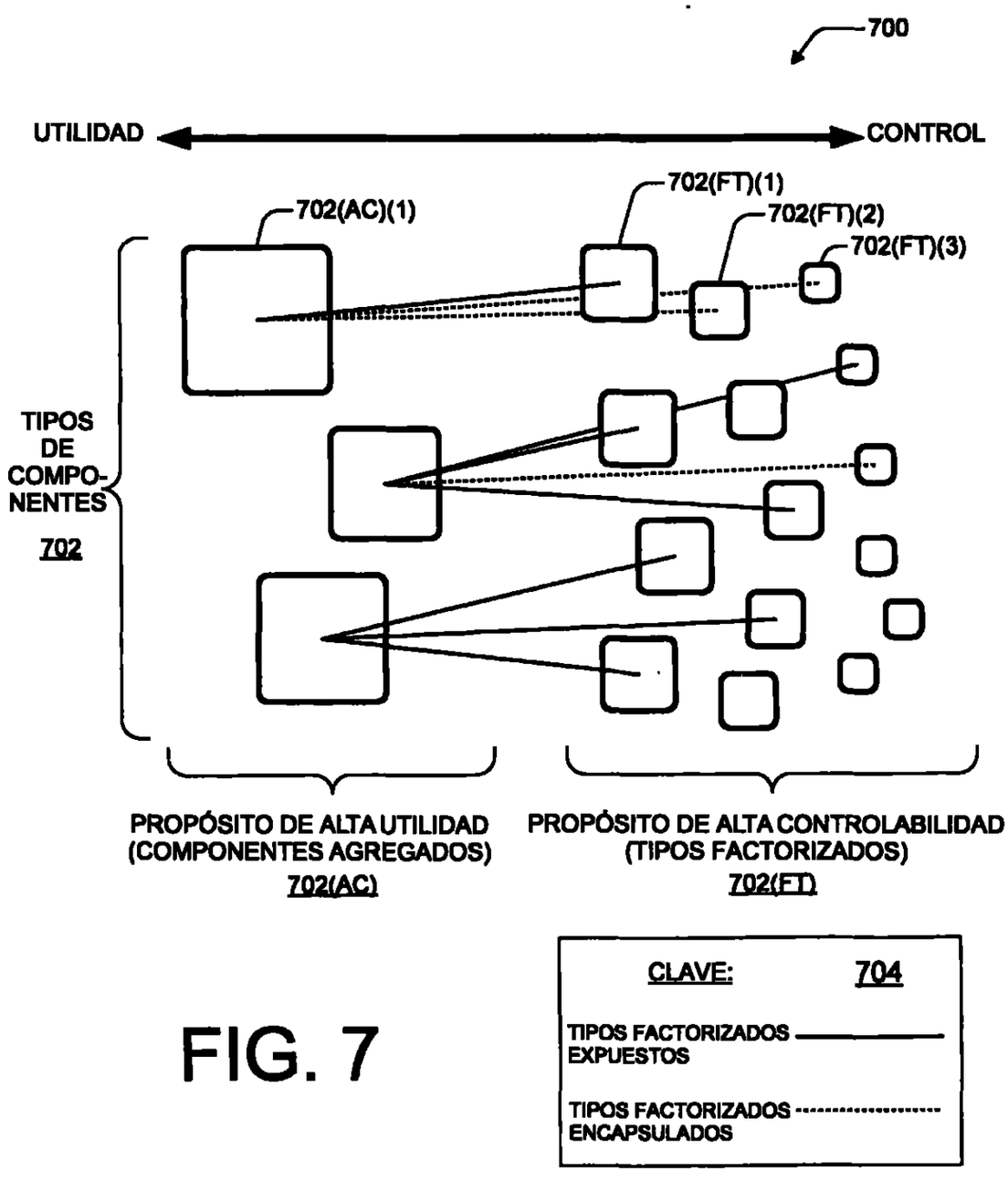

700
UTILIDAD
CONTROL
702(AC)(1)
702(FT)(1)
702(FT)(2)
702(FT)(3)
TIPOS DE COMPO-NENTES
702
PROPÓSITO DE ALTA UTILIDAD (COMPONENTES AGREGADOS)
702(AC)
PROPÓSITO DE ALTA CONTROLABILIDAD (TIPOS FACTORIZADOS)
702(FT)
CLAVE: 704
TIPOS FACTORIZADOS EXPUESTOS
TIPOS FACTORIZADOS ENCAPSULADOS

FIG. 7

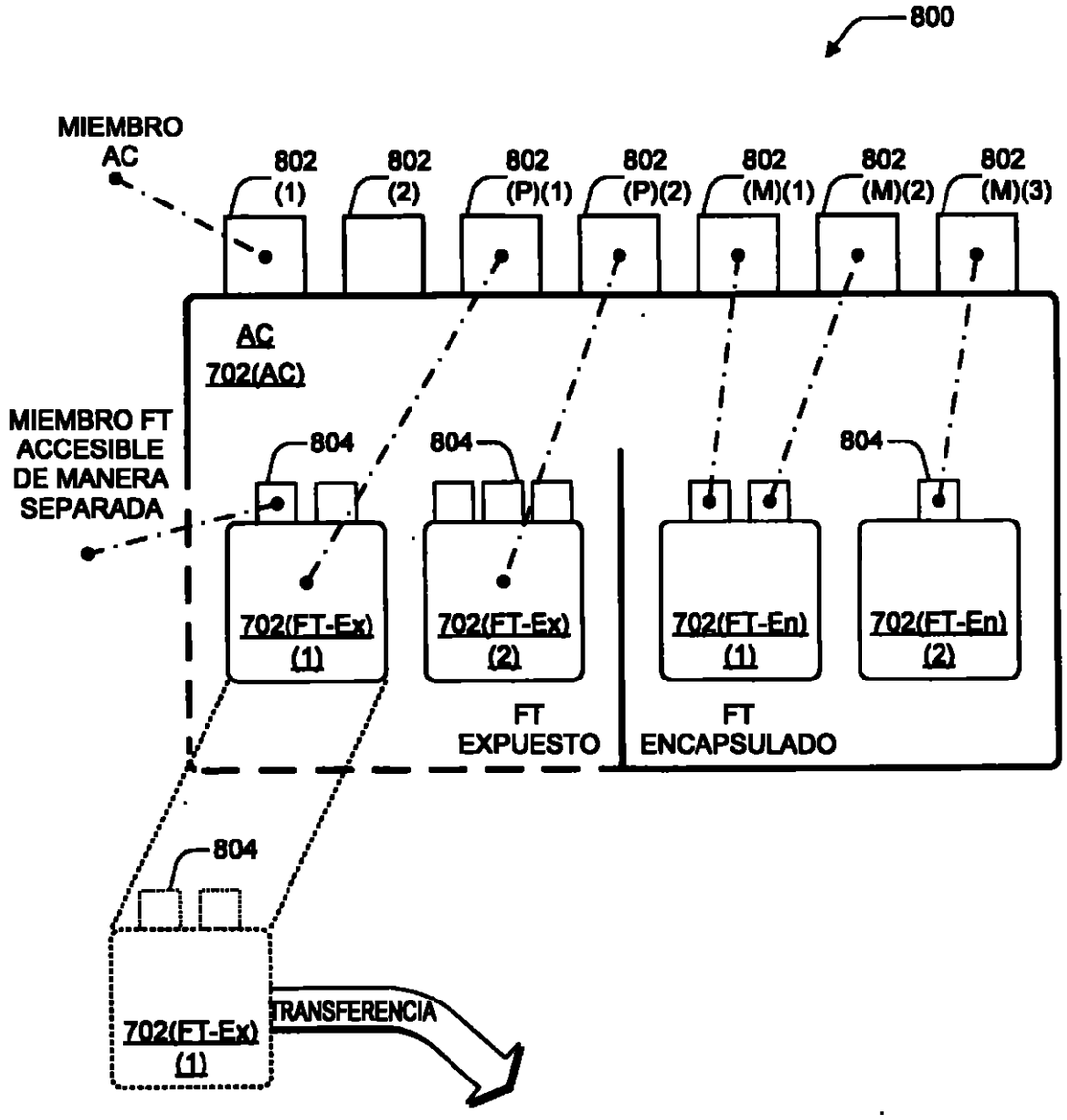

800
MIEMBRO AC
802 (1)
802 (2)
802 (P)(1)
802 (P)(2)
802 (M)(1)
802 (M)(2)
802 (M)(3)
AC
702(AC)
MIEMBRO FT ACCESIBLE DE MANERA SEPARADA
804
804
804
804
702(FT-Ex) (1)
702(FT-Ex) (2)
702(FT-En) (1)
702(FT-En) (2)
FT EXPUESTO
FT ENCAPSULADO
702(FT-Ex) (1)
TRANSFERENCIA

FIG. 8

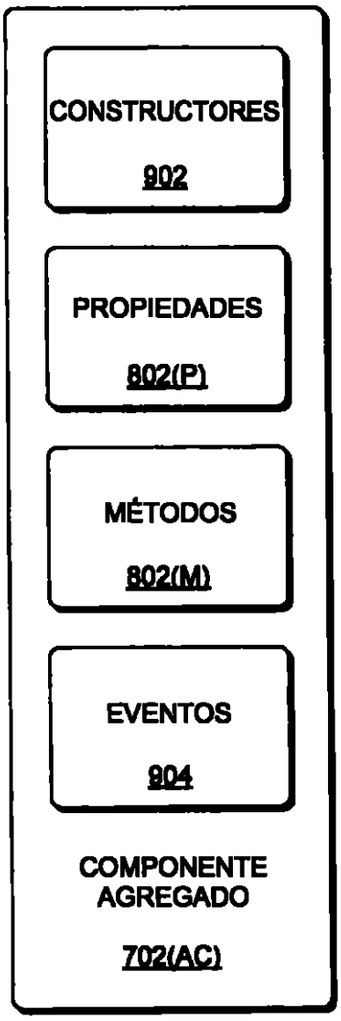

CONSTRUCTORES
902
PROPIEDADES
802(P)
MÉTODOS
802(M)
EVENTOS
904
COMPONENTE AGREGADO
702(AC)

FIG. 9

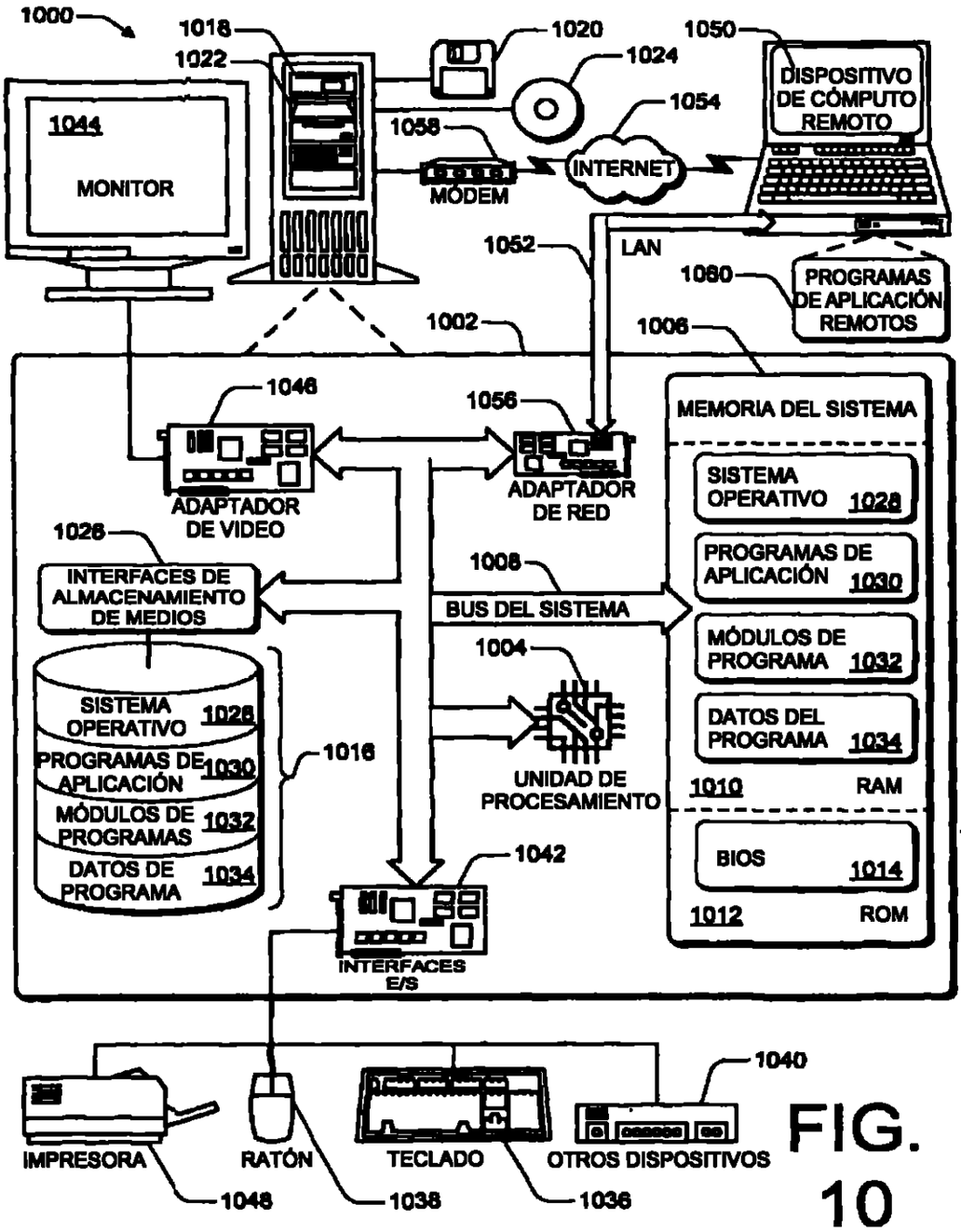

1000
1018
1022
1020
1024
1050
DISPOSITIVO DE CÓMPUTO REMOTO
1044
MONITOR
1058
MÓDEM
1054
INTERNET
1052
LAN
1060
PROGRAMAS DE APLICACIÓN REMOTOS
1002
1006
1046
ADAPTADOR DE VIDEO
1056
ADAPTADOR DE RED
MEMORIA DEL SISTEMA
SISTEMA OPERATIVO 1028
PROGRAMAS DE APLICACIÓN 1030
MÓDULOS DE PROGRAMA 1032
DATOS DEL PROGRAMA 1034
1010 RAM
BIOS 1014
1012 ROM
1026
INTERFACES DE ALMACENAMIENTO DE MEDIOS
1008
BUS DEL SISTEMA
1004
UNIDAD DE PROCESAMIENTO
SISTEMA OPERATIVO 1028
PROGRAMAS DE APLICACIÓN 1030
MÓDULOS DE PROGRAMAS 1032
DATOS DE PROGRAMA 1034
1016
1042
INTERFACES E/S
1040
IMPRESORA
RATÓN
TECLADO
OTROS DISPOSITIVOS
1048
1038
1036

FIG. 10

210

PA 1195444

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

July 20, 2004

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

APPLICATION NUMBER: *10/692,320*
FILING DATE: *October 23, 2003*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

P. R. GRANT
Certifying Officer

211

211

19085 U.S.PTO 10/23/03

PTO/SB/05 (03-01)
Approved for use through 10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Please type a plus sign (+) inside this box → +

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	MS1-1748US
First Inventor	Cwalina
Title	Design of Application Programming Interfaces (APIs)
Express Mail Label No.	EV355227055

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

1. ☐ Fee Transmittal Form (e.g., PTO/SB/17) *(Submit an original and a duplicate for fee processing)*
2. ☐ Applicant claims small entity status. See 37 CFR 1.27.
3. ☑ Specification [Total Pages 82] *(preferred arrangement set forth below)*
   - Descriptive title of the invention
   - Cross Reference to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to sequence listing, a table, or a computer program listing appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings *(if filed)*
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
4. ☑ Drawing(s) *(35 U.S.C. 113)* [Total Sheets 10]
5. Oath or Declaration [Total Pages]
   a. ☐ Newly executed (original or copy)
   b. ☐ Copy from a prior application (37 CFR 1.63 (d)) *(for continuation/divisional with Box 18 completed)*
      i. ☐ DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
6. ☐ Application Data Sheet. See 37 CFR 1.76

ADDRESS TO:

Mail Stop Patent Application
Commissioner for Patents/P.O. Box 1450
Alexandria, VA 22313-1450

7. ☐ CD-ROM or CD-R in duplicate, large table or Computer Program (Appendix)
8. Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
   a. ☐ Computer Readable Form (CRF)
   b. Specification Sequence Listing on:
      i. ☐ CD-ROM or CD-R (2 copies); or
      ii. ☐ paper
   c. ☐ Statements verifying identity of above copies

22386 U.S.PTO 10/692320 10/23/03

ACCOMPANYING APPLICATION PARTS

9. ☐ Assignment Papers (cover sheet & document(s))
10. ☐ 37 CFR 3.73(b) Statement *(when there is an assignee)* ☐ Power of Attorney
11. ☐ English Translation Document *(if applicable)*
12. ☐ Information Disclosure Statement (IDS)/PTO-1449 ☐ Copies of IDS Citations
13. ☐ Preliminary Amendment
14. ☑ Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
15. ☐ Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
16. ☐ Nonpublication Request under 35 U.S.C. 122 (b)(2)(B)(i). Applicant must attach form PTO/SB/35 or its equivalent.
17. ☐ Other: .....

18. If a CONTINUING APPLICATION, *check appropriate box, and supply the requisite information below and in a preliminary amendment, or in an Application Data Sheet under 37 CFR 1.76:*

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: ____/____

Prior application information: Examiner _____ Group Art Unit: _____

For CONTINUATION OR DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 5b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

19. CORRESPONDENCE ADDRESS

☑ Customer Number or Bar Code Label 22801 or ☐ Correspondence address below

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Name (Print/Type)	Keith W. Saunders	Registration No. (Attorney/Agent)	41,462
Signature	Keith W. Saunders	Date	10/23/2003

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

212

EV355227055 212

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Design of Application Programming Interfaces (APIs)

Inventor(s):

Krzysztof J. Cwalina
Bradley M. Abrams
Anthony J. Moore
Christopher L. Anderson
Michael J. Pizzo
Robert A. Brigham II

ATTORNEY'S DOCKET NO. MS1-1748US

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPL NO.	FILING OR 371 (c) DATE	ART UNIT	FIL FEE REC'D	ATTY.DOCKET NO	DRAWINGS	TOT CLMS	IND CLMS
10/692,320	10/23/2003	2127	0.00	MS1-1748US	10	59	6

CONFIRMATION NO. 8610

22801
LEE & HAYES PLLC
421 W RIVERSIDE AVENUE SUITE 500
SPOKANE, WA 99201

FILING RECEIPT

"OC000000011732361"

Date Mailed: 01/22/2004

Receipt is acknowledged of this regular Patent Application. It will be considered in its order and you will be notified as to the results of the examination. Be sure to provide the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION when inquiring about this application. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. If an error is noted on this Filing Receipt, please write to the Office of Initial Patent Examination's Filing Receipt Corrections, facsimile number 703-746-9195. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections (if appropriate).

Applicant(s)

Krzysztof J. Cwalina, Residence Not Provided;
Bradley M. Abrams, Residence Not Provided;
Anthony J. Moore, Residence Not Provided;
Christopher L. Anderson, Residence Not Provided;
Michael J. Pizzo, Residence Not Provided;
Robert A. Brigham II, Residence Not Provided;

Domestic Priority data as claimed by applicant

Foreign Applications

If Required, Foreign Filing License Granted: 01/21/2004

Projected Publication Date: To Be Determined - pending completion of Missing Parts

Non-Publication Request: No

Early Publication Request: No

Microsoft Patent Dept.
JAN 3 0 2004
D PD ND By: _____

Title

Design of application programming interfaces (APIs)

Preliminary Class
719

LICENSE FOR FOREIGN FILING UNDER
Title 35, United States Code, Section 184
Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Office of Export Administration, Department of Commerce (15 CFR 370.10 (j)); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

215

LOS ESTADOS UNIDOS DE AMERICA

A TODO AQUEL A QUIEN CONCIERNA LO PRESENTE

DEPARTAMENTO DE COMERCIO DE LOS ESTADOS UNIDOS

Oficina de Patentes y Marcas de los Estados Unidos

Julio 20, 2004

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

692320

NÚMERO DE SOLICITUD: *10/602,320*
FECHA DE PRESENTACIÓN: *Octubre 23, 2003*

Por la Autoridad del
COMISARIO DE PATENTES Y MARCAS

(Aparece Firma Ilegible)
P.R. GRANT
Oficial Certificador

216

Por favor escriba a máquina un signo más (+) en esta casilla → ⊞

PTO/SB/05(03-01)
Aprobado para uso durante 10/31/2002. OMB 0651-0032
Oficina de Marcas y Patentes de los Estados Unidos;
DEPARTAMENTO DE COMERCIO DE LOS ESTADOS UNIDOS
De acuerdo con el acta de reducción de papel de 1995, no se requiere que ninguna persona responda para una recolección de información a menos que éste muestre un número de control OMB válido.

TRASPASO DE SOLICITUD DE PATENTE DE UTILIDAD

(Solo para nuevas solicitudes no provisionales de acuerdo con el 37 C.F.R. 1,53(b))

Registro de abogado	No. MS1-1748US
Primer Inventor	Cwalina
Titulo	Diseños de Interfaces de Programación de Aplicación (API)
Etiqueta de correo expreso No.	EV355227055

217

ELEMENTOS DE LA SOLICITUD

Ver capítulo 600 MPEP que se relacionan con los contenidos de la solicitud de patente de utilidad.

DIRIGIDO A:

Solicitud de Patente Parada de Correo
Comisionado para Patentes/P.O. Box 1450
Alexandria, VA 22313 - 1450

1. ☑ Tasa del Formulario de Traspaso (por ejemplo, PTO/SB/17)
   (Presentar un original y un duplicado para la tasa de procesamiento)

2. ☐ El solicitante reivindica el estado de pequeña entidad.
   Ver 37 CFR 1.27.

3. ☑ Descripción *[total de hojas* [62]*]*

(La disposición preferida se establece adelante)

- Título descriptivo de la invención

28

- Referencia Cruzada a Solicitudes Relacionadas
- Declaración con respecto a patrocinio suministrado por R&D
- Referencia al listado de secuencias, una tabla, o un apéndice de lista de programa de computador
- Antecedentes de la Invención
- Breve Resumen de la Invención
- Breve Descripción de los Dibujos (si se presentan)
- Descripción Detallada
- Reivindicación(es)
- Resumen de la Descripción

4. ☑ Dibujo (s) (35 U.S.C. 113) [Total Hojas [10]]

5. Juramento o Declaración [total de páginas [ ]]

a. ☐ Formalizado nuevamente (original o copia)

b. ☐ copia de solicitud anterior (37 C.F.R. 1.63(d)) (para continuación/divisional con la casilla 18 Completada)

i. ☐ SUPRESIÓN DE INVENTOR(ES)

29

Declaración firmada adjunta suprimiendo inventor(es) mencionados en la solicitud anterior, ver 37 C.F.R. 1.63(d)(2) y 1.33(b).

6. ☐ Hoja de Datos de la Solicitud. Ver 37 CFR 1.76

7. ☐ CD-ROM o CD-R en duplicado, tabla grande o programa de Computador (Apéndice)

8. ☐ Presentación de secuencias de ácido amino y/o nucleótido *(si aplica, todo lo necesario)*

   a. ☐ Copia legible en Computador (CRF)

   b. ☐ listado de la Especificación de Secuencia en:

      i. ☐ CD-ROM o CD-R (2 copias); o

      ii. ☐ papel

   c. ☐ Declaración que verifica la identidad de las anteriores copias

230

PARTES DE LA SOLICITUD QUE LAS ACOMPAÑAN

9. ☐ Documentos de Traspaso (portadas & documento(s))

10. ☐ 37 C.F.R. 3.73(b) Declaración (cuando exista cesionario) ☐ Poder de Abogado

11. ☐ Documento de traducción al Inglés (si aplica)

12. ☐ Declaración de descripción de la información (IDS/PTO-1449) ☐ Copias de citaciones de IDS

13. ☐ Correcciones preliminares

14. ☑ Retorno postal de recibo (MPEP 503) *(debe estar itemizado específicamente)*

15. ☐ Copia Certificada de Documento(s) de Prioridad *(si se reivindica prioridad extranjera)*

221

16. ☐ Solicitud de no publicación de acuerdo con el U.S.C. 35 122 (b)(2)(B)(i). El Solicitante debe adjuntar formulario PTO/SB/35 o su equivalente.

17. ☐ Otro:.....

18. Si una SOLICITUD DE CONTINUACIÓN, marque la casilla apropiada, y suministre la información solicitada adelante y en una corrección preliminar, o en una Hoja de Datos de Solicitud de acuerdo con el 37 CFR 1.76:

☐ Continuación. ☐ Divisional. ☐ Continuación en parte (CIP) de una solicitud anterior No. _____

Información de la solicitud anterior: _____

Examinador: _____

Grupo/Unidad Técnica _____

Solo para SOLICITUDES de CONTINUACIÓN o DIVISIONAL: La descripción completa de la solicitud anterior, a partir de la cual un juramento o declaración se suministra de acuerdo con la Casilla 5b, se considera parte de la descripción de la continuación acompañante o solicitud divisional y por lo tanto se incorpora aquí como

222

referencia. La incorporación solo puede ser confiada cuando una porción de las partes de la solicitud presentada ha sido omitida inadvertidamente.

19. DIRECCIÓN DE CORRESPONDENCIA

☑ Número de Cliente o Etiqueta de Código de Barras o

☐ Dirección de Correspondencia adelante

22801

Nombre

Dirección

Ciudad

País

Teléfono Fax Código Zip

Nombre (Impreso/escrito a máquina) **Keith W. Saunders**

Firma (aparece firma)

Registro No. (Abogado/Agente) **41,462**

Fecha *10/23/2003*

223

Declaración de Límite de Tiempo: se estima que este formulario toma 0,2 horas en completarse. El tiempo variará dependiendo de las necesidades en cada caso individual. Cualquier comentario sobre la cantidad de tiempo que usted necesita para completar este formulario se debe enviar al Oficial Jefe de Información, Oficina de Marcas y Patentes en los Estados Unidos, Washington, DC 20231. NO ENVIAR TASAS O FORMULARIOS COMPLETOS A ESTA DIRECCIÓN. ENVIAR A: Comisionado Asistente para Patentes, Casilla Solicitud de Patente, Washington, DC 20231.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Design of Application Programming Interfaces (APIs)

Inventor(s):
Krzysztof J. Cwalina
Bradley M. Abrams
Anthony J. Moore
Christopher L. Anderson
Michael J. Pizzo
Robert A. Brigham II

ATTORNEY'S DOCKET NO. MS1-1748US

Design of Application Programming Interfaces (APIs)

TECHNICAL FIELD

This disclosure relates in general to application programming interfaces (APIs) and in particular, by way of example but not limitation, to designing APIs that are easy to use while simultaneously providing control and flexibility.

BACKGROUND

Application programming interfaces (APIs) are used by developers to create a wide variety of applications and programs. Developers range from office workers recording macros to low-level device driver authors. These developers rely on different languages and/or different frameworks of differing complexities while programming with different skill sets and/or for different purposes. Traditionally, different APIs have been designed to target different individual levels of skill and different demands for control (e.g., based on different relevant scenarios).

Although this approach can be successful in providing APIs that are optimized for a specific developer, it has significant drawbacks. For example, the multiple framework approach creates situations where developers have difficulty transferring knowledge from one skill level and scenario type to another. When there is a need for them to implement a scenario using a different framework, developers hit a very steep learning curve. And not only is the learning curve very steep, but it generally requires that the code written to a first lower-skill-level framework has to be rewritten from scratch to a second higher-skill-level

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

framework. Moreover, the creation of separate frameworks for different developer skill levels typically results in a situation in which APIs that are targeted for or implemented by one level of developer are unusable by another level of developer.

FIG. 1 illustrates a graph 101 of a traditional API learning curve with regard to two different frameworks. The first framework corresponds to a framework that has a relatively lower level of required skills and/or difficulty and a concomitantly relatively lower capacity for control by a developer. The second framework, on the other hand, corresponds to a framework that has a relatively higher level of required skills and/or difficulty and a concomitantly relatively higher capacity for control by a developer. Such a first framework might be used by a novice or infrequent developer, and such a second framework might be used by an experienced or professional developer. For example, the first framework may correspond to one designed for Visual Basic, and the second framework may correspond to one designed for C++.

In this traditional approach, relatively separate and disparate APIs are designed and employed as part of each framework. A steep but relatively short learning curve is traversed to enable API usage for the first framework at the relatively lower skill level and control capability. Because of the separate and disparate nature of the two API frameworks, the experience with the first framework contributes little if any knowledge toward learning the second API of the second framework. Consequently, an equally steep but even taller learning curve is traversed to enable API usage for the second framework.

In other words, learning an API of the first framework does not provide a stepping stone to learning an API of the second framework. The regressive nature of this disjointed set of API frameworks is indicated by the continuity gap. A

Eliminado: MSI-1748US PATAPP (2).DOC
Insertado: MSI-1748US PATAPP (2).DOC
Eliminado: MSI-1748US.PATAPP

developer who has learned the API of the first framework is no closer to learning the API of the second framework and must therefore start with the basics of the second framework.

Another problem with traditional frameworks is that they tend to have an overall poor usability in any case. In general, object oriented design/development (OOD) methodologies (e.g. unified modeling language (UML)) are "optimized" for maintainability of the resulting design and not for usability of the resulting frameworks. OOD methodologies are better suited for internal architecture designs and less suited for designs of an API layer of a large reusable library. For example, poor usability can result from OOD methodologies that focus only on distillation to a lowest fundamental block and/or that have an unwavering allegiance to a strict inheritance hierarchy throughout an API design.

Accordingly, there is a need for schemes and/or techniques that can at least ameliorate the regressive continuity gap of a traditional API learning curve and/or that can deliver better overall API usability.

SUMMARY

In a first exemplary method implementation, a method for designing an application programming interface (API) includes: preparing multiple code samples for a core scenario, each respective code sample of the multiple code samples corresponding to a respective programming language of multiple programming languages; and deriving the API from the core scenario responsive to the multiple code samples. In a second exemplary method implementation, a method for designing an API includes: selecting a core scenario for a feature area; writing at least one code sample for the core scenario; and deriving an API for the

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

core scenario responsive to the at least one code sample. In a third exemplary method implementation, a method for designing an API includes: deriving an API for a scenario responsive to at least one code sample written with regard to the scenario; performing one or more usability studies on the API utilizing multiple developers; and revising the API based on the one or more usability studies.

Other method, system, approach, apparatus, device, media, API, procedure, arrangement, etc. implementations are described herein.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like and/or corresponding aspects, features, and components.

FIG. 1 illustrates a graph of a traditional API learning curve with regard to two different frameworks.

FIG. 2 illustrates a graph of an exemplary progressive API learning curve with regard to two different levels of abstraction.

FIG. 3 illustrates exemplary design principles and practices for APIs.

FIG. 4 is a flow diagram that illustrates an exemplary technique for designing APIs per feature area.

FIG. 5 is a block diagram that illustrates an exemplary scheme for designing APIs per core scenario.

FIG. 6 illustrates potential disparity between exemplary component types that are targeted to two different purposes.

FIG. 7 illustrates an exemplary relationship between component types that are designed to be extensible and/or interoperable so as to cover two different purposes.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

FIG. 8 illustrates an exemplary aggregate component (AC) and associated factored types (FTs) for handling two different purposes with a two-layer API.

FIG. 9 illustrates an exemplary aggregate component and associated APIs that can support a create-set-call usage pattern.

FIG. 10 illustrates an exemplary computing (or general device) operating environment that is capable of (wholly or partially) implementing at least one aspect of designing and/or using APIs as described herein.

DETAILED DESCRIPTION

FIG. 2 illustrates a graph 200 of an exemplary progressive API learning curve with regard to two different levels of abstraction. The two different illustrated levels of abstraction are a relatively high level of abstraction and a relatively low level of abstraction. The high level of abstraction corresponds to a development environment that involves a relatively lower level of required skills and/or difficulty and a concomitantly relatively lower capacity for control by a developer. The low level of abstraction, on the other hand, corresponds to a development environment that involves a relatively higher level of required skills and/or difficulty and a concomitantly relatively higher capacity for control by a developer.

A progressive API learning curve is shown rising from a point of lower required skills and concomitant control capability in a relatively smooth manner through the areas for the high and low levels of abstraction to a point of higher required skills and concomitant control capability. The progressive API learning curve exhibits a continuity zone between the areas of the high level of abstraction and the low level of abstraction. An integrated API framework enables a gradual

Eliminado: MSI-1748US PATAPP (2).DOC
Insertado: MSI-1748US PATAPP (2).DOC
Eliminado: MSI-1748US.PATAPP

learning curve. Because of the integrated nature of the API framework, experience with the high level of abstraction contributes to knowledge toward learning API usage for the low level of abstraction as well as for scenarios demanding greater control.

In other words, learning the API for higher levels of abstraction provides a stepping stone to learning and/or extending the API into lower levels of abstraction. This is indicated by the two-layer (API) framework shape encompassing both the high and the low level of abstraction areas. The progressive nature of certain APIs, as described herein below, enable developers to use simple APIs initially and to gradually (and partially) begin using more complicated API components. Thus, developers who have learned the APIs targeting the higher levels of abstraction can move to using the APIs targeting the lower levels of abstraction as their experience warrants and/or as the complexity of the scenarios that they are facing demand.

A progressive API can be easily usable (especially during early learning phases) and highly powerful (especially as the API is explored over time). A usable API may include one or more of the following exemplary attributes: a small number of concepts and/or classes are required to get started, a few lines of code can implement simple scenarios, classes/methods have intuitive names, a natural and/or obvious starting point is apparent, and there is a clear (e.g., discoverable) progression to additional required and/or relevant concepts/classes.

A progressive API can also enable an incremental advancement from developing at a point of lower difficulty and concomitant control capability to a point of higher difficulty and concomitant control capability. Exemplary

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

paradigms for designing progressive APIs, as well as generally highly usable APIs, are described herein below.

FIG. 3 illustrates exemplary design principles and practices for APIs in a table 300. Table 300 indicates general design principles and practices for four exemplary categories 302-308. Specifically, the following four categories are addressed: scenario driven design 302, component oriented design 304, customizable defaults 306, and self documenting object model 308.

When designing a given API, the design principles and practices for any one or more of the indicated categories 302-308 may be employed. Furthermore, within any given category 302-308, one or more of the illustrated design principles and practices may be implemented. In other words, neither every category nor every design principle and practice thereof need be employed or implemented for a given API design.

Scenario driven design category 302 illustrates four exemplary design principles and practices. Firstly, core scenarios for selected features or technological areas are defined. Secondly, code samples corresponding to the core scenarios are written first, and the API is designed responsive thereto second. Thirdly, a progressive API, as introduced above and described further herein below, is designed. Fourthly, utilizing the defined core scenarios is made easy while utilizing other scenarios is made possible. Scenario driven design 302 is described further below in the section entitled "Scenario Driven Design".

Component oriented design category 304 illustrates three exemplary design principles and practices. Firstly, aggregate components (ACs) are created. Generally, aggregate components are directed toward core scenarios, are relatively simple and easy to use, and are built on top of factored types (FTs). Factored

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

types are more fundamental and are decomposed to lower logical levels. This results in a two-layer API design. Secondly, these aggregate components are interrelated with the factored types. Thirdly, a create-set-call usage pattern is supported, especially for aggregate components. Component oriented design 304 is described further below in the section entitled "Component Oriented Design".

Customizable defaults category 306 illustrates two exemplary design principles and practices. Firstly, required initializations to use at least aggregate components are reduced. Defaults are used to reduce required initializations. Secondly, defaults are selected that are appropriate for the defined core scenarios. Customizable defaults 306 is described further below in the section entitled "Customizable Defaults".

Self documenting object model category 308 illustrates four exemplary design principles and practices. Firstly, simple and intuitive names are reserved for core scenarios. Secondly, names are selected based on the intended use or purpose of the component type, instead of a hidebound adherence to the inheritance hierarchy. Thirdly, actionable exceptions are thrown so that a developer receives instructions indicating how to fix an error from the exception message. Fourthly, clean namespaces are produced by placing types that are rarely used into sub-namespaces to avoid cluttering the main namespaces. Self documenting object model 308 is described further below in the section entitled "Self Documenting Object Model".

Scenario Driven Design

In a described implementation, API specifications are driven by scenarios. Accordingly, API designers first write the code that the users of the API will have to write in core (e.g., main) scenarios. API designers then design an object model

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

to support these code samples. This approach contrasts with starting a design of an object model (using various design methodologies) and then writing code samples based on the resulting API.

In other words, especially for public API design, API designers start with a list of scenarios for each feature or technology area and code samples therefor and produce a header-style object model description based thereon. Examples of feature areas include: file I/O, networking, messaging, console, diagnostics, database access, web pages, graphical user interface (GUI) programming, and so forth.

FIG. 4 is a flow diagram 400 that illustrates an exemplary technique for designing APIs per feature area. At block 402, core scenarios are selected for the given feature area. For example, for a given technology area, the top 5-10 scenarios may be selected. They may be selected based on the most commonly used functions (e.g., most common tasks) or the most frequently pursued goals for the given technology area. For instance, exemplary scenarios for a file I/O technology feature area are reading from a file and writing to a file.

At block 404, code samples for a core scenario are written in multiple (e.g., two or more) languages. For example, code samples associated with a selected core scenario may be written in three different languages. The code samples may implement the current selected core scenario in the three languages. Such languages include, for example, VB, C#, MC++, a markup language, and so forth; however, other languages may also be used. As indicated by the asterisk, it should be understood that a code sample (or even more than one code sample) may be written for the core scenario in a single language when designing a usable and powerful API for a single language.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Writing code samples in multiple languages may be performed because sometimes code written in different languages differs significantly. In a described implementation, the code samples for the current selected core scenario are written using different coding styles that are common among users of the particular language (e.g., using language-specific features or traits, using the practices/habits of developers, etc.) in which a particular code sample is written. For example, the samples may be written using language-specific casing. For instance, VB is case-insensitive, so code samples written in VB reflect that variability. Code samples written in C#, on the other hand, follow the standard casing therefor.

Another example relates to a statement called "using", which C# supports. For instance, the "using" call encapsulates a try/finally block. However, VB does not support this feature, and writing code samples can indicate that utilizing this feature in a try/finally statement is awkward for VB users. Yet another example relates to assignments in a conditional clause, which C# supports. In a file I/O instance: "if ((text = reader.ReadLine() != null)" works in C#. However, the assignment statement cannot be used within the "if" clause in VB; instead, the code is broken into multiple statements. Still yet another example relates to the tendency of C# developers to utilize parameterized constructors while VB developers usually do not. For instance, a C# coding may be "MyClass x = new MyClass("value")" while a corresponding VB coding is "Dim x As MyClass" and "x.Property = "value"."

At block 406, an API is derived from the current core scenario responsive to the code samples written in the multiple languages. For example, factors gleaned from the code samples written in each of the multiple languages may be incorporated into the API. Such factors may include similarities across the

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

different code samples, differences between/among two or more code samples, and so forth. Such factors, as well as other aspects of blocks 404 and 406, are described further below with reference to FIG. 5.

Similarly, when designing an API for a single language, the API is derived from the current core scenario responsive to the code sample(s) written in the single language. Thus, factors gleaned from the code sample(s) written in the single language may be incorporated into the API. As an additional API design factor example for single or multiple language situations, an API design factor may include compatibility with tools that are oriented toward the language or languages for which the code sample(s) are written.

At block 408, it is determined if the API is too complex. For example, the API may be reviewed by the API designer(s) to determine if the API is or is not too complex. In other words, an initial check may be performed to consider whether the API can be used without significant understanding of multiple other specific APIs, without undue experimentation, and so forth. Such an initial check may also verify that the derived API is actually workable in the current core scenario in every relevant language. If the API is too complex, then the API is refined by the designers with reference to the current core scenario and responsive to the code samples written in the multiple languages at block 406.

If, on the other hand, it is determined that the API is not too complex (at block 408), then at block 410 usability studies with typical developers are performed. For example, one or more usability studies may be performed using a development environment akin to that which the typical developer normally uses. Such a normal development environment likely includes intellisense, editors,

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

language, and a documentation set that is most widely used by the targeted developer group.

Usability Studies

Usability studies that target a wide range of developers facilitate scenario-driven design, especially when designing general public APIs. The code samples written by the API designer(s) for the core scenarios probably appear simple to them, but the code samples might not be equally simple to certain groups of developers that are in fact targeted (e.g., especially novice and/or occasional developers). Additionally, the understanding, which is garnered through usability studies, regarding the manner in which developers approach each core scenario can provide powerful insight into the design of the API and how well it meets the needs of all of the targeted developers.

Generally, usability studies may be conducted early in the product cycle and again after any major redesign of the object model. Although this is a costly design practice, it can actually save resources in the long run. The cost of fixing an unusable or merely defective API without introducing breaking changes is enormous.

At block 412, it is ascertained whether typical developers are able to use the API without significant problem(s). For example, most subjects should be able to write code for the current selected scenario without major problems. If they cannot, the API is revised (as described below with reference to block 414).

The interpretation of significant/major problems hinges on a desired level of usability for a given targeted developer group. For example, frequent and/or extensive reference to detailed API documentation for the current core scenario by test subjects may constitute significant problems. Generally, if the majority of test

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

developers cannot implement the current core scenario, or if the approach they take is significantly different from what was expected, the API should be evaluated for possible revisions (up to and including a full redesign).

If it is ascertained that typical developers are unable to use the API without major problems (at block 412), then at block 414 the API is revised based on lessons from the usability studies. For example, a default may be changed, another property may be added, one or more attributes may be exposed instead of encapsulated, and so forth.

If, on the other hand, it is ascertained that typical developers are able to use the API without major problems (at block 412), then at block 416 the process is repeated for each core scenario. For example, another core scenario of the selected core scenarios for the given feature becomes the current core scenario for which code samples are written (at block 404). Another exemplary technique for designing APIs, which focuses more on two-layer API design, is described further below in conjunction with FIG. 8.

FIG. 5 is a block diagram 404/406 that illustrates an exemplary scheme for designing APIs per core scenario. The illustrated exemplary scheme corresponds to blocks 404 and 406 of FIG. 4 for a multiple language implementation. A code sample 502(1) for a first language, a code sample 502(2) for a second language, and a code sample 502(3) for a third language is shown. Each of the three code samples 502(1, 2, 3) are directed to a given current core scenario. Although three code samples 502(1, 2, 3) corresponding to three languages are shown, two or more code samples 502 of any arbitrary number of targeted languages may alternatively be used in this exemplary multiple-language implementation.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

In a described implementation, factors 506 are gleaned from code samples 502(1, 2, 3) that are written in each of the three languages. These factors 506 are incorporated into the API 504. Specifically, API 504 is designed to support the three code samples 502(1, 2, 3) that are written in the three respective corresponding languages. However, it should be understood that factors 506 may also be applicable to single-language implementations.

Some exemplary factors 506 are described above with reference to blocks 404 and 406 of FIG. 4, and other exemplary factors 506 are indicated in block diagram 404/406 of FIG. 5. Such factors 506 include language-specific mandates that are revealed by a review of code samples 502(1, 2, 3). An example of a language-specific constraint is described with regard to the following sample line of code: "Foo f = new Foo();". A progressive API that is designed to support this sample line has to include a default constructor; otherwise, the code sample does not compile correctly.

Factors 506 also include developer expectations that are inspired by both language peculiarities and the different skill/experience levels of typical developers that naturally gravitate toward the different languages. Factors 506 further include commonalities of code and coding practices across the different languages as discoverable by a review of code samples 502(1, 2, 3).

While considering factors 506 that directly relate to different languages, other factors 506, as described herein, continue to be considered. For example, the following factors 506 are also pertinent to progressive APIs targeted to single-language environments as well as multiple-language environments. First, the number of different component types that are required to complete a scenario is a factor. Generally, the more component types that are required, the harder it is to

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

learn. A second factor is the connection between succeeding lines of code. To the extent that usage of one component type leads a developer towards usage of the next required component type, the easier the API is to use.

Third, consistency in the naming of identifiers is another factor. A fourth factor involves the appropriate usage of properties, methods, and events. A fifth factor relates to possible similarities to one or more existing APIs. Sixth, another factor involves compliance with overall design guidelines for an API. A seventh factor relates to whether the APIs overlap with other component types of the framework. Eighth, compatibility with tools that are oriented toward a particular language is yet another factor. For instance, VB developers typically want parameter-less constructors and property setters. Other factors 506 may alternatively be considered.

Still other factors 506 that relate to interrelationships of aggregate components and factored types are described below, especially with reference to FIG. 8. Although the method and scheme of FIGS. 4 and 5 may be applied to designing APIs in general, they are particularly applicable to designing two-layer APIs. A two-layer API paradigm (e.g., with aggregate components and factored types) is described below with reference to FIGS. 6-8 in the section entitled "Component Oriented Design".

Component Oriented Design

FIG. 6 illustrates potential disparity between exemplary component types 602 that are targeted to two different purposes along a continuum 600. Continuum 600 extends from a high usability range on the left side to a high controllability range on the right side. Multiple component types 602 are spread across continuum 600.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Generally, component types 602 that are illustrated as being relatively larger represent types that are simpler and therefore easier to use. Conversely, component types 602 that are illustrated as being relatively smaller represent types that are more complex and therefore more difficult to use. Simple and complex in this context refer to how easy or how difficult the particular component types 602 are to use when implementing a specified scenario.

The component types 602 that are illustrated as being relatively smaller are generally more difficult to use for a number of exemplary reasons as follows: First, developers have more choices as to which component types 602 they should use. In the illustrated example, there are 14 "choices" for the smaller component types 602 as compared to three "choices" for the larger component types 602. More specifically, a developer has to know or discern, from among the various component types 602(HC), which component type or types to use. This involves understanding each of the (e.g., 14) multiple component types 602(HC) as well as how they interrelate, which contrasts with starting with a single component type 602(HU) from among the fewer (e.g., 3) component types 602(HU). Differences between component types 602(HU) and component types 602(HC) are described further below.

By way of an exemplary analogy, the smaller component types 602 are like the individual components of a stereo system; hence, a user has to know which components are needed and how to hook them together. Without hooking them together, they are generally not useful. The larger component types 602 are like all-in-one stereos that are easily usable but likely to be less powerful as well as less flexible. A second reason that smaller component types 602 are harder to use is that there are potentially more "starting points". Third, there are generally more

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

concepts to understand. Fourth, a developer has to understand how individual components types 602 relate to other component types 602.

In a described implementation, component types 602 are divided into those with a high usability purpose 602(HU) and those with a high controllability purpose 602(HC). High usability component types 602(HU) are simpler and easier to use, but they tend to be more inflexible, limiting, and/or constraining. They can generally be used without extensive knowledge of an overall API. High usability component types 602(HU) are usually capable of implementing a limited number of scenarios or at most a limited number of approaches to each scenario of interest.

High controllability component types 602(HC), on the other hand, are complex to use, but they provide a greater degree of control to developers. They are relatively powerful and enable developers to effectuate low-level tweaking and tuning. However, developing with high controllability component types 602(HC) entails a fuller understanding of many component types 602 to enable the instantiation of multiple high controllability component types 602(HC) that are correctly interlinked to implement even relatively straight-forward scenarios.

Typically, high usability component types 602(HU) are present in introductory languages such as VB, and high controllability component types 602(HC) are present in advanced professional-programmer-type languages such as C++. The potential disparity between high usability component types 602(HU) and high controllability component types 602(HC) that is illustrated in FIG. 6 is at least partly ameliorated by component types 702 of FIG. 7. Specifically, an interrelationship between high usability component types 602(HU) and high controllability component types 602(HC) is established by a progressive API.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

FIG. 7 illustrates an exemplary relationship between component types 702 that are designed to be extensible and/or interoperable so as to cover at least two different purposes along a continuum 700. In a described implementation, component types with a high usability purpose are realized as aggregate components 702(AC), and component types with a high controllability purpose are realized as factored types 702(FT). Although component types 702 are divided into only two purposes, they may alternatively be separated into three or more purposes (or other categories).

A key 704 indicates that a solid line represents a relationship for exposed factored types and that a dashed line represents a relationship for encapsulated factored types. As illustrated, aggregate component 702(AC)(1) has a relationship with three factored types 702(FT). Specifically, factored type 702(FT)(1) has an exposed factored type relationship with aggregate component 702(AC)(1), and factored types 702(FT)(2) and 702(FT)(3) have an encapsulated factored type relationship with aggregate component 702(AC)(1).

Although not so illustrated, two or more aggregate components 702(AC) may have an encapsulated and/or exposed relationship with the same factored type 702(FT). Exposed and encapsulated factored types 702(FT) and aggregate components 702(AC), as well as relationships therebetween, are described further below, including with reference to FIG. 8.

Component oriented design relates to offering a single object per user concept as opposed to requiring multiple objects per logical concept. Aggregate components therefore usually correspond to a user concept and are simpler from a usability perspective. Aggregate components are layered on top of factored types. By way of an exemplary comparison, aggregate components may model a thing

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

such as a file, and factored types may model a state of a thing such as a view on the file. Together, aggregate components and factored types provide a progressive and gradual learning curve for new developers, especially with respect to a particular given API.

Component Oriented Design for Aggregate Components

Many feature areas may benefit from façade types that act as simplified views over a more complex but well-factored remainder of the feature area APIs. In a described implementation, the façade covers the top 5-10 scenarios in a given feature area and optionally other high-level operations. Aggregate components 702(AC) can serve as such façade types, and factored types 702(FT) can provide a remaining well-factored complex API landscape.

Each aggregate component ties multiple lower level factored classes into a higher-level component to support the top core scenarios. For example, a mail aggregate component may tie together SMTP protocol, sockets, encodings, and so forth. Generally, each aggregate component provides a higher abstraction level rather than just a different way of doing things. Providing simplified high-level operations is helpful for those developers who do not want to learn the whole extent of the functionality provided by a feature area and merely wish to accomplish their often very simple tasks without significant study or API exploration.

Generally, component oriented design is a design based on constructors, properties, methods, and events. Using aggregate components is a relatively extreme application of component oriented design. An exemplary set of parameters for component oriented design of aggregate components is provided below:

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Constructors: aggregate components have default (parameter-less) constructors.

Constructors: optional constructor parameters correspond to properties.

Properties: most properties have getters and setters.

Properties: properties have sensible defaults.

Methods: methods do not take parameters if the parameters specify options that stay constant across method calls (in the selected core scenarios). Such options may be specified using properties.

Events: methods do not take delegates as parameters. Callbacks are implemented in terms of events.

Component oriented design entails considering how the API is used instead of focusing on the mere inclusions of the methods, properties, and events in the object model. An exemplary usage model for component oriented design involves a pattern of instantiating a type with a default or relatively simple constructor, setting some properties on the instance, and then calling simple methods. This pattern is termed a Create-Set-Call usage pattern. A general example follows:

```
' VB
' Instantiate
Dim T As New T()

' Set properties/options.
T.P1 = V1
T.P2 = V2
T.P3 = V3
```

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

```
' Call methods; optionally change options between calls.
T.M1()
T.P3 = V4
T.M2()
```

When aggregate components support this Create-Set-Call usage pattern, the aggregate components comport with the expectations of the main users of aggregate components. Moreover, tools, such as intellisense and designers, are optimized for this usage pattern. A concrete code example showing the Create-Set-Call usage pattern follows:

```
' VB
' Instantiate
Dim File As New FileObject()

' Set properties.
File.Filename = "c:\foo.txt"
File.Encoding = Encoding.Ascii

' Call methods.
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
File.Close()
```

With an exemplary aggregate component that is part of a progressive API, setting the "File.Encoding" property is optional. The API has a default for a pre-selected

Eliminado: MSI-1748US PATAPP (2).DOC
Insertado: MSI-1748US PATAPP (2).DOC
Eliminado: MSI-1748US.PATAPP

file encoding if one is not specified. Similarly, with regard to "File.Open()", specifying an "OpenMode.Write" is optional. If it is not specified, a default "OpenMode" as pre-selected by the API is employed.

An issue with component oriented design is that it results in types that can have modes and invalid states. For example, a default constructor allows users to instantiate a "FileObject" without specifying a "FileName". Attempting to call Open() without first setting the "FileName" results in an exception because the "FileObject" is in an invalid state with respect to being opened (e.g., no file name has yet been specified). Another issue is that properties, which can be set optionally and independently, do not enforce consistent and atomic changes to the state of the object. Furthermore, such "modal" properties inhibit sharing of an object instance between consumers because a first user has to check a previously-set value before reusing it in case a second user has changed the value in the interim. However, the usability of aggregate components outweighs these issues for a vast multitude of developers.

When users call methods that are not valid in the current state of the object, an "InvalidOperationException" is thrown. The exception's message can clearly explain what properties need to be changed to get the object into a valid state. These clear exception messages partially overcome the invalid state issue and result in an object model that is more self-documenting.

API designers often try to design types such that objects cannot exist in an invalid state. This is accomplished, for example, by having all required settings as parameters to the constructor, by having get-only properties for settings that cannot be changed after instantiation, and by breaking functionality into separate types so that properties and methods do not overlap. In a described

Eliminado: MS1-1749US PATAPP (7).DOC

Insertado: MS1-1748US PATAPP (7).DOC

Eliminado: MS1-1748US.PATAPP

implementation, this approach is employed with factored types but not with aggregate components. For aggregate components, developers are offered clear exceptions that communicate invalid states to them. These clear exceptions can be thrown when an operation is being performed, instead of when the component is initialized (e.g., when a constructor is called or when a property is set), so as to avoid situations where the invalid state is temporary and gets "fixed" in a subsequent line of code.

Factored Types

As described above, aggregate components provide shortcuts for most common high level operations and are usually implemented as a façade over a set of more complex but also richer types, which are called factored types. In a described implementation, factored types do not have modes and do have very clear lifetimes.

An aggregate component may provide access to its internal factored types through some properties and/or methods. Users access the internal factored types in relatively advanced scenarios or in scenarios where integration with different parts of the system is required. The ability to access factored type(s) that are being used by an aggregate component enables code that has been written using the aggregate component to incrementally add complexity for advanced scenarios, or integrate with other component types, without having to re-write code from the beginning with a focus on using the factored types.

The following example shows an exemplary aggregate component ("FileObject") exposing its exemplary internal factored type ("StreamWriter"):

`VB

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

```
Dim File As New FileObject("c:\foo.txt")
File.Open(OpenMode.Write)
File.WriteLine("Hello World")
AppendMessageToTheWorld(File.StreamWriter)
File.Close()
...

Public Sub AppendMessageToTheWorld(ByVal Writer As
StreamWriter)
  ...
End Sub
```

High Level Operations

In a described implementation, aggregate components, as the upper or higher level APIs (e.g., from a level of abstraction perspective), are implemented such that they appear to "magically" work without the user being aware of the sometimes complicated things happening underneath. For example, an "EventLog" aggregate component hides the fact that a log has both a read handle and a write handle, both of which are opened in order to use it. As far as a developer may be concerned, the aggregate component can be instantiated, properties can be set, and log events can be written without concern for the under-the-hood functioning.

In some situations, a bit more transparency may facilitate some task with the developer. An example is an operation in which the user takes an explicit action as a result of the operation. For instance, implicitly opening a file and then requiring the user to explicitly close it is probably taking the principle of

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

"magically" working too far. Nevertheless, a diligent API designer may often be capable of designing clever solutions that hide even those complexities. For example, reading a file can be implemented as a single operation that opens a file, reads its contents, and closes it; the user is thus shielded from the complexities related to opening and closing the file handles.

Furthermore, using aggregate components does not involve implementing any interfaces, modifying any configuration files, and so forth. Instead, library designers can ship default implementations for interfaces that are declared. Moreover, configuration settings are optional and backed by sensible defaults.

FIG. 8 illustrates an exemplary aggregate component 702(AC) and associated factored types 702(FT) for handling two different purposes with a two-layer API 800. Aggregate component 702(AC) represents a first or higher layer, and factored types 702(FT) represent a second or lower layer. The first layer effectively builds on the second layer with a custom interface.

As illustrated, aggregate component 702(AC) includes multiple aggregate component (AC) members 802. Specifically, aggregate component members 802(1), 802(2), 802(P)(1), 802(P)(2), 802(M)(1), 802(M)(2), and 802(M)(3) are shown. Aggregate component 702(AC) also includes exposed factored types 702(FT-Ex) and encapsulated factored types 702(FT-En). Specifically, exposed factored types 702(FT-Ex)(1) and 702(FT-Ex)(2) and encapsulated factored types 702(FT-En)(1) and 702(FT-En)(2) are shown. Factored types 702(FT) also include factored type (FT) members 804.

In a described implementation, aggregate component 702(AC) includes at least one aggregate component member 802, which may be a method or a property for example. Aggregate component members 802 can therefore include aggregate

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

component methods 802(M) and aggregate component properties 802(P). These aggregate component members 802, such as aggregate component members 802(1) and 802(2), may be specific to the aggregate component 702(AC). In other words, some aggregate component members 802 like aggregate component members 802(1) and 802(2) that are on aggregate component 702(AC) may not rely on any factored types 702(FT). Alternatively, some aggregate component members 802 may be linked to underlying factored types 702(FT).

Factored types 702(FT) may be exposed factored types 702(FT-Ex) or encapsulated factored types 702(FT-En). Exposed factored types 702(FT-Ex) are factored types 702(FT) of a given aggregate component 702(AC) that may be accessible by or to other general component types 702(FT or AC) without using or going through individual aggregate component members 802 of the given aggregate component 702(AC). If a factored type 702(FT-Ex/En) is returned by an aggregate component member 802 (either a method or a property), then that factored type 702(FT-Ex) is exposed. Otherwise, that factored type 702(FT-En) is encapsulated.

In other words, an aggregate component member 802 can expose a factored type member 804, or an aggregate component member 802 can return a factored type instance. The latter can occur with exposed factored types 702(FT-Ex), and the former can occur with encapsulated factored types 702(FT-En). Encapsulated factored types 702(FT-En) are factored types 702(FT) of a given aggregate component 702(AC) that are contained within or internal to the given aggregate component 702(AC). Each factored type 702(FT) may include one or more members 804 (some of which are specifically indicated in FIG. 8) that are methods and/or properties.

Eliminado: MS1-1746US PATAPP (2).DOC
Insertado: MS1-1746US PATAPP (2).DOC
Eliminado: MS1-1746US.PATAPP

As illustrated, two method members 804 of encapsulated factored type 702(FT-En)(1) are exposed by aggregate component 702(AC) as method member 802(M)(1) and method member 802(M)(2). One method member 804 of encapsulated factored type 702(FT-En)(2) is exposed by aggregate component 702(AC) as method member 802(M)(3).

Exposed factored type 702(FT-Ex)(1) is itself exposed as a property member 802(P)(1) of aggregate component 702(AC). Similarly, exposed factored type 702(FT-Ex)(2) is also exposed as a property member 802(P)(2) of aggregate component 702(AC). As indicated, a factored type (FT) member 804 of exposed factored type 702(FT-Ex)(1) is exposed so as to be separately accessible (i.e., accessible without directly using an individual member 802 of aggregate component 702(AC)). Hence, a factored type member 804 of an exposed factored type 702(FT-Ex) can still be accessed even if it is not individually exposed by an aggregate component member 802 of aggregate component 702(AC).

Thus, the indicated member 804 of exposed factored type 702(FT-Ex)(1) is exposed so as to be accessible by component types 702 that are external to aggregate component 702(AC) without using a member 802 thereof. As indicated by the dashed lines emanating from exposed factored type 702(FT-Ex)(1), exposed factored types 702(FT-Ex) may be "handed off" for use by other component types 702 (especially by other factored types 702(FT)) that are unable to interact with aggregate components 702(AC) or that can better achieve their intended purpose using the handed-off exposed factored type 702(FT-Ex)(1) alone. It should be noted that the object (exposed factored type 702(FT-Ex)(1)) that is "handed off" is not a copy but rather an actual part of the exposing aggregate component 702(AC).

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

As a result, the operations on the handed off object affect aggregate component 702(AC).

Generally, if a factored type 702(FT) is encapsulated, it is not exposed to a consumer; instead, setting properties 802(P) or calling methods 802(M) on an aggregate component 702(AC) may cause factored types 702(FT) to be created, properties 804 to be set, or methods 804 to be called on the underlying factored type 702(FT). These members 802 and 804 may not have a one-to-one correspondence; for example, setting several properties 802(P) on an aggregate component 702(AC) may be cached in the aggregate component 702(AC). Subsequently calling a method 802(M) on the aggregate component 702(AC) may cause a factored type 702(FT) to be created using the previously-specified values of the several properties 802(P) as constructor arguments for the factored type 702(FT).

In a described implementation, aggregate components 702(AC) differ from more-traditional object-oriented components in at least two ways in addition to the exposure of exposed factored types 702(FT-Ex). First, an aggregate component 702(AC) does not necessarily expose every member 804 of all of its factored types 702(FT). In other words, aggregate components 702(AC) are not strictly devoted to an inheritance hierarchy. Second, aggregate components 702(AC) can have modes and thus may periodically have states that result in invalid operations.

As a guideline to component oriented design with respect to factors 506 (of FIG. 5), whether a factored type 702(FT) is exposed or encapsulated within an aggregate component 702(AC) may be based on one or more of a number of factors. First, a particular factored type 702(FT) is exposed as a property member 802(P) of a given aggregate component 702(AC) if the particular factored type

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

702(FT) includes functionality that is not exposed by the given aggregate component 702(AC). Second, a particular factored type 702(FT) is exposed as a property member 802(P) of a given aggregate component 702(AC) if other general component types 702 of the framework may benefit from a handoff for direct consumption of the particular factored type 702(FT). On the other hand, a particular factored type 702(FT) is not exposed (and is therefore encapsulated) when functionality of the particular factored type 702(FT) is completely exposed by a given aggregate component 702(AC) and when the particular factored type 702(FT) is not useful for handing off to other component types 702.

A developer can start with aggregate component 702(AC), especially for implementing simpler and/or core scenarios. When the developer wishes or needs to implement a more complex scenario, the developer can incrementally and gradually begin to directly access and use exposed factored types 702(FT-Ex), including low-level attributes thereof, over time. The original code that relied on the simpler aggregate component 702(AC) does not need to be jettisoned and replaced with more complicated coding that relies solely on factored types 702(FT). The two-layers of the API framework can be used in varying proportions and can co-exist simultaneously.

Designing a two-layer API framework can be accomplished using the following exemplary technique that is described in ten phases: First, a set of core scenarios for a particular feature area is selected. Second, sample codes showing the preferred lines of code for the selected core scenarios are written. Third, aggregate components are derived with the appropriate methods, defaults, abstractions, naming, etc. to support the code samples from the lines of code.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Fourth, the code samples from the second phase are refined as appropriate according to the derived aggregate components. Fifth, the refined code samples are evaluated for whether or not they are sufficiently simple. If not, the technique continues again at the third phase. If so, then at the sixth phase it is determined whether additional scenarios, usages, interactions with other components, and/or other requirements exist. Seventh, the API designer decides if any of the additional requirements discovered in the sixth phase can be added to the aggregate components without adding undue complexity to the selected core scenarios.

Eighth, if the additional requirements cannot be added to the aggregate components, an ideal factoring (e.g., based on object-oriented or other analytical methodologies) of a full set of functionality for the factored types is defined based on the seventh phase. Ninth, it is determined how and whether the aggregate components encapsulate or expose the functionality from the factored types that are defined in the eighth phase. Tenth, the factored types are refined as appropriate to support the aggregate components as well as the additional requirements. Using this exemplary technique, a two-layer API framework having aggregate components 702(AC) and factored types 702(FT) may be designed.

FIG. 9 illustrates an exemplary aggregate component 702(AC) and associated APIs 902, 802(P), 802(M), and 904 that can support a create-set-call usage pattern. The following exemplary API groups are illustrated: constructors 902, properties 802(P), methods 802(M), and events 904. With a create, set, call usage pattern, an instance of aggregate component 702(AC) is initially created by a developer with reliance on default (e.g., parameter-less) constructors 902.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Secondly, the developer sets any properties 802(P) for which the default values are inappropriate and/or non-preferred for the intended use of the object. Thirdly, desired methods 802(M) are called by the developer. Callbacks are then implemented in terms of events 904.

Customizable Defaults

Customizable defaults relates to having defaults whenever practicable for at least aggregate components. When designing an API for example with multiple code samples corresponding to multiple languages, an identical value that is passed in each of the code samples can instead be set as a default for the aggregate component. The customizable defaults may be changed by setting one or more properties on the aggregate component.

Many developers prefer to code by trial and error as opposed to taking the time to read the documentation and fully understand a feature area prior to beginning a project. This is particularly true for novice and occasional developers, such as those that code with VB. These developers often try to experiment with an API to discover what it does and how it works, and then they adjust their code slowly and incrementally until the API implementation achieves their goal. The popularity of the editing and continuing approach to development is a manifestation of this preference.

Some API designs lend themselves to "coding by experimentation" and some do not. There are multiple aspects that affect the level of success a developer is likely to have when using a coding by experimentation approach. These aspects include: (i) how easy it is to locate the right API for the task at hand; (ii) how easy it is to start using an API, regardless of whether it (initially) does what the developer wants it to do or not; (iii) how easy it is to discover what

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

the points of customization are for an API; (iv) how easy it is to discover the correct customization for a given scenario; and (v) so forth.

In a described implementation, APIs are designed to require little if any initialization (e.g., a minimal amount of initialization). For example, an API can be designed so that a default constructor or a constructor with one simple parameter is sufficient to start working with a type. When initialization is necessary, an exception that results from not performing the initialization clearly explain what needs to be done and/or changed in order to remove or prevent the exception. For example, an exception may stipulate what or which property needs to be set.

By way of example but not limitation, a rule of thumb is that the simplest constructor has three or fewer parameters (with an upper limit of five). In addition, the simplest constructors should avoid complex types as any of the parameters, where complex types may be other factored types or aggregate components. Another rule of thumb is that the simplest constructors rely on primitive types like enumerations, strings, integers, and so forth. Types may also implement more complex constructor overloads to support more complex scenarios.

In short, an API's customizability can be simplified by providing properties with good defaults for all customization points. (However, developers should generally be able to add new code to the existing code when customizing their scenarios; rewriting the entire code from scratch using a different API should be optional.) For example, a system messaging queue aggregate component enables the sending of messages after passing a path string to the constructor and calling a

Eliminado: MSI-1748US PATAPP (2).DOC

Insertado: MSI-1748US PATAPP (2).DOC

Eliminado: MSI-1748US.PATAPP

send method. Message properties, such as message priority and encryption algorithms, can be customized by adding code to the core scenario.

Self Documenting Object Model

Self documenting object model relates to designing an API framework in which a developer can look at objects and members thereof to learn about them as well as be able to use them. For example, names can be based on how a type is expected to be used instead of devotion to an inheritance hierarchy that many developers do not wish to study. In short, a self documenting object model facilitates discoverability by would-be developers.

As noted above, some developers prefer to code by trial and error and resort to reading documentation only when their intuition fails to implement their intended scenario. Thus, a self documenting object model should avoid requiring that developers read documentation every time they want to perform even simple tasks. An exemplary set of principles and practices to help in producing intuitive APIs that are relatively self documenting for a described implementation are presented below. Any one or more of them may be utilized in a given API and/or API design implementation.

Naming

A first guiding principle is to reserve simple and intuitive names for types that users need to use (e.g., instantiate) in the most common scenarios. Designers often "squander" the best names for abstractions, with which most users do not have to be concerned. For example, naming an abstract base class "File" and then providing a concrete type "XYZFile" works well if the expectation is that all users will have to understand the inheritance hierarchy before they can start using the APIs. However, if users do not understand the hierarchy, the first thing they will

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

likely try to use, most often unsuccessfully, is the "File" type. More specifically, the most common or expected names are reserved for aggregate components targeting the top core scenarios with less common or familiar names being used on concepts and abstractions.

A second guiding principle is to use descriptive identifier names that clearly state what each method does and what each type and parameter represents. For example, API designers should not hesitate to be rather verbose when choosing identifier names. For instance, "EventLog.DeleteEventSource(string source, string machineName)" may be seen as rather verbose, but it arguably has a net positive usability value. Moreover, type and parameter names state what a type or a parameter represents, instead of what it does. Method names state what the method does. Of course, accurate verbose method names are easier for methods that have simple and clear semantics, which is another reason why avoiding complex semantics is a good general design principle to follow.

A guiding design practice is to include a discussion about naming choices as a significant part of API specification reviews and/or tests. Exemplary considerations and questions include: What are the types most scenarios start with? What are the names most people think of first when trying to implement a given scenario? Are the names of the common types what users think of first? For example, since "File" is the name most people think of when dealing with file I/O scenarios, the aggregate component for accessing files can be named "File". Additionally, the most commonly used methods of the most commonly used types and their parameters are reviewed and tested. For example, can anybody familiar with the technology, but not the specific API design under consideration, recognize and call those methods quickly, correctly, and easily?

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Exceptions

As indicated above, exceptions can facilitate self-documenting APIs. In other words, APIs should lead the user to do the next required thing, and exceptions are capable of and good for communicating what is required next. For example, the following sample code throws an exception with a message "The 'FileName' property needs to be set before attempting to open the 'FileObject'.":

```
'VB
'Instantiate
Dim File As New FileObject()
'The file name is not set.

File.Open()
```

Strong Typing

Another guiding principle for facilitating intuitive APIs is strong typing. For example, calling "Customer.Name" is easier than calling "Customer.Properties['Name']". Furthermore, having such a "Name" property return the name as a string is more usable than if the property returned an object.

There are cases where property bags with a string based accessor, late bind calls, and other not strongly types APIs are desired, but they are relegated to rarity and are not the rule. Moreover, API designers can provide strongly typed helpers for the more common operations that the user performs on the non-strongly typed API layer. For example, a customer type may have a property bag, but it may additionally provide strongly typed APIs for more common properties like "name", "address", and so forth.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

Vectoring toward Simplicity

Yet another guiding principle is to strive for simplicity, especially for core scenarios. Standard design methodologies are aimed at producing designs that are optimized for maintainability, such as by using abstractions. Consequently, modern design methodologies produce a lot of abstractions. An issue is that such design methodologies operate on an assumption that users of the resulting designs will become experts in that design before starting to implement even simple scenarios. However, that is often not the cases in the real world.

In a described implementation, for at least simple scenarios, API designers ensure that object model hierarchies are sufficiently simple so that they can be used without having to understand how the entire feature area fits together or interoperates. A resulting well-designed API may require that the developer understand the core scenario being implemented, but it does not require a full understanding of the design of the library being used to implement it.

Generally, core scenario APIs are directed or correspond to physical or well-known logical parts of the system instead of abstractions. Types that correspond to abstractions are usually difficult to use without understanding how all the parts of the feature area fit together and interoperate; they are therefore more relevant when cross-feature integration is required.

Another guiding practice is to use standard design methodologies (e.g., UML) when designing internal architectures and some of the factored types, but not when designing the APIs for the core or common scenarios (e.g., those with aggregate components). When designing aggregate components for core scenarios, scenario driven design together with prototyping, usability studies, and iteration (as described herein above) is employed instead.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

Clean Namespaces

Yet another guiding principle is that types that are (very) rarely used are placed in sub-namespaces to avoid clutter of the main namespaces. For example, the following two groups of types may be separated from their main namespaces: permission types and design types. For instance, permission types can reside in a ".Permissions" sub-namespace, and design-time-only types can reside in a ".Design" sub-namespace.

The actions, aspects, features, components, etc. of FIGS. 1-9 are illustrated in diagrams that are divided into multiple blocks. However, the order, interconnections, interrelationships, layout, etc. in which FIGS. 1-9 are described and/or shown is not intended to be construed as a limitation, and any number of the blocks can be modified, combined, rearranged, augmented, omitted, etc. in any manner to implement one or more systems, methods, devices, procedures, media, APIs, apparatuses, arrangements, etc. for designing APIs. Furthermore, although the description herein includes references to specific implementations (and the exemplary operating environment of FIG. 10), the illustrated and/or described implementations can be implemented in any suitable hardware, software, firmware, or combination thereof and using any suitable software architecture(s), coding language(s), scenario definitions(s), usability study format(s), and so forth.

Exemplary Operating Environment for Computer or Other Device

FIG. 10 illustrates an exemplary computing (or general device) operating environment 1000 that is capable of (fully or partially) implementing at least one system, device, apparatus, component, arrangement, protocol, approach, method, procedure, media, API, some combination thereof, etc. for designing APIs as

Eliminado: MSI-1748US PATAPP (2).DOC
Insertado: MSI-1748US PATAPP (2).DOC
Eliminado: MSI-1748US.PATAPP

described herein. Operating environment 1000 may be utilized in the computer and network architectures described below.

Exemplary operating environment 1000 is only one example of an environment and is not intended to suggest any limitation as to the scope of use or functionality of the applicable device (including computer, network node, entertainment device, mobile appliance, general electronic device, etc.) architectures. Neither should operating environment 1000 (or the devices thereof) be interpreted as having any dependency or requirement relating to any one or to any combination of components as illustrated in FIG. 10.

Additionally, designing APIs and/or the APIs resulting therefrom may be implemented with numerous other general purpose or special purpose device (including computing system) environments or configurations. Examples of well known devices, systems, environments, and/or configurations that may be suitable for use include, but are not limited to, personal computers, server computers, thin clients, thick clients, personal digital assistants (PDAs) or mobile telephones, watches, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set-top boxes, programmable consumer electronics, video game machines, game consoles, portable or handheld gaming units, network PCs, minicomputers, mainframe computers, network nodes, distributed or multi-processing computing environments that include any of the above systems or devices, some combination thereof, and so forth.

Implementations for the design of APIs and/or the APIs resulting therefrom may be described in the general context of processor-executable instructions. Generally, processor-executable instructions include routines, programs, modules, protocols, objects, interfaces, components, data structures, etc. that perform and/or

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

enable particular tasks and/or implement particular abstract data types. Designing APIs and/or the APIs resulting therefrom, as described in certain implementations herein, may also be practiced and/or present in distributed processing environments where tasks are performed by remotely-linked processing devices that are connected through a communications link and/or network. Especially but not exclusively in a distributed computing environment, processor-executable instructions may be located in separate storage media, executed by different processors, and/or propagated over transmission media.

Exemplary operating environment 1000 *includes* a general-purpose computing device in the form of a computer 1002, which may comprise any (e.g., electronic) device with computing/processing capabilities. The components of computer 1002 may include, but are not limited to, one or more processors or processing units 1004, a system memory 1006, and a system bus 1008 that couples various system components including processor 1004 to system memory 1006.

Processors 1004 are not limited by the materials from which they are formed or the processing mechanisms employed therein. For example, processors 1004 may be comprised of semiconductor(s) and/or transistors (e.g., electronic integrated circuits (ICs)). In such a context, processor-executable instructions may be electronically-executable instructions. Alternatively, the mechanisms of or for processors 1004, and thus of or for computer 1002, may include, but are not limited to, quantum computing, optical computing, mechanical computing (e.g., using nanotechnology), and so forth.

System bus 1008 represents one or more of any of many types of wired or wireless bus structures, including a memory bus or memory controller, a point-to-point connection, a switching fabric, a peripheral bus, an accelerated graphics port,

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

and a processor or local bus using any of a variety of bus architectures. By way of example, such architectures may 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, a Peripheral Component Interconnects (PCI) bus also known as a Mezzanine bus, some combination thereof, and so forth.

Computer 1002 typically includes a variety of processor-accessible media. Such media may be any available media that is accessible by computer 1002 or another (e.g., electronic) device, and it includes both volatile and non-volatile media, removable and non-removable media, and storage and transmission media.

System memory 1006 includes processor-accessible storage media in the form of volatile memory, such as random access memory (RAM) 1040, and/or non-volatile memory, such as read only memory (ROM) 1012. A basic input/output system (BIOS) 1014, containing the basic routines that help to transfer information between elements within computer 1002, such as during start-up, is typically stored in ROM 1012. RAM 1010 typically contains data and/or program modules/instructions that are immediately accessible to and/or being presently operated on by processing unit 1004.

Computer 1002 may also include other removable/non-removable and/or volatile/non-volatile storage media. By way of example, FIG. 10 illustrates a hard disk drive or disk drive array 1016 for reading from and writing to a (typically) non-removable, non-volatile magnetic media (not separately shown); a magnetic disk drive 1018 for reading from and writing to a (typically) removable, non-volatile magnetic disk 1020 (e.g., a "floppy disk"); and an optical disk drive 1022 for reading from and/or writing to a (typically) removable, non-volatile optical

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

disk 1024 such as a CD, DVD, or other optical media. Hard disk drive 1016, magnetic disk drive 1018, and optical disk drive 1022 are each connected to system bus 1008 by one or more storage media interfaces 1026. Alternatively, hard disk drive 1016, magnetic disk drive 1018, and optical disk drive 1022 may be connected to system bus 1008 by one or more other separate or combined interfaces (not shown).

The disk drives and their associated processor-accessible media provide non-volatile storage of processor-executable instructions, such as data structures, program modules, and other data for computer 1002. Although exemplary computer 1002 illustrates a hard disk 1016, a removable magnetic disk 1020, and a removable optical disk 1024, it is to be appreciated that other types of processor-accessible media may store instructions that are accessible by a device, such as magnetic cassettes or other magnetic storage devices, flash memory, compact disks (CDs), digital versatile disks (DVDs) or other optical storage, RAM, ROM, electrically-erasable programmable read-only memories (EEPROM), and so forth. Such media may also include so-called special purpose or hard-wired IC chips. In other words, any processor-accessible media may be utilized to realize the storage media of the exemplary operating environment 1000.

Any number of program modules (or other units or sets of instructions/code, including an API framework and/or objects based thereon) may be stored on hard disk 1016, magnetic disk 1020, optical disk 1024, ROM 1012, and/or RAM 1040, including by way of general example, an operating system 1028, one or more application programs 1030, other program modules 1032, and program data 1034.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

A user may enter commands and/or information into computer 1002 via input devices such as a keyboard 1036 and a pointing device 1038 (e.g., a "mouse"). Other input devices 1040 (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 processing unit 1004 via input/output interfaces 1042 that are coupled to system bus 1008. However, input devices and/or output devices may instead be connected by other interface and bus structures, such as a parallel port, a game port, a universal serial bus (USB) port, an infrared port, an IEEE 1394 ("Firewire") interface, an IEEE 802.11 wireless interface, a Bluetooth® wireless interface, and so forth.

A monitor/view screen 1044 or other type of display device may also be connected to system bus 1008 via an interface, such as a video adapter 1046. Video adapter 1046 (or another component) may be or may include a graphics card for processing graphics-intensive calculations and for handling demanding display requirements. Typically, a graphics card includes a graphics processing unit (GPU), video RAM (VRAM), etc. to facilitate the expeditious display of graphics and performance of graphics operations. In addition to monitor 1044, other output peripheral devices may include components such as speakers (not shown) and a printer 1048, which may be connected to computer 1002 via input/output interfaces 1042.

Computer 1002 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computing device 1050. By way of example, remote computing device 1050 may be a personal computer, a portable computer (e.g., laptop computer, tablet computer, PDA, mobile station, etc.), a palm or pocket-sized computer, a watch, a gaming device, a

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

server, a router, a network computer, a peer device, another network node, or another device type as listed above, and so forth. However, remote computing device 1050 is illustrated as a portable computer that may include many or all of the elements and features described herein with respect to computer 1002.

Logical connections between computer 1002 and remote computer 1050 are depicted as a local area network (LAN) 1052 and a general wide area network (WAN) 1054. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, the Internet, fixed and mobile telephone networks, ad-hoc and infrastructure wireless networks, other wireless networks, gaming networks, some combination thereof, and so forth. Such networks and communications connections are examples of transmission media.

When implemented in a LAN networking environment, computer 1002 is usually connected to LAN 1052 via a network interface or adapter 1056. When implemented in a WAN networking environment, computer 1002 typically includes a modem 1058 or other component for establishing communications over WAN 1054. Modem 1058, which may be internal or external to computer 1002, may be connected to system bus 1008 via input/output interfaces 1042 or any other appropriate mechanism(s). It is to be appreciated that the illustrated network connections are exemplary and that other manners for establishing communication link(s) between computers 1002 and 1050 may be employed.

In a networked environment, such as that illustrated with operating environment 1000, program modules or other instructions that are depicted relative to computer 1002, or portions thereof, may be fully or partially stored in a remote media storage device. By way of example, remote application programs 1060 reside on a memory component of remote computer 1050 but may be usable

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

or otherwise accessible via computer 1002. Also, for purposes of illustration, application programs 1030 and other processor-executable instructions such as operating system 1028 are illustrated herein as discrete blocks, but it is recognized that such programs, components, and other instructions reside at various times in different storage components of computing device 1002 (and/or remote computing device 1050) and are executed by processor(s) 1004 of computer 1002 (and/or those of remote computing device 1050).

Although systems, media, devices, methods, procedures, apparatuses, techniques, APIs, schemes, approaches, procedures, arrangements, and other implementations have been described in language specific to structural, logical, algorithmic, functional, and action-based features and/or diagrams, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or diagrams described. Rather, the specific features and diagrams are disclosed as exemplary forms of implementing the claimed invention.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

CLAIMS

1. A method for designing an application programming interface (API), the method comprising:

preparing a plurality of code samples for a core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages; and

deriving the API from the core scenario responsive to the plurality of code samples.

2. The method as recited in claim 1, further comprising:

determining, by an API designer, if the derived API is too complex.

3. The method as recited in claim 2, further comprising:

if the derived API is determined to be too complex, then

refining, by the API designer, the derived API to produce a refined API.

4. The method as recited in claim 3, further comprising:

determining, by the API designer, if the refined API is too complex.

5. The method as recited in claim 1, further comprising:

performing one or more usability studies on the API utilizing a plurality of developers.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

27d

6. The method as recited in claim 5, wherein the performing comprises:

performing the one or more usability studies on the API utilizing the plurality of developers wherein the plurality of developers are typical for the plurality of programming languages.

7. The method as recited in claim 5, further comprising:

ascertaining whether the plurality of developers are able to use the API without significant problems.

8. The method as recited in claim 7, further comprising:

if the plurality of developers are not ascertained to be able to use the API without significant problems, then revising the API.

9. The method as recited in claim 8, wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies.

10. The method as recited in claim 1, wherein the deriving comprises:

deriving the API to support the plurality of code samples that correspond respectively to the plurality of programming languages.

Eliminado: MS1-1748US PATAPP (3).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

11. The method as recited in claim 1, wherein the deriving comprises:

gleaning language-specific mandates from the plurality of code samples; and

incorporating the language-specific mandates into the API.

12. The method as recited in claim 1, wherein the deriving comprises:

gleaning language-inspired developer expectations from the plurality of code samples; and

incorporating the language-inspired developer expectations into the API.

13. The method as recited in claim 1, wherein the deriving comprises:

gleaning commonalities from the plurality of code samples; and

incorporating the commonalities into the API.

14. The method as recited in claim 1, wherein the deriving comprises:

deriving the API to have an aggregate component that ties a plurality of lower-level factored types together to support the core scenario.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

15. A method for designing an application programming interface (API), the method comprising:

selecting a core scenario for a feature area;

writing at least one code sample for the core scenario; and

deriving an API for the core scenario responsive to the at least one code sample.

16. The method as recited in claim 15, wherein the selecting comprises:

selecting a plurality of core scenarios for the feature area.

17. The method as recited in claim 16, further comprising:

repeating the writing and the deriving for each core scenario of the plurality of core scenarios that are selected for the feature area.

18. The method as recited in claim 15, wherein the writing comprises:

writing a plurality of code samples for the core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages.

19. The method as recited in claim 18, wherein the deriving comprises:

deriving the API for the core scenario responsive to the plurality of code samples.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

20. The method as recited in claim 15, further comprising:

performing one or more usability studies on the API utilizing a plurality of developers;

ascertaining whether the plurality of developers are able to use the API without significant problems; and

if the plurality of developers are not ascertained to be able to use the API without significant problems, then revising the API.

21. The method as recited in claim 20, wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies to produce a revised API.

22. The method as recited in claim 21, further comprising:

repeating the performing and the ascertaining with respect to the revised API.

23. The method as recited in claim 15, wherein the deriving comprises:

deriving the API to support the at least one code sample written for the core scenario by producing a two-layer API that includes an aggregate component and a plurality of underlying factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-.748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

24. The method as recited in claim 15, wherein the deriving comprises:

gleaning one or more language-specific mandates from the at least one code sample; and

incorporating the one or more language-specific mandates into the API.

25. The method as recited in claim 15, wherein the deriving comprises:

encapsulating a particular factored type into an aggregate component that is associated with the core scenario if all members of the particular factored type are exposed by the aggregate component.

26. The method as recited in claim 15, wherein the deriving comprises:

encapsulating a particular factored type into an aggregate component that is associated with the core scenario if the particular factored type is independently uninteresting to other component types.

27. The method as recited in claim 15, wherein the deriving comprises:

exposing a particular factored type from an aggregate component that is associated with the core scenario if at least one member of the particular factored type is not exposed by the aggregate component.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

28. The method as recited in claim 15, wherein the deriving comprises:

exposing a particular factored type from an aggregate component that is associated with the core scenario if the particular factored type can be beneficially used independently of the aggregate component by another component type.

29. The method as recited in claim 15, wherein the deriving comprises:

producing a two-layer framework that includes component types targeting a relatively higher level of abstraction and component types targeting a relatively lower level of abstraction.

30. The method as recited in claim 29, wherein the component types targeting the relatively higher level of abstraction are directed to core scenarios.

31. The method as recited in claim 29, wherein the component types targeting the relatively lower level of abstraction provide a relatively greater amount of control to developers as compared to the component types targeting the relatively higher level of abstraction.

32. The method as recited in claim 15, wherein the deriving comprises:

deriving the API so as to enable a developer to implement a create-set-call usage pattern for the core scenario.

Eliminado: MS1-1748US PATAPP (2).DOC
Insertado: MS1-1748US PATAPP (2).DOC
Eliminado: MS1-1748US.PATAPP

33. The method as recited in claim 32, wherein the deriving comprises:

producing the API with pre-selected parameters that are appropriate for the core scenario.

34. A method for designing an application programming interface (API), the method comprising:

deriving an API for a scenario responsive to at least one code sample written with regard to the scenario;

performing one or more usability studies on the API utilizing a plurality of developers; and

revising the API based on the one or more usability studies.

35. The method as recited in claim 34, further comprising:

writing a plurality of code samples with regard to the scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages;

wherein the deriving comprises:

deriving the API for the scenario responsive to the plurality of code samples.

Eliminado: MSI-1748US PATAPP (2).DOC

Insertado: MSI-1748US PATAPP (2).DOC

Eliminado: MSI-1748USLP/.TAPP

36. The method as recited in claim 34, further comprising, prior to the performing one or more usability studies on the API:

determining, by an API designer, if the derived API is too complex;

if the derived API is determined to be too complex, then

refining, by the API designer, the derived API to produce a refined API; and

determining, by the API designer, if the refined API is too complex.

37. The method as recited in claim 34, further comprising:

ascertaining whether the plurality of developers are able to use the API without significant problems; and

when the plurality of developers are not ascertained to be able to use the API without significant problems, then implementing the revising;

wherein the revising comprises:

revising the API based on at least one lesson from the one or more usability studies to produce a revised API.

38. The method as recited in claim 37, further comprising:

repeating at least the performing and the ascertaining with respect to the revised API.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

39. The method as recited in claim 37, wherein the ascertaining comprises:

ascertaining whether the plurality of developers are able to use the API without significant problems with regard to a desired level of usability for at least one targeted developer group, wherein the desired level of usability includes considerations with respect to (i) frequent and/or extensive reference to detailed API documentation by the plurality of developers, (ii) a failure of a majority of the plurality of developers to implement the scenario, and (iii) whether the plurality of developers take an approach that is significantly different from what is expected by an API designer.

40. The method as recited in claim 34, further comprising:

selecting a plurality of core scenarios for a feature area;

repeating the deriving, the performing, and the revising for each core scenario of the plurality of core scenarios.

41. The method as recited in claim 40, wherein the deriving comprises:

producing an aggregate component for each core scenario of the plurality of core scenarios.

42. The method as recited in claim 34, wherein the deriving comprises:

producing an aggregate component that has a respective relationship with each respective factored type of a plurality of factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

43. The method as recited in claim 42, wherein the producing comprises:

producing the aggregate component to support the scenario for which the at least one code sample is written.

44. The method as recited in claim 42, wherein the producing comprises:

producing the aggregate component to have an exposed relationship with at least one factored type of the plurality of factored types and an encapsulated relationship with at least one other factored type of the plurality of factored types.

45. The method as recited in claim 44, wherein the at least one other factored type of the plurality of factored types that has the encapsulated relationship with the aggregate component can be handed off by the aggregate component for direct interaction with another component type.

46. The method as recited in claim 42, wherein the plurality of factored types are designed using an object-oriented methodology.

Eliminado: MS1-1748US PATAPP (.).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

47. A method for designing an application programming interface (API), the method comprising:

preparing a plurality of code samples for a core scenario, each respective code sample of the plurality of code samples corresponding to a respective programming language of a plurality of programming languages;

deriving the API for the core scenario responsive to the plurality of code samples;

performing one or more usability studies on the API utilizing a plurality of developers; and

revising the API based on the one or more usability studies.

48. A method for designing an application programming interface (API), the method comprising:

writing at least one code sample for a scenario; and

deriving an API for the scenario responsive to the at least one code sample, the API including (i) an aggregate component that is adapted to facilitate implementation of the scenario and (ii) a plurality of factored types that provide underlying functionality for the aggregate component, the API enabling a gradual progression from using the aggregate component in simpler situations to using an increasing portion of the plurality of factored types in increasingly complex situations.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

49. A method for designing an application programming interface (API), the method comprising:

deriving at least one aggregate component to support at least one code sample for at least one scenario;

determining additional requirements with respect to the at least one scenario;

deciding if the additional requirements can be added to the at least one aggregate component without adding undue complexity to the at least one scenario; and

if not, defining a plurality of factored types responsive to the deciding.

50. The method as recited in claim 49, further comprising:

if so, refining the at least one aggregate component to incorporate the additional requirements.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

51. The method as recited in claim 49, further comprising:

selecting a plurality of core scenarios for a feature area, the plurality of core scenarios including the at least one scenario; and

writing a plurality of code samples showing preferred lines of code for the plurality of core scenarios, the plurality of code samples including the at least one code sample;

wherein the deriving comprises:

deriving a plurality of aggregate components, which include the at least one aggregate component, to support the plurality of code samples for the plurality of core scenarios.

52. The method as recited in claim 49, wherein the deriving comprises:

deriving the at least one aggregate component with appropriate methods, defaults, and abstractions to support the at least one code sample for the at least one scenario.

53. The method as recited in claim 49, further comprising:

refining the at least one code sample according to the at least one derived aggregate component; and

evaluating the refined at least one code sample with regard to simplicity; and

repeating the deriving if the refined at least one code sample fails to be sufficiently simple in the evaluating.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

54. The method as recited in claim 49, wherein the determining comprises:

determining additional requirements with respect to the at least one scenario, wherein the additional requirements include additional scenarios, additional usages, and additional interactions with other component types.

55. The method as recited in claim 49, wherein the deciding comprises:

considering whether adding the additional requirements to the at least one aggregate component hinders a create-set-call usage pattern.

56. The method as recited in claim 49, wherein the defining comprises:

defining the plurality of factored types responsive to the deciding with an ideal factoring of a full set of functionality.

57. The method as recited in claim 49, wherein the defining comprises:

defining the plurality of factored types responsive to the deciding using one or more object-oriented methodologies.

58. The method as recited in claim 49, further comprising:

determining whether the at least one aggregate component is to encapsulate or expose the functionality of each factored type of the plurality of factored types.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

59. The method as recited in claim 49, further comprising:

refining the plurality of factored types to support the at least one aggregate component and the additional requirements.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (2).DOC

Eliminado: MS1-1748US.PATAPP

ABSTRACT

A first exemplary method implementation for designing an application programming interface (API) includes: preparing multiple code samples for a core scenario, each respective code sample of the multiple code samples corresponding to a respective programming language of multiple programming languages; and deriving the API from the core scenario responsive to the multiple code samples. A second exemplary method for designing an API includes: selecting a core scenario for a feature area; writing at least one code sample for the core scenario; and deriving an API for the core scenario responsive to the at least one code sample. A third exemplary method for designing an API includes: deriving an API for a scenario responsive to at least one code sample written with regard to the scenario; performing one or more usability studies on the API utilizing multiple developers; and revising the API based on the one or more usability studies.

Eliminado: MS1-1748US PATAPP (2).DOC

Insertado: MS1-1748US PATAPP (3).DOC

Eliminado: MS1-1748US.PATAPP

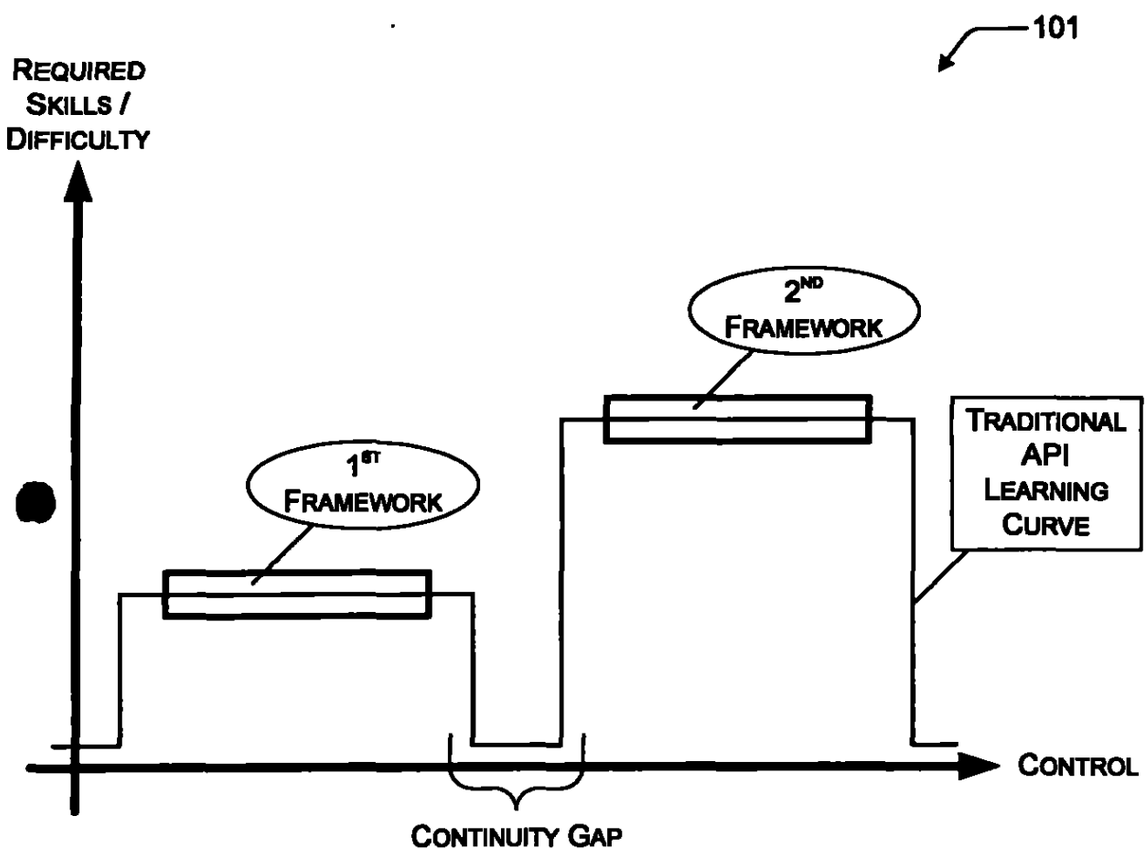


FIG. 1 Prior Art

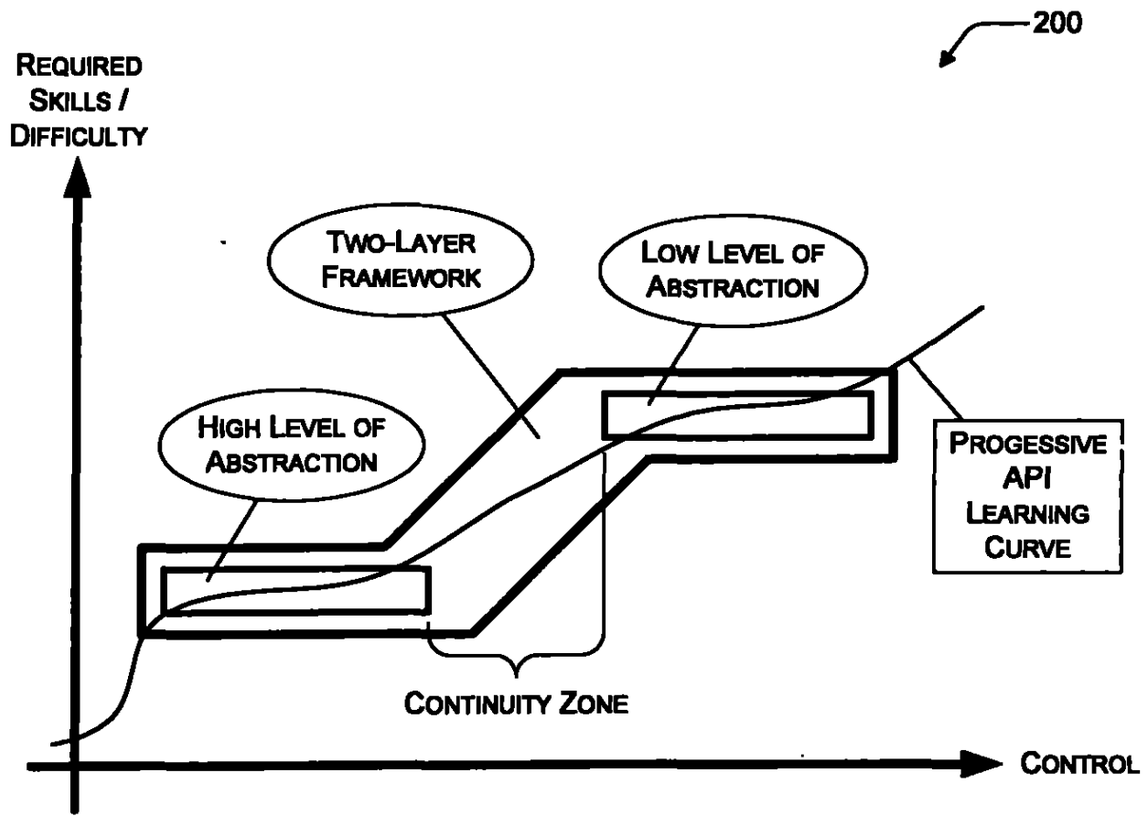

200
REQUIRED SKILLS / DIFFICULTY
TWO-LAYER FRAMEWORK
LOW LEVEL OF ABSTRACTION
HIGH LEVEL OF ABSTRACTION
PROGESSIVE API LEARNING CURVE
CONTINUITY ZONE
CONTROL

FIG. 2

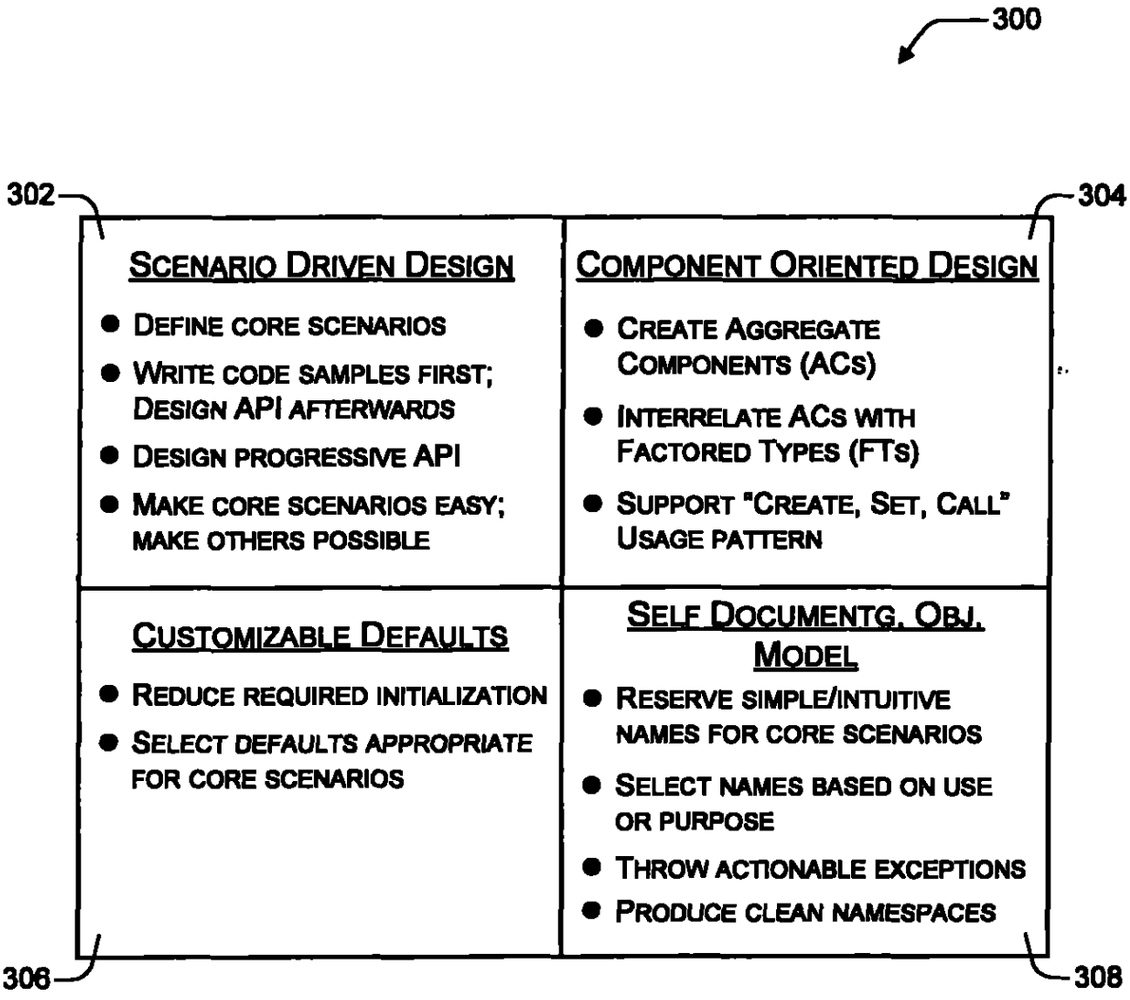

300
302
304
SCENARIO DRIVEN DESIGN
DEFINE CORE SCENARIOS
WRITE CODE SAMPLES FIRST; DESIGN API AFTERWARDS
DESIGN PROGRESSIVE API
MAKE CORE SCENARIOS EASY; MAKE OTHERS POSSIBLE
COMPONENT ORIENTED DESIGN
CREATE AGGREGATE COMPONENTS (ACS)
INTERRELATE ACS WITH FACTORED TYPES (FTS)
SUPPORT "CREATE, SET, CALL" USAGE PATTERN
CUSTOMIZABLE DEFAULTS
REDUCE REQUIRED INITIALIZATION
SELECT DEFAULTS APPROPRIATE FOR CORE SCENARIOS
SELF DOCUMENTG. OBJ. MODEL
RESERVE SIMPLE/INTUITIVE NAMES FOR CORE SCENARIOS
SELECT NAMES BASED ON USE OR PURPOSE
THROW ACTIONABLE EXCEPTIONS
PRODUCE CLEAN NAMESPACES
306
308

FIG. 3

289

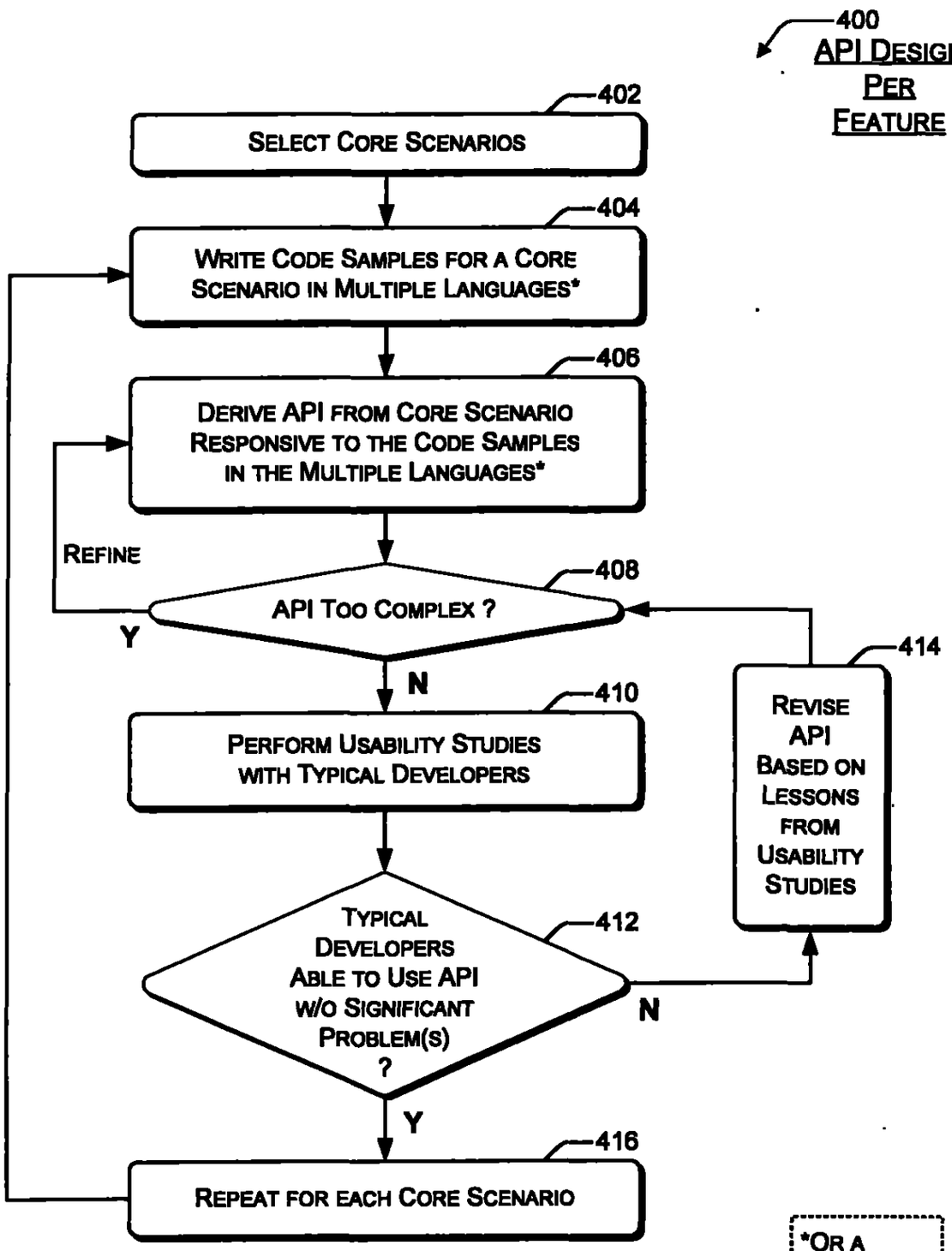

400
API DESIGN PER FEATURE
402
SELECT CORE SCENARIOS
404
WRITE CODE SAMPLES FOR A CORE SCENARIO IN MULTIPLE LANGUAGES*
406
DERIVE API FROM CORE SCENARIO RESPONSIVE TO THE CODE SAMPLES IN THE MULTIPLE LANGUAGES*
REFINE
408
API TOO COMPLEX ?
Y
N
410
PERFORM USABILITY STUDIES WITH TYPICAL DEVELOPERS
414
REVISE API BASED ON LESSONS FROM USABILITY STUDIES
412
TYPICAL DEVELOPERS ABLE TO USE API W/O SIGNIFICANT PROBLEM(S) ?
N
Y
416
REPEAT FOR EACH CORE SCENARIO
*OR A SINGLE LANGUAGE

FIG. 4

290

290

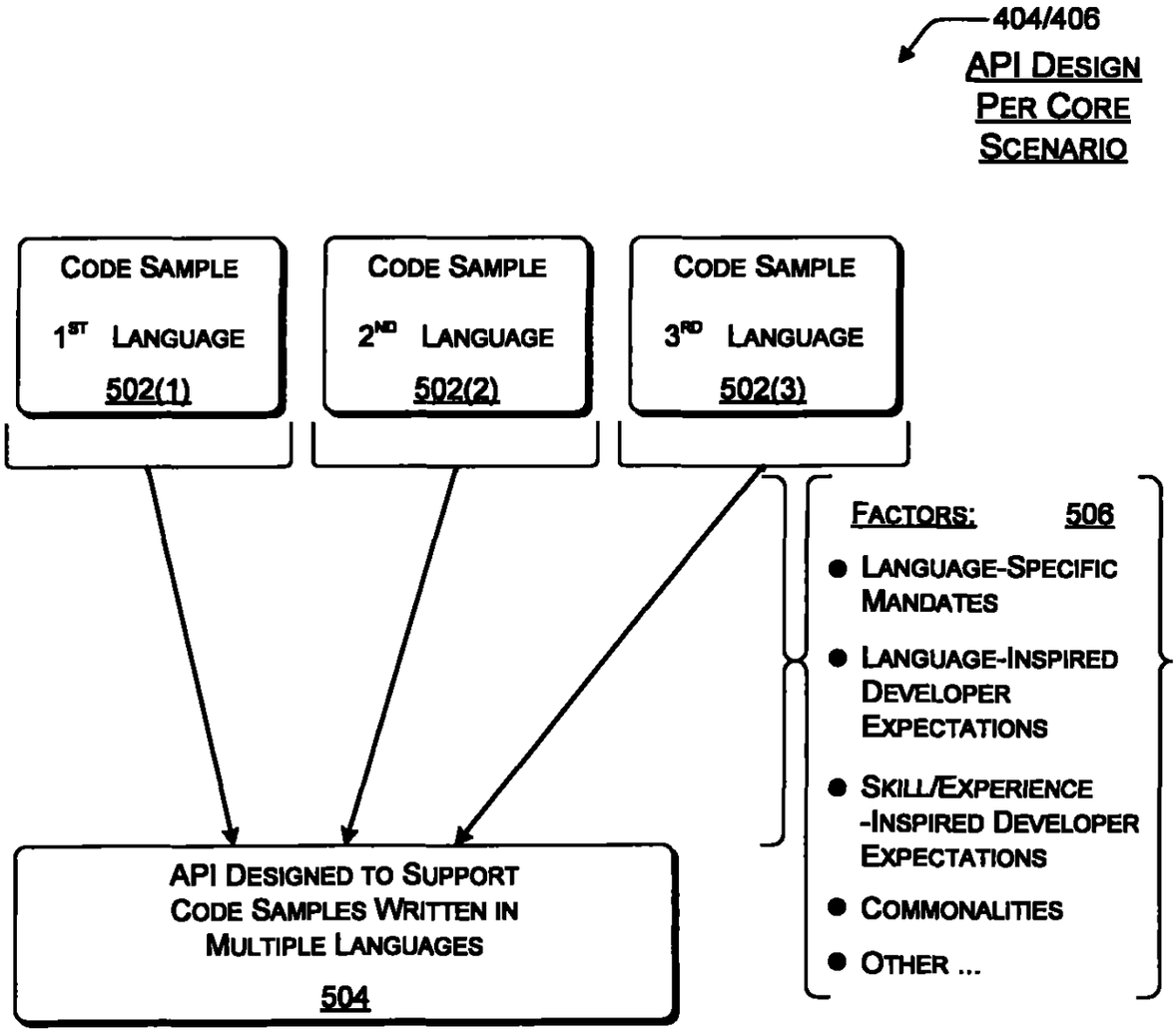

404/406
API DESIGN PER CORE SCENARIO
CODE SAMPLE 1ST LANGUAGE 502(1)
CODE SAMPLE 2ND LANGUAGE 502(2)
CODE SAMPLE 3RD LANGUAGE 502(3)
FACTORS: 506
LANGUAGE-SPECIFIC MANDATES
LANGUAGE-INSPIRED DEVELOPER EXPECTATIONS
SKILL/EXPERIENCE -INSPIRED DEVELOPER EXPECTATIONS
COMMONALITIES
OTHER ...
API DESIGNED TO SUPPORT CODE SAMPLES WRITTEN IN MULTIPLE LANGUAGES 504

FIG. 5

291

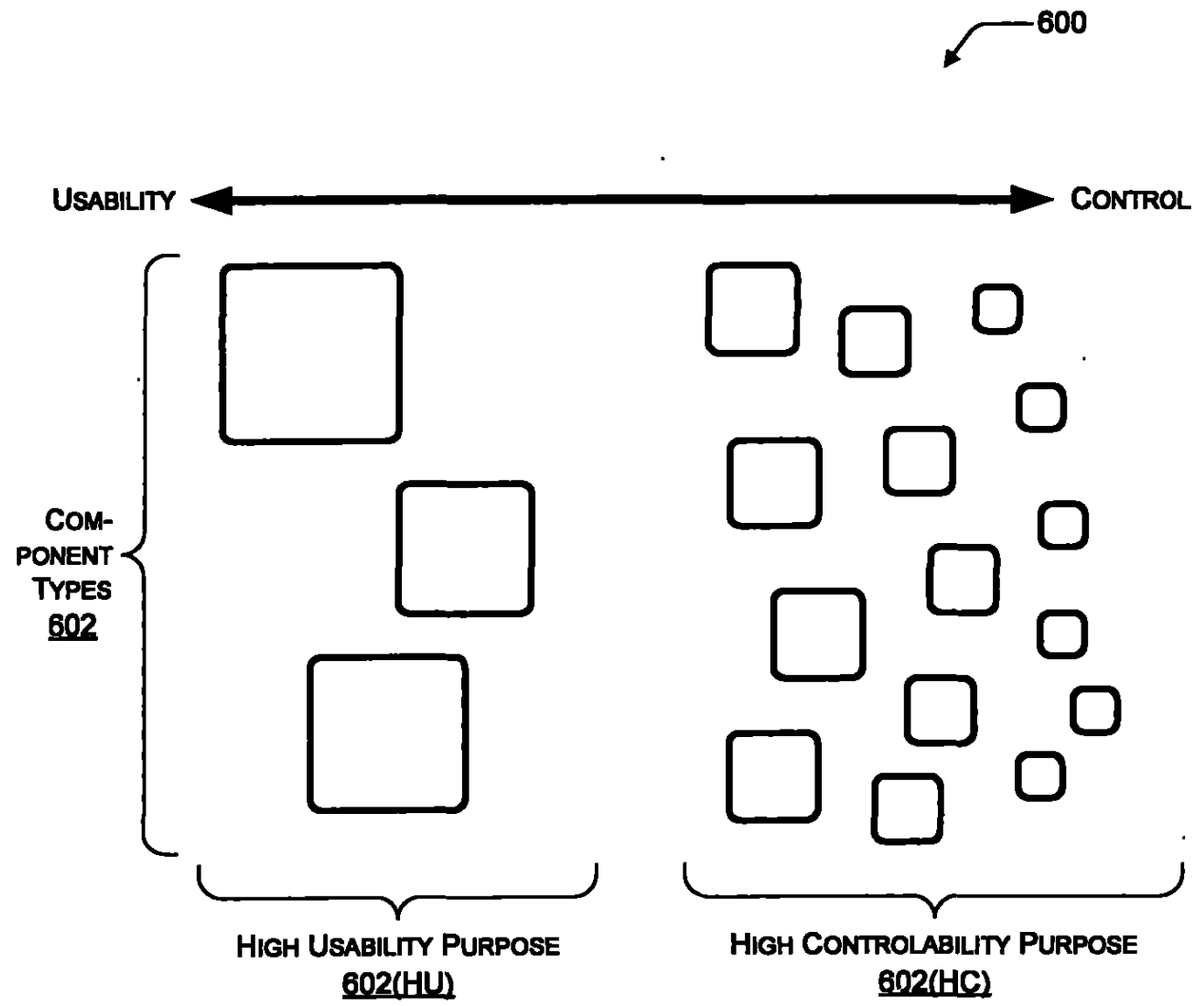

600
USABILITY
CONTROL
COM-
PONENT
TYPES
602
HIGH USABILITY PURPOSE
602(HU)
HIGH CONTROLABILITY PURPOSE
602(HC)

FIG. 6

292

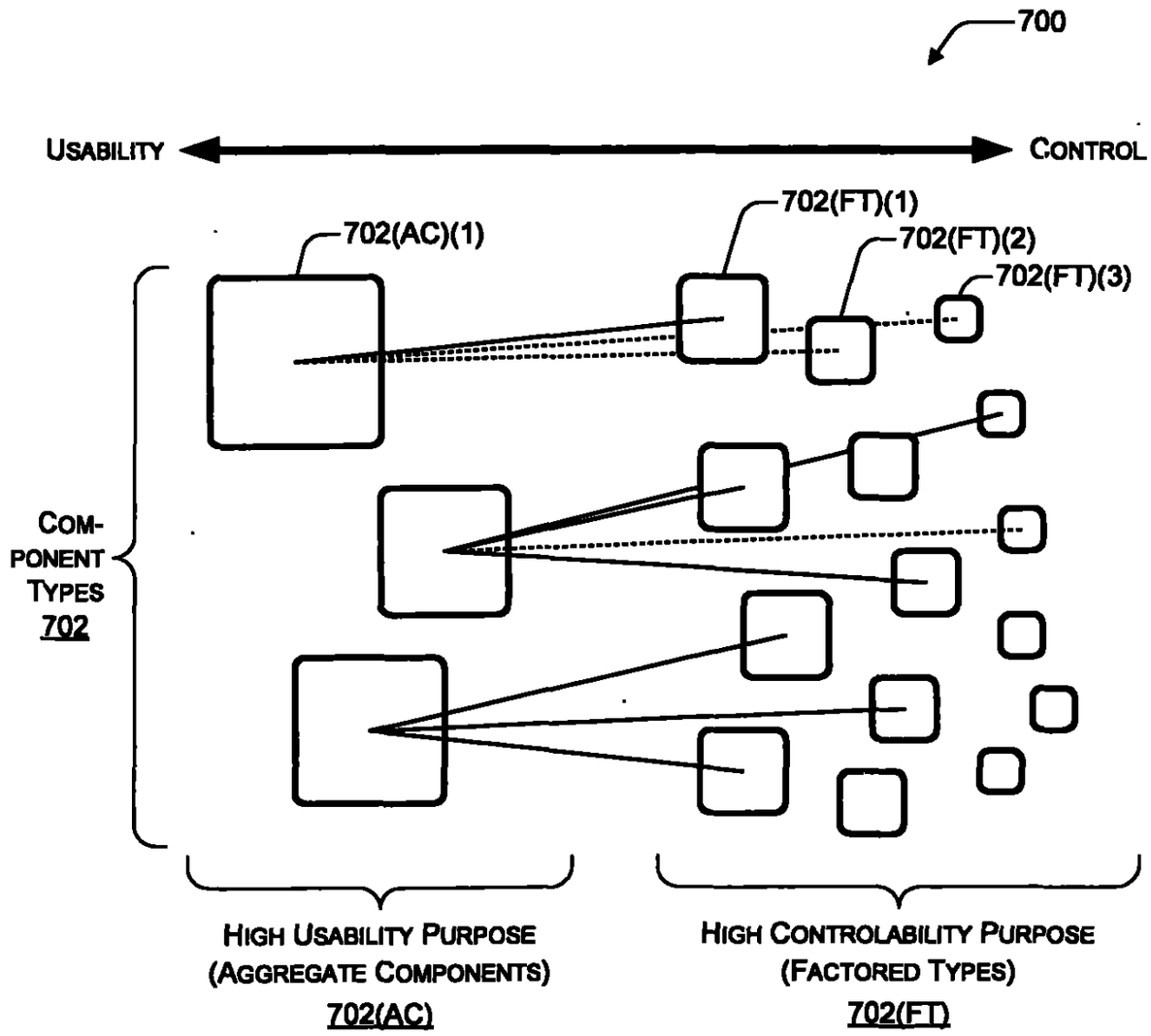


FIG. 7

KEY:	704
EXPOSED FACTORED TYPES	———
ENCAPSULATED FACTORED TYPES	.....

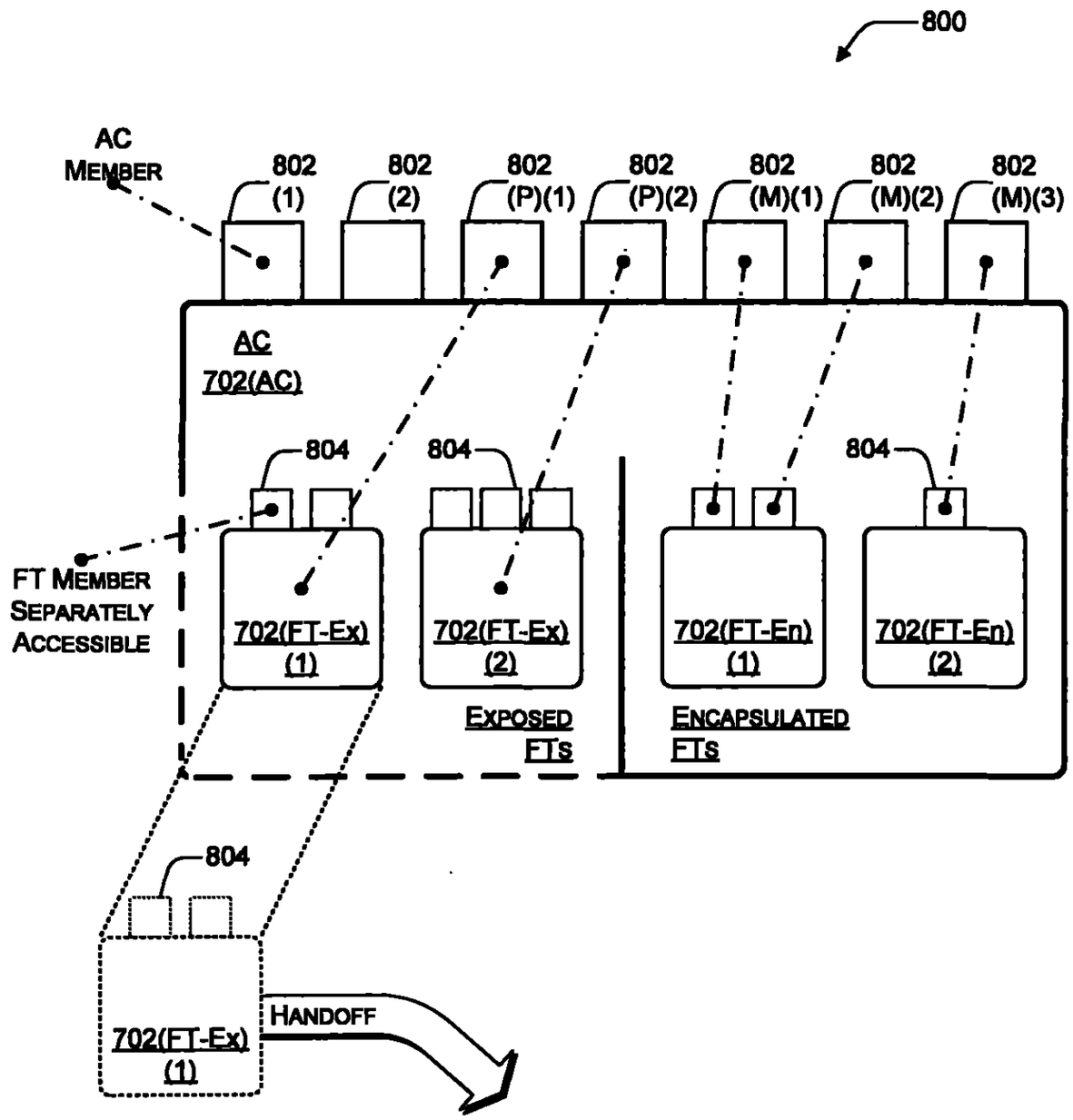

800
AC MEMBER
802 (1)
802 (2)
802 (P)(1)
802 (P)(2)
802 (M)(1)
802 (M)(2)
802 (M)(3)
AC
702(AC)
804
804
804
FT MEMBER SEPARATELY ACCESSIBLE
702(FT-Ex) (1)
702(FT-Ex) (2)
702(FT-En) (1)
702(FT-En) (2)
EXPOSED FTs
ENCAPSULATED FTs
804
HANDOFF
702(FT-Ex) (1)

FIG. 8

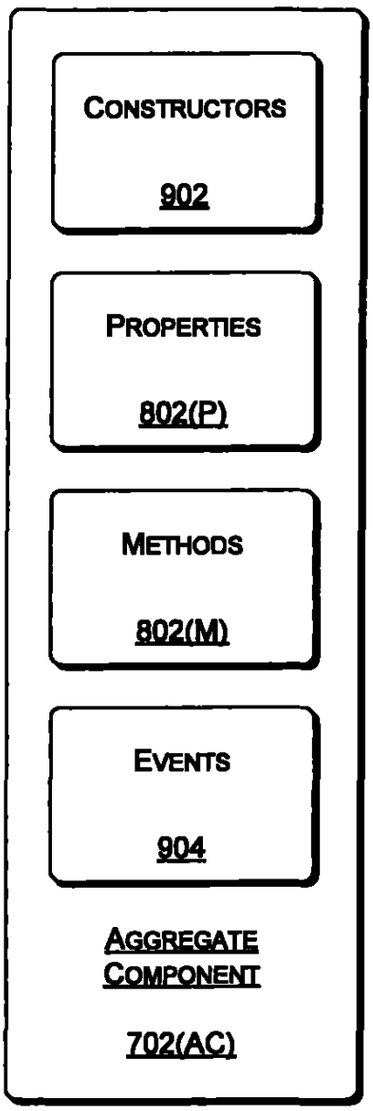
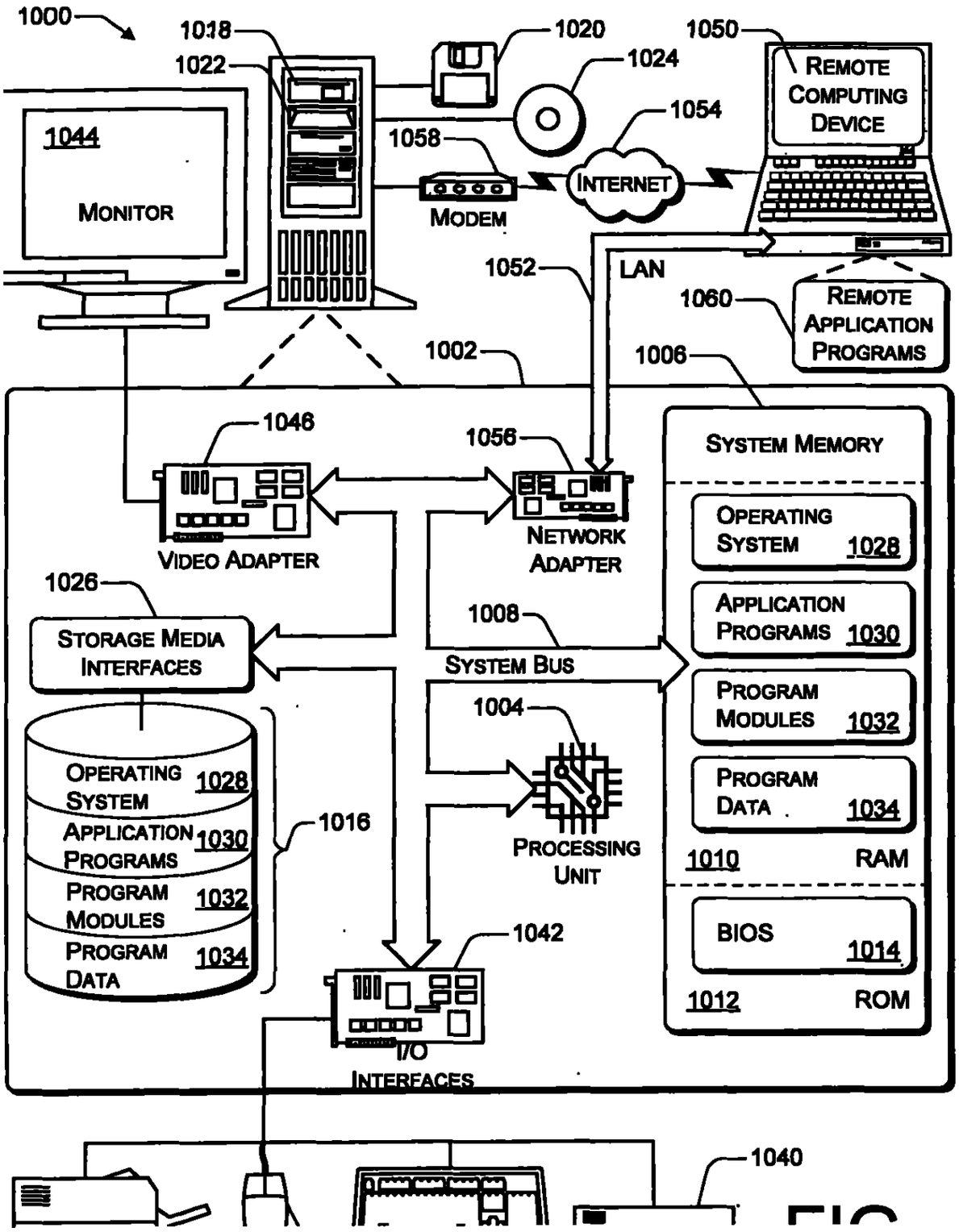

CONSTRUCTORS
902
PROPERTIES
802(P)
METHODS
802(M)
EVENTS
904
AGGREGATE COMPONENT
702(AC)

FIG. 9


1000
1018
1022
1020
1024
1050
REMOTE COMPUTING DEVICE
1054
1058
INTERNET
MODEM
1044
MONITOR
1052
LAN
1060
REMOTE APPLICATION PROGRAMS
1002
1006
1046
1056
SYSTEM MEMORY
VIDEO ADAPTER
NETWORK ADAPTER
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
1026
STORAGE MEDIA INTERFACES
1008
SYSTEM BUS
PROGRAM MODULES 1032
1004
PROGRAM DATA 1034
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1016
PROCESSING UNIT
1010 RAM
BIOS 1014
1012 ROM
1042
I/O INTERFACES
1040

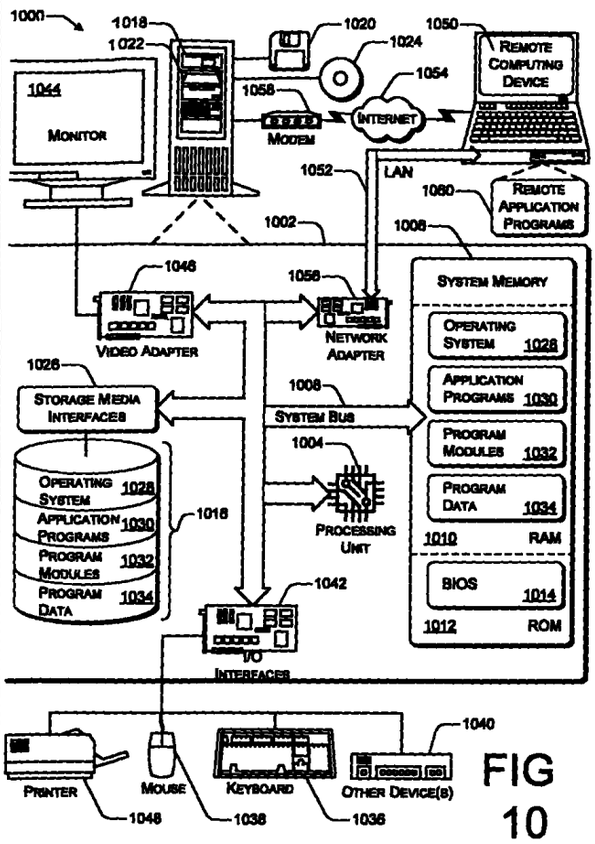

1000
1018
1022
1020
1024
1050
REMOTE COMPUTING DEVICE
1044
MONITOR
1058
1054
MODEM
INTERNET
1052
LAN
1060
REMOTE APPLICATION PROGRAMS
1002
1006
1046
1056
SYSTEM MEMORY
VIDEO ADAPTER
NETWORK ADAPTER
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1010 RAM
BIOS 1014
1012 ROM
1026
STORAGE MEDIA INTERFACES
1008
SYSTEM BUS
1004
PROCESSING UNIT
1018
OPERATING SYSTEM 1028
APPLICATION PROGRAMS 1030
PROGRAM MODULES 1032
PROGRAM DATA 1034
1042
I/O INTERFACES
1040
PRINTER
MOUSE
KEYBOARD
OTHER DEVICE(S)
1048
1038
1036

FIG 10

APOSTILLE

(Convention de La Haye du 5 octobre 1961)

1. Country: *United States of America*

This public document

2. has been signed by N. Williams

3. acting in the capacity of Certifying Officer

4. bears the seal/stamp of Patent and Trademark Office

Certified

5. at Washington, D.C.

6. the fifth of August, 2004

7. by *Assistant Authentication Officer, United States Department of State*

8. No. 04025116-1

9. Seal/Stamp:

10. Signature:

Fernesia T. Crawford

NOTARIA 16
CIRCULO DE BOGOTA D.C.
OTTO CARLOS GALVIS
27 AGO. 2004
ESTA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

A 1382229

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

August 03, 2004

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE RECORDS OF THIS OFFICE OF A DOCUMENT RECORDED ON MARCH 19, 2004

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

N. WILLIAMS
Certifying Officer

NOTARIA 16
OTTO BARRAGAN DE BOGOTA D.C.
27 AGO. 2004
ESTA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004A48
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Form PTO-1595 (Rev. 03/01)
OMB No. 0651-0027 (exp. 6/31/2002)

RECORDATION FORM COVER SHEET
PATENTS ONLY

U.S. DEPARTMENT OF COMMERCE
U.S. Patent and Trademark Office

Tab setting ⇨ ⇨ ⇨ ▼ ▼ ▼ ▼ ▼ ▼ ▼

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

1. Name of conveying party(ies):
Krzysztof J. Cwalina
Bradley Moore Abrams
Anthony J. Moore
Christopher L. Anderson
Additional name(s) of conveying party(ies) attached? ☐ Yes ☐ No

2. Name and address of receiving party(ies)
Name: Microsoft Corporation
Internal Address: _____
Street Address: One Microsoft Way
City: Redmond State: WA Zip: 98052-6399
Additional name(s) & address(es) attached? ☐ Yes ☒ No

3. Nature of conveyance:
☒ Assignment ☐ Merger
☐ Security Agreement ☐ Change of Name
☐ Other _____
Execution Date: 01/29/04; 02/03/04; 01/29/04

4. Application number(s) or patent number(s): 10/692,330
If this document is being filed together with a new application, the execution date of the application is: _____
A. Patent Application No.(s)
B. Patent No.(s)
Additional numbers attached? ☐ Yes ☒ No

5. Name and address of party to whom correspondence concerning document should be mailed:
Name: Keith W. Saunders, Reg. No. 41,462
Internal Address: Lee & Hayes, PLLC
Street Address: 421 West Riverside Avenue
Suite 500
City: Spokane State: WA Zip: 99201

6. Total number of applications and patents involved: 1

7. Total fee (37 CFR 3.41).....$ 40.00
☐ Enclosed
☒ Authorized to be charged to deposit account

8. Deposit account number: 12-0769
(Attach duplicate copy of this page if paying by deposit account)

DO NOT USE THIS SPACE

9. Statement and signature.
To the best of my knowledge and belief, the foregoing information is true and correct and any attached copy is a true copy of the original document.
Keith W. Saunders, Reg. No. 41,462
Name of Person Signing
Signature
3/19/2004
Date
Total number of pages including cover sheet, attachments, and documents: 14

Mail comments to be recorded with assigned, patent, and/or application no.
Mail Stop Assignment Recordation Services
Alexandria, VA 22313-1450

NOTARIO 16 BUENOS AIRES D.C.
ESTA FOTOCOPIA CONCUERDA CON LA VISTA
DOCTICO QUE TUVE A LA VISTA

700073326

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA
30.00

300

Recordation Form Cover Sheet
Page 2
Application number: 10/692,320

1. Name of conveying parties:

Michael Pizzo
Robert Allan Brigham II

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
ESTA FOTOCOPIA CONCUERDA CON UN DOCUMENTO IDENTICO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004-FR
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Attorney Docket. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,320

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial Number 10/692,320
Filing Date ..... Oct 23, 2003
Inventorship ..... Cwalina et al.
Applicant ..... Microsoft Corporation
Attorney's Docket No. MS1-1748US
MSDocket No. 305796.01

Title: Design of Application Programming Interfaces (APIs)

PATENT ASSIGNMENT

PARTIES TO THE ASSIGNMENT

Assignor(s):

Krzysztof J. Cwalina
7583 Old Redmond Road, Apt. A204
Redmond, WA 98052

Bradley Moore Abrams
7517 128th Pl. NE
Kirkland, WA 98033

Anthony J. Moore
4418 Corliss Avenue North, Apt. A
Seattle, WA 98112

Christopher L. Anderson
18612 SE 45th Street
Issaquah, WA 98027

Michael Pizzo
528 W Lake Sammamish PKWY SE
Bellevue, WA 98008

Robert Allan Brigham II
21226 NE 132nd Ct.
Woodinville, WA 98072

Assignee:

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

AGREEMENT

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
ESTA FOTOCOPIA COINCIDE CON UN DOCUMENTO IDENTICO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004AP12
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA

Attorney Docket No. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,330

WHEREAS, ASSIGNOR(S) (listed above) are inventor(s) of an invention entitled "Design of Application Programming Interfaces (APIs)," as described and claimed in the specification forming part of an application for United States letters patent referenced above;

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

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

NOTARIA 16 ESTADO DE CALIFORNIA CIUDAD DE BALBOA REPUBLICA DE PANAMA D.C. 27 AGO 2004 FIRMA CONFORME CON UN DOCUMENTO QUE SE TUVO A LA VISTA

TRADUCCION OFICIAL No. 2004A492
HILDA AGUILERA DE PEDRARIAS
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

303

Attorney Docket No. MSFT-1745US
MS Docket No. 305796.01
Serial No. 10/692,320

01-29-04
Date

Krzysztof J. Cwalina

01-29-04
Date

Bradley Moore Abrams

29-Jan-2004
Date

Anthony J. Moore

Date

Christopher L. Andersson

Date

Michael Fizzo

Date

Robert Allan Brigham II

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
ESTA FOTOCOPIA CONCUERDA CON UN DOCUMENTO IDENTICO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Attorney Docket No. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,320

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial Number..... 10/692,320
Filing Date..... Oct 23, 2003
Inventorship..... Cwalina et al.
Applicant..... Microsoft Corporation
Attorney's Docket No. MS1-1748US
MSDocket No..... 305796.01
Title: Design of Application Programming Interfaces (APIs)

PATENT ASSIGNMENT

PARTIES TO THE ASSIGNMENT

Assignor(s):

Krzysztof J. Cwalina
7583 Old Redmond Road, Apt. A204
Redmond, WA 98052

Bradley Moore Abrams
7517 128th Pl. NE
Kirkland, WA 98033

Anthony J. Moore
4418 Corliss Avenue North, Apt. A
Seattle, WA 98112

Christopher L. Anderson
18612 SE 45th Street
Issaquah, WA 98027

Michael Pizzo
528 W Lake Sammamish PKWY SE.
Bellevue, WA 98008

Robert Allan Brigham II
21226 NE 132nd Ct.
Woodinville, WA 98072

Assignee:

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

AGREEMENT

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
COPIA FOTOCOPIA CONCUERDA CON UN DOCUMENTO IDENTICO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Attorney Docket No. MS1-1781US
MS Docket No. 305796.01
Serial No. 10/692,520

WHEREAS, ASSIGNOR(S) (listed above) are inventor(s) of an invention entitled "Design of Application Programming Interfaces (APIs)," as described and claimed in the specification forming part of an application for United States letters patent referenced above;

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

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

NOTARIA 16 CIRCULO DE BOGOTA D.C. 27 AGO 2004 PRESENTACION PERSONAL

Page 2 of 3

TRADUCCION OFICIAL No. 2004AA12
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
REG. No. 323 DEL MIN. DE JUSTICIA

306

Attorney Docket No. MSI-1748US
MS Docket No. 305796.01
Serial No. 10/697,320

Date	Krzysztof J. Cwalina
Date	Bradley Moore Abrams
Date	Anthony J. Moore
Date 2/3/04	Christopher L. Anderson
Date	Michael Pizzo
Date	Robert Allan Brigham II

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
DESPACHO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE [illegible]

Attorney Docket No. MSI-1748 US
MS Docket No. 305796.01
Serial No. 10/692,320

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial Number 10/692,320
Filing Date ..... Oct 23, 2003
Inventorship ..... Cwalina et al.
Applicant ..... Microsoft Corporation
Attorney's Docket No. MSI-1748US
MSDocket No. 305796.01
Title: Design of Application Programming Interfaces (APIs)

PATENT ASSIGNMENT

PARTIES TO THE ASSIGNMENT

Assignor(s):

Krzysztof J. Cwalina
7583 Old Redmond Road. Apt. A204
Redmond. WA 98052

Bradley Moore Abrams.
7517 128th Pl NE
Kirkland, WA 98033

Anthony J. Moore
4418 Corliss Avenue North. Apt. A
Seattle, WA 98112

Christopher L. Anderson
18612 SE 45th Street
Issaquah, WA 98027

Michael Pizzo
528 W Lake Sammamish PKWY SE
Bellevue, WA 98008

Robert Allan Brigham II
21226 NE 192nd Ct.
Woodinville, WA 98072

Assignee:

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

AGREEMENT

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO 2004
ESTA FOTOCOPIA CONCUERDA CON EL DOCUMENTO
FIRMADO QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PIEDRAHITA

Attorney Docket No. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,330

WHEREAS, ASSIGNOR(S) (listed above) are inventor(s) of an invention entitled "Design of Application Programming Interfaces (APIs)," as described and claimed in the specification forming part of an application for United States letters patent referenced above;

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

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

NOTARIA 16 CIUDAD DE SAN PEDRO SULA 7 AGO. 2004

TRADUCCION OFICIAL No. 2004-492
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
REG. No. 323 DEL MIN. DE JUSTICIA

Attorney Docket MSI-1748US
MS Docket No. 305796.01
Serial No. 10/692,320

Date	Krzysztof J. Cwalina
Date	Bradley Moore Abrams
Date	Anthony J. Moore
Date	Christopher L. Anderson
Date 1/22/2004	Michael Plizga
Date	Robert Allen Brigham II

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
FOTOCOPIA COINCIDE EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 20411
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

310

Attorney Docl No. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,320

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial Number 10/692,320
Filing Date ..... Oct 23, 2003
Inventorship ..... Cwalina et al.
Applicant ..... Microsoft Corporation
Attorney's Docket No. MS1-1748US
MSDocket No. 305796.01
Title: Design of Application Programming Interfaces (APIs)

PATENT ASSIGNMENT

PARTIES TO THE ASSIGNMENT

Assignor(s):

Krzysztof J. Cwalina
7583 Old Redmond Road, Apt. A204
Redmond, WA 98052

Bradley Moore Abrams
7517 128th Pl. NE
Kirkland, WA 98033

Anthony J. Moore
4418 Corliss Avenue North, Apt. A
Seattle, WA 98112

Christopher L. Anderson
18612 SE 45th Street
Issaquah, WA 98027

Michael Pizzo
528 W Lake Sammamish PKWY SE
Bellevue, WA 98008

Robert Allan Brigham II
21226 NE 132nd Ct.
Woodinville, WA 98072

Assignee:

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

AGREEMENT

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
21 AGO. 2004
ESTA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/192
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL

Attorney Doc. No. MSI-1748US
MS Docket No. 308796.01
Serial No. 10/692,320

WHEREAS, ASSIGNOR(S) (listed above) are inventor(s) of an invention entitled "Design of Application Programming Interfaces (APIs)," as described and claimed in the specification forming part of an application for United States letters patent referenced above;

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

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

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
17 AGO. 2004
LA PRESENTE COPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 333 DEL MIN. DE JUSTICIA

Attorney Doc. No. MS1-1748US
MS Docket No. 305796.01
Serial No. 10/692,320

Date	Krzysztof J. Cwalina
Date	Bradley Moore Abrams
Date	Anthony J. Moore
Date	Christopher L. Anderson
Date	Michael Pizzo
1/29/2004 Date	Robert Allan Brigham II

NOTARIA 16
SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004

TRADUCCION OFICIAL No. 2004-1492
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Traducción Oficial No. 2004/492 de un documento escrito en inglés el cual para su identificación se sella con el sello del traductor al tiempo con la presente. Esta traducción ha sido preparada por Hilda Aguilera de Piedrahita, traductora oficial debidamente reconocida por el Ministerio de Relaciones Exteriores y autorizada por el Ministerio de Justicia y del Derecho por Resolución No. 323 de 1980.

A12002225

ESTADOS UNIDOS DE AMERICA

A TODOS A QUIENES EL PRESENTE LLEGARE A INTERESAR

DEPARTAMENTO DE COMERCIO DE LOS ESTADOS UNIDOS

Oficina de Marcas y Patentes

Agosto 03,2004

EL PRESENTE TIENE COMO FIN CERTIFICAR QUE EL ANEXO ES UNA COPIA FIEL DE LOS ARCHIVOS DE ESTA OFICINA DE UN DOCUMENTO REGISTRADO EL 19 DE MARZO DE 2004.

Por autoridad del COMISIONADO DE MARCAS Y PATENTES

(Firmado: Ilegible)

N. WILLIAMS

Funcionario que certifica

Sello de Oblea y en Relieve.

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
ESTA FOTOCOPIA COINCIDE EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

CARATULA DE FORMATO DE REGISTRO

UNICAMENTE PATENTES

Formato PTO-1595

(Rev 03/01)

OMB No. 0651-0027(exp.5/31/2002)

Departamento de Comercio de E.U.

Oficina de Marcas y Patentes de los E.U.

Tabulados

AL: Honorable Comisionado de Patentes y Marcas: Por favor registre los anexos documento(s) o copia(s) originales.

1. Nombre(s) de la(s) parte(s) que entregan:

   Krzysztof J. Chalina, Bradley Moore Abrams, Anthony J. Moore, Christopher L. Anderson

   Nombres Adicionales de la(s) parte(s) que entregan anexas? _X__Si __No

2. Nombre(s) y Dirección(es) de la(s) parte(s) que reciben:

   Nombre:Microsoft Corporation

   Dirección Interna:

   Calle:One Microsoft Way

   Ciudad:Redmond Estado: WA Código Postal: 98052

   Nombre(s) y Dirección(es) adicional(es) anexas? ___Si _X_No

3. Naturaleza de la Entrega:

   __X__Cesión _____Fusión

   _____Gravamen _____Cambio de Nombre

NOTARIA 16 DE SANTAFE DE BOGOTA D.C. DR. BARRIOS GALVIS 27 AGO. 2004 COPIA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA

_____Otro

Fecha de Suscripción: 01/29/04; 02/03/04; 01/28/04

4. Numero(s) de Solicitud o Número(s) de Patente:

   Si este documento se está presentando con una nueva solicitud, la fecha de suscripción de la nueva solicitud es:

   A. Solicitud de Patente No.(s)

   B. Patente No.(s)

   Números Adicionales anexos? ___Si _X_No

5. Nombre y Dirección de la parte a quien la correspondencia relacionada con este asunto deberá ser enviada:

   Nombre:Keith W. Saunders, Reg No. 41.462

   Dirección Lee & Hayes, PLLC

   Dirección Calle 421 West Riverside Avenue Suite 500

   Cuidad: Spokane

   Estado WA

   Zip: 99201

6. Numero total de Patentes y solicitudes involucradas: 1

7. Total Derechos (37 CFR 3.41).....$ 40.00

   _____Adjunto __X__Autorizado para cargarse a cuenta de depósito

8. Número de Cuenta de Depósito: 12-00769

(Adjunte copia duplicado de esta página si paga mediante Cuenta de Depósito)

NO USE ESTE ESPACIO

9. Declaración y Firma

NOTARIA 16
CONSULADO SANTA FE DE BOGOTA D.C.
ALBERTO BARRIOS GALVIS
27 AGO. 2004
DILIGENCIA DE PRESENTACION
COPIA FOTOSTATICA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004/A012
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Según mi leal saber y entender, la anterior información es verdadera y correcta y cualquier copia anexa es una copia verdadera del documento original.

Keith W. Saunders, Reg No. 41.462 (Firmado) 3/19/2004

Nombre de la persona firmante Firma Fecha

Número Total de Páginas incluyendo esta carátula 14

Envíe documentos para ser registrados con la información requerida en la carátula a:

Caja de Cesiones

Comisionado Oficina de Marcas y Patentes de EUA, Correo de Cesiones

Alexandria, Virginia 22313-1450

7000733326 PATENTE ROLLO 00114447 MARCO:0269

CARATULA DE FORMATO DE REGISTRO

Página 2

Solicitud No. 10/692,320

1.Nombre(s) de la(s) parte(s) que entregan:

Michael Pizzo

Robert Allan Brigham II

PATENTE ROLLO 00114447 MARCO:0269

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
ESTA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

EN LA OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Número de Serie.....10/692.320

Fecha de Presentación.....Oct. 23,2003

Inventor.....Cwalina et. Al.

Solicitante.....Microsoft Corporation

Expediente Abogado..... MS1-1748US

Expediente MS No.....305796.01

Titulo..: "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)"

CESION DE PATENTE

PARTES DE LA CESION

Cedente(s):

Krzysztof J. Cwalina

7583 Old Redmond Road, Apt. A204

Redmond, Washington 98052

Bradley Moore Abrams

7517 128th PL NE

Kirkland, Washington 98033

Anthony J. Moore

4418 Corliss Avenue North, Apt A

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
21 AGO. 2004
ESTA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 20041472
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 320 DEL MIN. DE JUSTICIA

Seattle, Washington 98112

Christopher L. Anderson

18612 SE 45th Street

Issaquah, Washington 98027

Michael Pizzo

528 W Lake Sammamish PKWY SE

Bellevue, Washington 98008

Robert Allan Brigham II

21226 NE 132nd Ct.

Woodinville, WA 98072

Cesionario:

Microsoft Corporation

Sociedad en el Estado de Washington

One Microsoft Way

Redmond, WA 98052

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
21 AGO. 2004
LA FOTOCOPIA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

ACUERDO

 PATENTE ROLLO 014447 MARCO 0271

TRADUCCION OFICIAL No. 2004/892
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

POR CUANTO, LOS CEDENTES (arriba enumerados) son inventores de una invención titulada "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)", como se describe y reivindica en la especificación que forma parte de la solicitud para Patente de los Estados Unidos arriba en referencia;

POR CUANTO, Microsoft Corporation (en adelante referida como CESIONARIO), una sociedad del Estado de Washington teniendo su domicilio de negocios en One Microsoft Way, Redmond, WA 98052, tiene deseo de adquirir el entero derecho, titulo e interés en y para la invención y en y para los certificados de patente que puedan ser otorgados a la misma en los Estados Unidos y en cualesquier y todos los países extranjeros;

AHORA POR CONSIGUIENTE, a cambio de una consideración buena y valuable, recibo de la cual por el presente se reconoce, los CEDENTES por el presente venden, ceden y transfieren al CESIONARIO, El entero derecho, titulo e interés en y para la mencionada invención, la mencionada solicitud y cualesquiera y todas las certificados de patente que pudieren ser otorgadas para tal invención en los Estados Unidos y en sus posesiones territoriales y en cualesquiera y todos los países extranjeros, y en cualesquiera y todas las divisiones, reexpediciones y

NOTARIA 16 OTTO D. ... 21 AGO. 2006

TRADUCCION OFICIAL No. 20041
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
REG. No. 323 DEL MIN. DE JUSTICIA

continuaciones de la misma, incluyendo el derecho de presentar solicitudes extranjeras directamente a nombre del CESIONARIO y de reivindicar derechos de prioridad derivados de la mencionada solicitud en los Estados Unidos para la cual dichas solicitudes extranjeras tengan derecho en virtud de convenciones internacionales, tratados o de cualquier otra manera, dicha invención, solicitud y todas los certificados de patente para dicha invención serán mantenidos y disfrutados por el CESIONARIO y sus sucesores y cesionarios para su uso y beneficio y de sus sucesores y cesionarios como si el mismo hubiese sido completamente y enteramente mantenido y disfrutado por los CEDENTES si esta cesión, transferencia y venta no hubiese sido efectuada. Los CEDENTES por el presente autorizan y solicitan al Comisionado de Patentes y Marcas que expida todos los certificados de patente para dicha invención al CESIONARIO. Los CEDENTES acuerdan suscribir todos los instrumentos y documentos necesarios para la presentación y prosecución de las solicitudes para los Estados Unidos y certificados de patentes extranjeras para dicha invención, para litigios respecto de dichos certificados de patente, o con el fin de proteger el título de dicha invención o certificados de patente de la misma.

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

01-29-04
Fecha

(Firmado)
Krzysztof J. Cwalina

NOTARIA 16
DEL CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
21 AGO. 2004
ESTA FOTOCOPIA COINCIDE EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2334-12
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

01-29-04 Fecha	(Firmado) Bradley Moore Abrams
Enero 29,2004 Fecha	(Firmado) Anthony J. Moore
Fecha	(En blanco) Christopher L. Anderson
F cha	(En blanco) Michael Pizzo
Fecha	(En blanco) Robert Allan Brigham II

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

EN LA OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Número de Serie.....10/692.320

Fecha de Presentación.....Oct. 23,2003

Inventor.....Cwalina et. Al.

Solicitante.....Microsoft Corporation

Expediente Abogado..... MS1-1748US

Expediente MS No.....305796.01

Titulo..: "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)"

NOTARIA 16 SANTAFE DE BOGOTA D.C. 1 AGO. 2004 COPIA FOTOSTATICA CONCUERDA FIELMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2644-02
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 333 DEL MIN. DE JUSTICIA

322.

CESION DE PATENTE

PARTES DE LA CESION

Cedente(s):

Krzysztof J. Cwalina

7583 Old Redmond Road, Apt. A204

R dmond, Washington 98052

Bradley Moore Abrams

7517 128th PL NE

Kirkland, Washington 98033

Anthony J. Moore

4418 Corliss Avenue North, Apt A

Seattle, Washington 98112

Christopher L. Anderson

18612 SE 45th Street

Issaquah, Washington 98027

Michael Pizzo

528 W Lake Sammamish PKWY SE

Bellevue, Washington 98008

TRADUCCION OFICIAL No. 2604492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA

323

Robert Allan Brigham II

21226 NE 132nd Ct.

Woodinville, WA 98072

Cesionario:

Microsoft Corporation

Sociedad en el Estado de Washington

One Microsoft Way

Redmond, WA 98052

ACUERDO

Página 1 de 3 PATENTE ROLLO 014447 MARCO:0274

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

POR CUANTO, LOS CEDENTES (arriba enumerados) son inventores de una invención titulada "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)", como se describe y reivindica en la especificación que forma parte de la solicitud para Patente de los Estados Unidos arriba en referencia;

NOTARIA 27 AGO. 2004

TRADUCCION OFICIAL No. 2004/472
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

POR CUANTO, Microsoft Corporation (en adelante referida como CESIONARIO), una sociedad del Estado de Washington teniendo su domicilio de negocios en One Microsoft Way, Redmond, WA 98052, tiene deseo de adquirir el entero derecho, titulo e interés en y para la invención y en y para los certificados de patente que puedan ser otorgados a la misma en los Estados Unidos y en cualesquier y todos los países extranjeros;

AHORA POR CONSIGUIENTE, a cambio de una consideración buena y valuable, recibo de la cual por el presente se reconoce, los CEDENTES por el presente venden, ceden y transfieren al CESIONARIO, El entero derecho, titulo e interés en y para la mencionada invención, la mencionada solicitud y cualesquiera y todas los certificados de patente que pudieren ser otorgadas para tal invención en los Estados Unidos y en sus posesiones territoriales y en cualesquiera y todos los países extranjeros, y en cualesquiera y todas las divisiones, reexpediciones y continuaciones de la misma, incluyendo el derecho de presentar solicitudes extranjeras directamente a nombre del CESIONARIO y de reivindicar derechos de prioridad derivados de la mencionada solicitud en los Estados Unidos para la cual dichas solicitudes extranjeras tengan derecho en virtud de convenciones internacionales, tratados o de cualquier otra manera, dicha invención, solicitud y todas los certificados de patente para dicha invención serán mantenidos y disfrutados por el CESIONARIO y sus sucesores y cesionarios para su beneficio y de sus sucesores y cesionarios como si el mismo hubiese sido completamente y enteramente mantenido y disfrutado por los CEDENTES si esta cesión, transferencia y venta no hubiese sido efectuada. Los CEDENTES por el

NOTARIA 16 ... 27 AGO. 2004

TRADUCCION OFICIAL No. 2004/492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA

presente autorizan y solicitan al Comisionado de Patentes y Marcas que expida todos los certificados de patente para dicha invención al CESIONARIO. Los CEDENTES acuerdan suscribir todos los instrumentos y documentos necesarios para la presentación y prosecución de las solicitudes para los Estados Unidos y certificados de patentes extranjeras para dicha invención, para litigios respecto de dichos certificados de patente, o con el fin de proteger el titulo de dicha invención o certificados de patente de la misma.

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

Fecha	(En blanco) Krzysztof J. Cwalina
Fecha	(En blanco) Bradley Moore Abrams
Fecha	(En blanco) Anthony J. Moore
2/3/04 Fecha	(Firmado) Christopher L. Anderson
Fecha	(En blanco) Michael Pizzo
Fecha	(En blanco) Robert Allan Brighton

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
COPIA FOTOSTATICA QUE COINCIDE EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004-AA22
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

EN LA OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Número de Serie.....10/692.320

Fecha de Presentación.....Oct. 23,2003

Inventor.....Cwalina et. Al.

Solicitante.....Microsoft Corporation

Expediente Abogado..... MS1-1748US

Expediente MS No.....305796.01

Titulo..: "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)"

CESION DE PATENTE

PARTES DE LA CESION

Cedente(s):

Krzysztof J. Cwalina

7583 Old Redmond Road, Apt. A204

Redmond, Washington 98052

Bradley Moore Abrams

7517 128th PL NE

Kirkland, Washington 98033

TRADUCCION OFICIAL No. 2004492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Anthony J. Moore

4418 Corliss Avenue North, Apt A

Seattle, Washington 98112

Christopher L. Anderson

18612 SE 45th Street

Issaquah, Washington 98027

Michael Pizzo

528 W Lake Sammamish PKWY SE

Bellevue, Washington 98008

Robert Allan Brigham II

21226 NE 132nd Ct.

Woodinville, WA 98072

Cesionario:

Microsoft Corporation

Sociedad en el Estado de Washington

One Microsoft Way

Redmond, WA 98052

ACUERDO

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
ESTA FOTOCOPIA CORRESPONDE EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 2004-1212
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES No. 320 DEL MIN. DE JUSTICIA

528

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

POR CUANTO, LOS CEDENTES (arriba enumerados) son inventores de una invención titulada "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)", como se describe y reivindica en la especificación que forma parte de la solicitud para Patente de los Estados Unidos arriba en referencia;

POR CUANTO, Microsoft Corporation (en adelante referida como CESIONARIO), una sociedad del Estado de Washington teniendo su domicilio de negocios en One Microsoft Way, Redmond, WA 98052, tiene deseo de adquirir el entero derecho, titulo e interés en y para la invención y en y para los certificados de patente que puedan ser otorgados a la misma en los Estados Unidos y en cualesquier y todos los paises extranjeros;

AHORA POR CONSIGUIENTE, a cambio de una consideración buena y valuable, recibo de la cual por el presente se reconoce, los CEDENTES por el presente venden, ceden y transfieren al CESIONARIO, El entero derecho, título e interés en y para la mencionada invención, la mencionada solicitud y cualesquiera y todas los certificados de patente que pudieren ser otorgadas para tal invención en los

NOTARIA 15 D.C. BARRIOS UNIDOS BOGOTA D.C. 27 ABRIL 2004

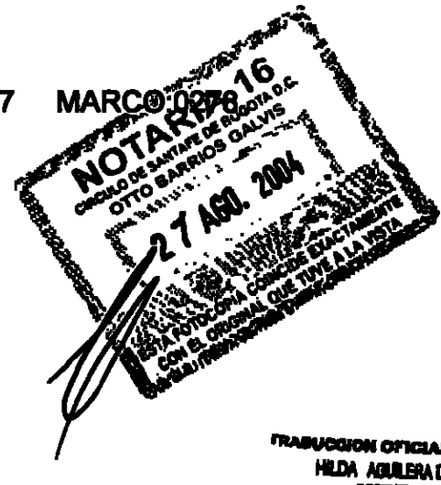
TRADUCCION OFICIAL No. 2004412
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 388 DEL MIN. DE JUSTICIA

Estados Unidos y en sus posesiones territoriales y en cualesquiera y todos los países extranjeros, y en cualesquiera y todas las divisiones, reexpediciones y continuaciones de la misma, incluyendo el derecho de presentar solicitudes extranjeras directamente a nombre del CESIONARIO y de reivindicar derechos de prioridad derivados de la mencionada solicitud en los Estados Unidos para la cual dichas solicitudes extranjeras tengan derecho en virtud de convenciones internacionales, tratados o de cualquier otra manera, dicha invención, solicitud y todas los certificados de patente para dicha invención serán mantenidos y disfrutados por el CESIONARIO y sus sucesores y cesionarios para su uso y beneficio y de sus sucesores y cesionarios como si el mismo hubiese sido completamente y enteramente mantenido y disfrutado por los CEDENTES si esta cesión, transferencia y venta no hubiese sido efectuada. Los CEDENTES por el presente autorizan y solicitan al Comisionado de Patentes y Marcas que expida todos los certificados de patente para dicha invención al CESIONARIO. Los CEDENTES acuerdan suscribir todos los instrumentos y documentos necesarios para la presentación y prosecución de las solicitudes para los Estados Unidos y certificados de patentes extranjeras para dicha invención, para litigios respecto de dichos certificados de patente, o con el fin de proteger el título de dicha invención o certificados de patente de la misma.

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320



TRADUCCION OFICIAL No. 2004-472
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA

330

Fecha	(En blanco) Krzysztof J. Cwalina
Fecha	(En blanco) Bradley Moore Abrams
Fecha	(En blanco) Anthony J. Moore
Fecha	(En blanco) Christopher L. Anderson
1/28/2004 Fecha	(Firmado) Michael Pizzo
Fecha	(En blanco) Robert Allan Brigham II

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

EN LA OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Número de Serie.....10/692.320

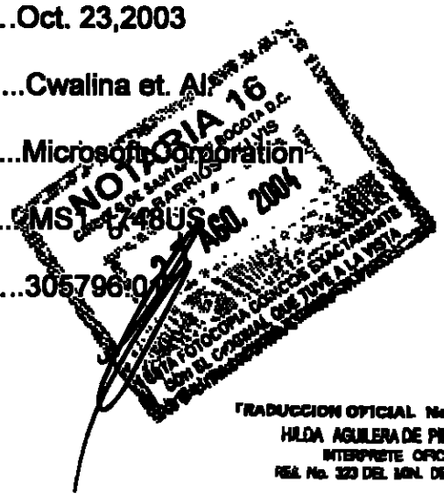
Fecha de Presentación.....Oct. 23,2003

Inventor.....Cwalina et. Al.

Solicitante.....Microsoft Corporation

Expediente Abogado.....MS1-1748US

Expediente MS No.....305796.01



TRADUCCION OFICIAL No. 2004-F12
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 320 DEL MIN. DE JUSTICIA

331

Titulo..: "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)"

CESION DE PATENTE

PARTES DE LA CESION

Cedente(s):

Krzysztof J. Cwalina

7583 Old Redmond Road, Apt. A204

Redmond, Washington 98052

Bradley Moore Abrams

7517 128th PL NE

Kirkland, Washington 98033

Anthony J. Moore

4418 Corliss Avenue North, Apt A

Seattle, Washington 98112

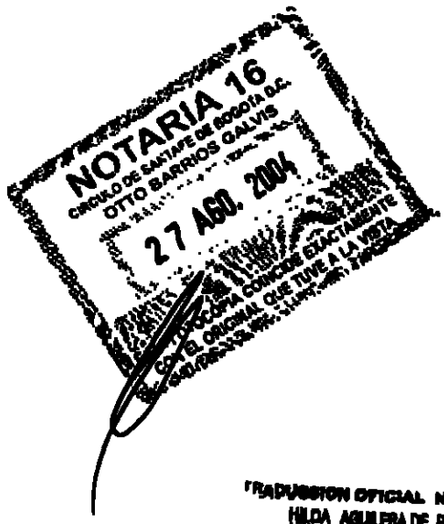
Christopher L. Anderson

18612 SE 45th Street

Issaquah, Washington 98027

Michael Pizzo

528 W Lake Sammamish PKWY SE



TRADUCCION OFICIAL No. 22492
HILDA AGUILERA DE PEDRANTA
INTERPRETE OFICIAL
RES. No. 382 DEL MIN. DE JUSTICIA

Bellevue, Washington 98008

Robert Allan Brigham II

21226 NE 132nd Ct.

Woodinville, WA 98072

Cesionario:

Microsoft Corporation

Sociedad en el Estado de Washington

One Microsoft Way

Redmond, WA 98052

ACUERDO

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

POR CUANTO, LOS CEDENTES (arriba enumerados) son inventores de una invención titulada "DISEÑO DE INTERFACES DE PROGRAMACION DE APLICACIONES (API)", como se describe y reivindica en la especificación que forma parte de la solicitud para Patente de los Estados Unidos arriba en referencia;

NOTARIA 16 OTTO SARRIOS GARCIA D.C. 23 AGO. 2004 COPIA FOTOSTATICA CONFORME CON EL ORIGINAL QUE TUVE A LA VISTA

TRADUCCION OFICIAL No. 21492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

333

POR CUANTO, Microsoft Corporation (en adelante referida como CESIONARIO), una sociedad del Estado de Washington teniendo su domicilio de negocios en One Microsoft Way, Redmond, WA 98052, tiene deseo de adquirir el entero derecho, titulo e interés en y para la invención y en y para los certificados de patente que puedan ser otorgados a la misma en los Estados Unidos y en cualesquier y todos los países extranjeros;

AHORA POR CONSIGUIENTE, a cambio de una consideración buena y valuable, recibo de la cual por el presente se reconoce, los CEDENTES por el presente venden, ceden y transfieren al CESIONARIO, El entero derecho, titulo e interés en y para la mencionada invención, la mencionada solicitud y cualesquiera y todas los certificados de patente que pudieren ser otorgadas para tal invención en los Estados Unidos y en sus posesiones territoriales y en cualesquiera y todos los países extranjeros, y en cualesquiera y todas las divisiones, reexpediciones y continuaciones de la misma, incluyendo el derecho de presentar solicitudes extranjeras directamente a nombre del CESIONARIO y de reivindicar derechos de prioridad derivados de la mencionada solicitud en los Estados Unidos para la cual dichas solicitudes extranjeras tengan derecho en virtud de convenciones internacionales, tratados o de cualquier otra manera, dicha invención, solicitud y todas los certificados de patente para dicha invención serán mantenidos y disfrutados por el CESIONARIO y sus sucesores y cesionarios para su uso beneficio y de sus sucesores y cesionarios como si el mismo hubiese sido completamente y enteramente mantenido y disfrutado por los CEDENTES si esta

NOTARIA 1ª 27 AGO 2004

TRADUCCION OFICIAL No. 2004-1612
HILDA AGUILERA DE PEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

33

cesión, transferencia y venta no hubiese sido efectuada. Los CEDENTES por el presente autorizan y solicitan al Comisionado de Patentes y Marcas que expida todos los certificados de patente para dicha invención al CESIONARIO. Los CEDENTES acuerdan suscribir todos los instrumentos y documentos necesarios para la presentación y prosecución de las solicitudes para los Estados Unidos y certificados de patentes extranjeras para dicha invención, para litigios respecto de dichos certificados de patente, o con el fin de proteger el título de dicha invención o certificados de patente de la misma.

Expediente de Abogado MS1-1748US

Expediente MS No. 305796.01

Serie No. 10/692.320

Fecha	(En blanco) Krzysztof J. Cwalina
Fecha	(En blanco) Bradley Moore Abrams
Fecha	(En blanco) Anthony J. Moore
Fecha	(En blanco) Christopher L. Anderson
Fecha	(En blanco) Michael Pizzo
1/29/2004 Fecha	(Firmado) Robert Allan Brigham II

REGISTRADO:03/19/2004

NOTARÍA 16
CÍRCULO DE SANTAFE DE BOGOTÁ D.C.
OTTO BARRIOS GALVIS
27 ABR. 2004
TRADUCCION OFICIAL No. 2004/1492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 223 DEL MIN. DE JUSTICIA
335

335

APOSTILLA

(Convención de la Haya de Octubre 5 de 1961)

1. País: Estados Unidos de América

   Este documento público

2. Ha sido firmado por N. Williams

3. Actuando en calidad de Funcionario que certifica

4. Lleva el sello/estampilla de la oficina de Patentes y Marcas de los EEUU

CERTIFICADO

5. En Washington, D.C.

6. El cinco de Agosto, 2004.

7. Por el Funcionario de Autenticación Asistente, Departamento de Estado de los Estados Unidos

8. No. 04025116-1

9. Sello (Sello Redondo)

10. Firma

    (Firmado)

    Fernesia T. Crawford

NOTARIA 16
CIRCULO DE SANTAFE DE BOGOTA D.C.
OTTO BARRIOS GALVIS
27 AGO. 2004
COPIA FOTOSTATICA CONCUERDA EXACTAMENTE CON EL ORIGINAL QUE TUVE A LA VISTA

ES TRADUCCIÓN FIEL Y COMPLETA de la cual se deja copia en los archivos de esta oficina para su confrontación y a la cual me remito.

Bogotá, D.C., Agosto 26 de 2004.

TRADUCCION OFICIAL No. 2004-1492
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

330

DE SANIDAD
LAS EXCEPCIONES INDICADAS

SANTAFE DE BOGOTA D.C.

336

SUPERINDUSTRIA Y COMERCIO

Radicacion 04004792 00000000 Folios: 336

Fecha (AMD) 2004-08-30 18:00:43

Tramite : 002 PATENTES D 1 REGISTRO/ 411 PRESENTA

Procedencia: PARA DIVISION DE NUEVAS CREACIONES

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		( )	( )
Representacion gráfica del esquema de trazado		( )	( )
Comprobante de pago de las tasas establecidas		( )	( )
COMPLETA ( )	INCOMPLETA	( )	( )

Firma Responsable

Fecha 30 Ago-04

337

PROTOCOLO

Expediente 04-84792

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
	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

1er. EXAMEN DE FORMA PATENTE DE INVENCION

Folio 339 338

Radicación No 04-84792

Concepto No.729 Clasificación Internacional de Patentes G 06 F 13/42

Practicado el examen de acuerdo a lo establecido en el Art. 38 de la Decisión 486 se determinó que esta solicitud de **Patente de Invención:**

REVISION TECNICA

SI [✓] NO [ ] Contiene un resumen de la invención (Art.26-e).

SI [✓] NO [ ] Contiene el título o nombre de la invención (Art.27-d).

SI [✓] NO [ ] Contiene la descripción de la invención (Art.26-b).

SI [✓] NO [ ] NO APLICA [ ] Contiene los dibujos necesarios para comprender la invención (Art.26-d)

SI [✓] NO [ ] Contiene una o más reivindicaciones. (Art.26-c).

SI [ ] NO [ ] NO APLICA [✓] Han sido desarrollados los productos y/o procedimientos a partir de recursos genéticos o de sus productos derivados de los que cualquiera de los Paises Miembros es país de origen. (Art. 26- h).

SI [ ] NO [ ] NO APLICA [✓] Los productos y/o procedimientos han sido obtenidos o desarrollados a partir de los conocimientos tradicionales de las comunidades indígenas, afroamericanas, o locales de los países miembros, de acuerdo a lo establecido en la Decisión 391 y sus modificaciones y reglamentaciones vigentes.(Art.26-i).

SI [–] NO [–] La invención se refiere a un producto relativo a un material biológico. (Art.26-J).

SI [✓] NO [ ] NO APLICA [ ] La prioridad coincide con la materia reivindicada.

SI [✓] NO [ ] CUMPLE LOS REQUISITOS TÉCNICOS

REVISION JURIDICA

SI [✓] NO [ ] NO APLICA [ ] Presentó el documento en que consta la cesión del derecho a la patente otorgado por el inventor al solicitante o a su causante (art. 26-k,Dec 486/2000).

SI [✓] NO [ ] NO APLICA [ ] Presentó el poder otorgado al abogado, con el lleno de los requisitos jurídicos exigidos.

SI [✓] NO [ ] NO APLICA [ ] Acredito la existencia y representación de la sociedad solicitante.

SI [✓] NO [ ] NO APLICA [ ] Presento la copia certificada de la prioridad invocada en los términos establecidos en los artículos 10 y 11, Dec. 486.

SI [✓] NO [ ] CUMPLE LOS REQUISITOS JURÍDICOS

[*] OBSERVACIONES Y OTROS:

EXAMINADOR 202027
Bogotá D. C., 11-10-2004

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

2020
Bogotá DC

Doctor(a)
DARIO CARDENAS NAVAS
Apoderado(a)

REFERENCIA:	Radicación No.	04-84792
	Trámite	002
	Evento	001
	Actuación	416
	Oficio No.	1571
	Folios	001

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

De acuerdo con lo previsto por el artículo 40 de la Decisión 486/00, la publicación en este caso se hará a partir del **día 30 de febrero de 2006.**

Cúmplase
Dado en Bogotá DC, a los 15 OCT. 2004

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

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