

156

SUPERINDUSTRIA Y COMERCIO

Radicacion : 04039312 00000002

Folios: 231

Fecha (AMD): 2004-07-06 13:57:38

Tramite : 011 PCT-PATENT 378 FASE NACI 444 RESPUESTA

Dependencia: 2020 DIVISION DE NUEVAS CREACIONES

Señores

Superintendencia de Industria y Comercio

División de Nuevas Creaciones

E. S. D.

Ref: Expediente No. 04-039.312

Solicitud de Patente de Invención PCT denominada "Sistema para Hospedar Objetos Gráficos de Diseño / Presentación"

Solicitante: Microsoft Corporation

Luz Helena Adarve Gómez, identificada como aparece al pie de mi firma y obrando en mi condición de apoderada de **Microsoft Corporation**, sociedad solicitante de la patente mencionada en la referencia, a continuación me permito dar respuesta formal al Oficio No. 05233 notificado por fijación en lista el 21 de Mayo de 2.004:

1. Según lo solicitado, adjunto a este escrito una copia auténtica del documento certificado de traspaso de la patente en referencia de los inventores a favor de la sociedad solicitante, el cual se encuentra debidamente legalizado y acompañado de su traducción oficial al español avalada por el Ministerio de Relaciones Exteriores.
2. Igualmente me permito indicar que el Número de Protocolo del Poder conferido por la sociedad **Microsoft Corporation** a favor de la suscrita, entre otros apoderados, sí fue citado de manera correcta en la solicitud de registro correspondiente, como el No. **16.069**. Para su rápida referencia adjunto una copia del mencionado Protocolo. Debe tenerse en cuenta que en el mencionado Poder aparece como designada además del Dr. Darío Cárdenas Navas la suscrita **Luz Helena Adarve Gómez**, entre otros apoderados.

De otro lado me permito adjuntar nuevamente el texto completo de la solicitud de patente en referencia junto con su traducción al español, por cuanto los documentos presentados anteriormente contenían algunos errores.

Solicito en consecuencia al Despacho se sirva reconocerme personería para continuar actuando dentro de este proceso.

Atentamente,



Luz Helena Adarve Gómez

C.C. 41.575.435 de Bogotá

T. P. 14.884 del Consejo Superior de la Judicatura



SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO

2010/

Santafé de Bogotá D.C.

SUPERINDUSTRIA Y COMERCIO
 Radicación : 95034320 000001 Folios: 1
 Fecha (AND): 1995-09-06 10:36:48
 Trámite : 015 PROTOCOLOS 66 REGISTRO 034 RESPUESTA
 Dependencia: 2010 DIVISION DE SIGNOS DISTINTIVOS

Doctor (a)
 CARDENAS NAVAS DARIO ✓
 CRA 7 N 71-52 TORRE B P. 9
 BOGOTA

REF: No. RADICACION 95 34920 0000 00
 Trámite : 015 PROTOCOLOS
 Actuación : 034 RESPUESTA

PODERDANTE: MICROSOFT CORPORATION ✓

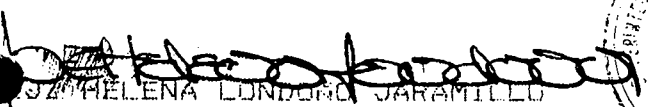
FECHA OTORGAMIENTO: 04/08/1995 ✓

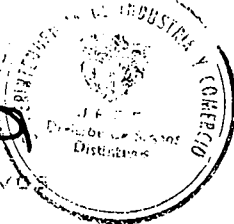
NUMERO PROTOCOLO : 16069 ✓

Apreciado(a) doctor(a) :

En atención a su solicitud de protocolización del documento de poder identificado en la referencia, me permito informarle que éste ha sido incorporado al protocolo de la entidad bajo el numero arriba indicado

Hasta una próxima oportunidad,


 HELENA LONDOÑO JARAMILLO
 Jefe.- División Signos Distintivos



BLETT
 SK

157

17



CONSULADO GENERAL DE COLOMBIA

595 Market Street

Sulte 2130

San Francisco, California MAYO. 30 / 1995 No. 969 4725

El suscrito Consul General de Colombia CERTIFICA que el Señor

RALPH MUNRO

quien autoriza el

presente documento, ejercia legalmente, en la fecha alli expresada,

las funciones de SECRETARIO DE ESTADO -

ESTADO DE WASHINGTON. y que la firma y sello que

en el documento aparecen como suyos son los que usa y acostumbra

en sus actos oficiales. El Consulado no asume responsabilidad

alguna por el contenido del documento.



[Signature]
MIGUEL S. MAZA
CONSUL GENERAL DE COLOMBIA

PAGO IMPUESTO DE TIMBRE
DERECHOS CONSULARES

JEFE DE LEGALIZACIONES
SANTAFE DE BOGOTA D.C.
1995 JUN 12 A 11:05
RESPONSABILIDAD DEL TEXTO
AD SE ASOM

MINISTERIO DE RELACIONES
EXTERIORES
185293
RECEBIDO EN EL
DOCUMENTO DESEMPENA
LAS FUNCIONES INDICADAS

159

STATE of WASHINGTON



SECRETARY of STATE

I, **RALPH MUNRO**, Secretary of State of the State of Washington and custodian of its seal, hereby certify that, according to the records on file in my office, **CAROLYN M. SCHUSSLER** is a Notary Public in and for the state of Washington.

GENERAL DE

DATE: May 22, 1995



Given under my hand and the seal of the
State of Washington at Olympia, the State
Capital

Ralph Munro, Secretary of State

Tral off. #4697/95
LINES DE HODSON
Indemnity Bond
Resolutions No. 2196
by Magistrate

POWER OF ATTORNEY

in the city of Redmond, Washington
on the 12 day of the month of May, one thousand nine
hundred and ninety five (1995), before
me, Carolyn M. Schusser Notary Public,
appeared David Curtis of age, and domiciled
at Seattle, Washington, and
stated:

FIRST: That he/she acts in his/her capacity of Assistant Secretary of
MICROSOFT CORPORATION a company legally constituted and in
existence in accordance with the laws of the State of Washington, United
States of America, and domiciled at One Microsoft Way, Redmond,
Washington 98052-6399, United States of America as evidenced in the
By Laws, documents which I have examined.

SECOND: That in such capacity he/she grants hereby special, ample
and sufficient Power of Attorney to DARIO CARDENAS-NAVAS and/or
EDUARDO CARDENAS-CABALLERO and/or BERNARDO P.
CARDENAS-MARTINEZ y/o JAMES VALDIRI-REYES and/or LUZ
HELENA ADARVE-GOMEZ, registered lawyers of the firm CARDENAS
& CARDENAS-LAWYERS, domiciled at Carrera 7 No. 71-52, Torre B,
Piso 9 Bogotá, D.C., Colombia, to carry before the corresponding
Colombian national Offices and authorities the proceedings to obtain
trademarks, patents, utility models, industrial drawings and designs,
deposits of commercial names and ensigns, as well as processing of
renewals, assignments, change of name or domicile of the owner,
waivers and licenses of the same for which purpose they are empowered
to take before said authorities all necessary actions for the indicated
purposes; to file applications, statements, claims, formulate descriptions,
amendments, oppositions and appeals; pay all taxes and installments
and make any other payments determined by law; to receive every
document and values issuing the respective discharge; to fulfill any other
requirements and take, in general, every measure that they shall deem
opportune for the defense and protection of the Industrial Property of the
company in the event of infringements, oppositions, proceedings for
cancellation or annulment, imposition of fines, passing the antecedents

Handwritten signature
J. P. HOBSON
Notary
No. 2106



161

to the courts; they are empowered to appear as plaintiff or defendant before the competent judges, being entitled to transact, submit to arbitration, waive, receive, resign, appeal and all other legal powers which may be necessary, and it is hereby declared ever since now valid and good whatever such attorneys do for its benefit, granting them also power to substitute this proxy and if necessary to revoke such substitutions.

THIRD: That for purposes of articles 597 and 543 of the Colombian code of Commerce the attorneys are empowered to receive service of notice and designate judicial and extrajudicial attorneys.

SIGNED AT Redmond, Washington, United States of America

ON May 12 1995

BY

David Curtis, Assistant Secretary

Carolyn M. Schissel
NOTARY PUBLIC

(To be legalized up to the Colombian Consul

102. 4/22/95
NOTARY PUBLIC
Carolyn M. Schissel
102. 4/22/95



The undersigned Notary Public certifies that Mr. David Curtis (nationality), of legal age, domiciled at Seattle, Washington (civil status) appeared before me and signed the preceding document; that I know him personally and I am certain that he has legal capacity to grant it; that he is Assistant Secretary

of MICROSOFT CORPORATION and is consequently authorized to grant such document; that the named company exists and is organized under the laws of the state of Washington in conformity with the laws of the U.S.A. and is domiciled in Redmond, Washington and that the act for which this document is granted is included among those which constitute the corporate object of the named company.

I am certain of all these facts because I have examined the respective documents consisting of the Bylaws, all of which I hereby attest.

Issued and signed in Redmond, Washington

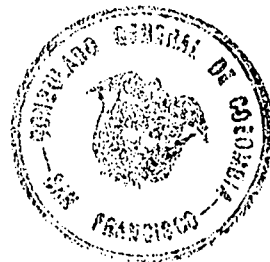
Dated May 12, 1995

Carolyn M. Schussler

Legalization by Colombian Consul is required. The Colombian Consul must certify that the company exists and practices its corporate purpose in accordance with the laws of the respective country and that the grantor is its representative

ECC/las

172 4424
JANE DE HODSON
Notary Public
San Francisco, CA 94104



TRADUCCION OFICIAL N°4697/95 de Poder escrito en idioma inglés, el cual para su identificación se sellan con el sello del traductor a tiempo con la presente.

=====

P O D E R

En la ciudad de Redmond, Washington, el día 12 de mayo de mil novecientos noventa y cinco (1995), ante mí, Carolyn M. Schussler, Notario Público, compareció David Curtis, mayor de edad y com- miciliado en Seattle, Washington, y manifestó:

PRIMERO: Que actúa en su calidad de Subsecretario de MICRO-
SOFT CORPORATION, compañía legalmente constituida y existen-
te de acuerdo con las leyes del Estado de Washington, Esta-
dos Unidos de América, y domiciliada en One Microsoft Way,
Redmond, Washington 98052-6399, Estados Unidos de América,
según consta en los Estatutos, documentos que he examinado.

SEGUNDO: Que en tal calidad, por el presente otorga Poder
especial, amplio y suficiente a DARIO CARDENAS NAVAS y/o E-
DUARDO CARDENAS CABALLERO y/o BERNARDO P. CARDENAS y/o JAMES
VALDIRI REYES y/o LUZ HELENA ADARVE GOMEZ, abogados titula-
dos de la firma CARDENAS & CARDENAS - ABOGADOS, domiciliada
en la Carrera 7 N° 71-52, Torre B, Piso 9, Bogotá, D.C., Co-
lombia para que adelanten ante las oficinas y autoridades na-
cionales colombianas correspondientes los trámites para ob-
tener marcas de fábrica, patentes, modelos de utilidad, di-
bujos y diseños industriales, depósitos de nombres comercia-
les y emblemas, así como el trámite de renovaciones, cesio-
nes, cambio de nombre o domicilio del propietario, desisti-
mientos y licencias de los mismos, a cuyo propósito están fa-
cultados para tomar ante dichas autoridades todos los pasos
necesarios para los propósitos indicados; presentar solici-
tudes, declaraciones, reclamos, formular descripciones, en-
miendas, oposiciones y apelaciones; pagar todos los impuestos

INCE DE HODSON
Traductor Oficial
Registrado No. 2196
de Abojusticia

y cuotas y hacer cualesquiera otros pagos determinados por ley; recibir todos los documentos y valores expidiendo el descargo respectivo; llenar cualesquiera otros requisitos y tomar, en general, todas las medidas que consideren oportunas para la defensa y protección de la propiedad industrial de la compañía en caso de infracciones, oposiciones, trámites para cancelación o anulación, imposición de multas, pasando los antecedentes a los tribunales. Están facultados para comparecer como demandante o demandado ante los jueces competentes, teniendo derecho a transar, someter a arbitramento, renunciar, recibir, desistir, apelar y todos los demás poderes legales que sean necesarios, y desde ahora se declara válido y bueno cuanto dichos apoderados hagan en beneficio de la compañía, otorgándoles también poder para sustituir este mandato y, si es necesario, revocar tales sustituciones.

TERCERO: Para los fines de los artículos 597 y 543 del Código de Comercio de Colombia, los apoderados están facultados para recibir notificaciones y designar agentes judiciales y extrajudiciales.

Firmado en Redmond, Washington, Estados Unidos de América el 12 de mayo de 1995

Por (fdo) David Curtis, Subsecretario

(fdo) Carolyn M. Schussler, Notario Público.

(Sello notarial en relieve).

(Debe legalizarse hasta por Cónsul colombiano)

El suscrito Notario Público certifica que el Sr. David Curtis (nacionalidad) mayor de edad, domiciliado en Seattle, Washington (estado civil) compareció ante mí y firmó el documento precedente; que lo conozco personalmente y me consta que tiene capacidad legal para otorgarlo; que es Subsecretario de MICROSOFT CORPORATION Y EN CONSECUENCIA ESTA AUTORIZADO PARA

DAVID CURTIS
Subsecretario Oficial
Resolución No. 2196
de Abogacía

165

otorgar dicho documento; que la mencionada compañía existe y está constituida según las leyes del Estado de Washington de conformidad con las leyes de los E.U.A. y domiciliada en Redmond, Washington, y que el acto para el cual se otorga este documento está incluido entre aquellos que constituyen el objeto social de la mencionada compañía.

Estoy seguro de todos estos hechos porque he examinado los respectivos documentos consistentes en los Estatutos, de todo lo cual doy aquí fé.

Expedido y firmado en Redmond, Washington

Fecha el 12 de mayo de 1995

(fdo) Carolyn M. Schussler (Sello notarial en relieve)

Se requiere legalización por Cónsul colombiano. El Cónsul colombiano debe certificar que la compañía existe y ejerce su objeto social de acuerdo con las leyes del respectivo país y que el otorgante es su representante.

- - - - -

ESTADO DE WASHINGTON

SECRETARIO DE ESTADO

Yo, RALPH MUNRO, Secretario de Estado del Estado de Washington y guardián de su sello, por el presente certifico que se acuerdo con los registros de mi oficina CAROLYN M. SCHUSSLER es un Notario Público en y para el Estado de Washington.

Fecha: Mayo 22 de 1995

Dado bajo mi firma y el sello del Estado de Washington en Olympia, la Capital del Estado.

(fdo) Ralph Munro - Secretario de Estado

(Sello del Estado de Washington)

En español, legalizaciones por el Consulado General de Colombia en San Francisco, California, y hasta por el Ministerio de Relaciones Exteriores de Colombia bajo el N°135293 de junio 12 de 1995.

10

Es traducción fiel y completa del documento adjunto.

Santa Fe de Bogotá, D.C., junio 13 de 1995

[Handwritten signature]
INES DE HOPSON
Intendente Oficial
Resolución No. 2196
de Ajustada
[Handwritten signature]

MINISTERIO DE RELACIONES
EXTERIORES
Nº 0200
167131
[Handwritten signature]
FIRMA APROBADA
DOCUMENTO DESEMPLEA
FUNCIONES INDICADAS

NO SE ASUME
RESPONSABILIDAD DEL TEXTO
1995 JUN 18 A 11:28
[Handwritten signature]
JEFE DE LEGALIZACIONES
SECRETARÍA DE BOGOTÁ D.C.



SPRINGFIELD, ILLINOIS

APOSTILLE
APOSTILLE

(Convention de La Haye du 5 Octobre 1961)

1. Country: United States of America

This public document

2. has been signed by MARY A SMITH

3. acting in the capacity of NOTARY PUBLIC, COOK COUNTY

4. bears the seal/stamp of STATE OF ILLINOIS

Certified

5. Chicago, Illinois

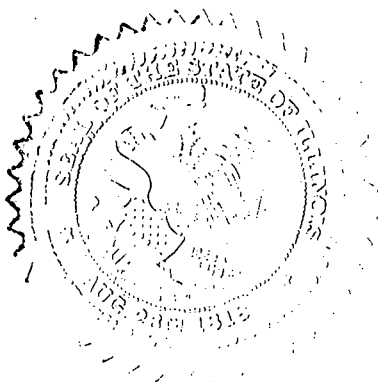
6. JUNE 10, 2004

7. by the Secretary of State, State of Illinois

8. No. C04JW19706

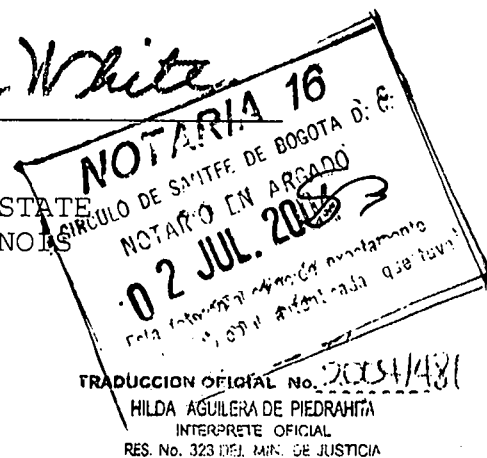
9. Seal/Stamp:

10. Signature:



JESSE WHITE
SECRETARY OF STATE
STATE OF ILLINOIS

Jesse White



CERTIFICATION OF ASSIGNMENT OF PRIORITY APPLICATION

I certify that the attached Assignment of Priority Application from SUJAL S, PARIKH, DMITRY TITOV, OLEG OVETCHKINE, GREGORY LETT, GRZEGORZ ZYGMUNT and DEBBIE A. NEWMAN in favor of Microsoft Corporation of their entire right, title and interest in and to the invention disclosed in the application for United States Letters Patent No. 10/434,850, filed May 15, 2003, and titled "SYSTEM FOR HOSTING GRAPHICAL LAYOUT/PRESENTATION OBJECTS," is an exact and true copy of the original Assignment.

Elizabeth J. Eich
Elizabeth J. Eich
Patent Paralegal
Leydig, Voit & Mayer, Ltd.

Date: June 9, 2004

COUNTY OF:)
STATE OF:) :SS

Subscribed and sworn to before me this 9th day of June, 2004, by Elizabeth J. Eich as Patent Paralegal of Leydig, Voit & Mayer, Ltd.

{SEAL}



Mary A. Smith
Notary Public

My Commission Expires: 7/29/2007





168
Karg
08/24

UNITED STATES PATENT AND TRADEMARK OFFICE

UNDER SECRETARY OF COMMERCE FOR INTELLECTUAL PROPERTY AND
DIRECTOR OF THE UNITED STATES PATENT AND TRADEMARK OFFICE

FEBRUARY 19, 2004

PTAS



102518662A

LEYDIG, VOIT & MAYER, LTD.
MARKJ JOY
TOW PRUDENTIAL PLAZA, SUITE 4900
CHICAGO, IL 60601-6780

LEYDIG, VOIT & MAYER
RECEIVED

FEB 24 2004

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

PAT/TRA Due Date Noted

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

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

RECORDATION DATE: 08/04/2003

REEL/FRAME: 014347/0251
NUMBER OF PAGES: 3

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

ASSIGNOR:
PARIKH, SUJAL S.

DOC DATE: 07/23/2003

ASSIGNOR:
TITOV, DMITRY

DOC DATE: 07/23/2003

ASSIGNOR:
OVETCHKINE, OLEG

DOC DATE: 07/23/2003

ASSIGNOR:
LETT, GREGORY

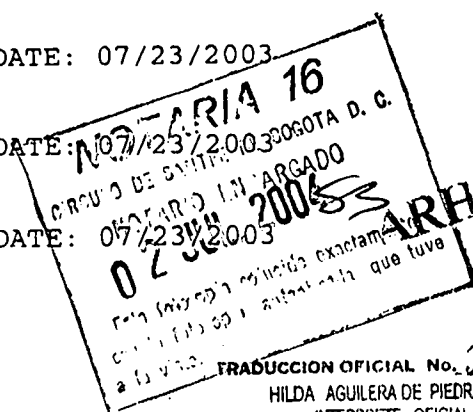
DOC DATE: 07/23/2003

ASSIGNOR:
ZYGMUNT, GRZEGORZ

DOC DATE: 07/23/2003

ASSIGNOR:
NEWMAN, DEBBIE A.

DOC DATE: 07/23/2003



LAH

TRADUCCION OFICIAL No. 2554481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

169

014347/0251 PAGE 2

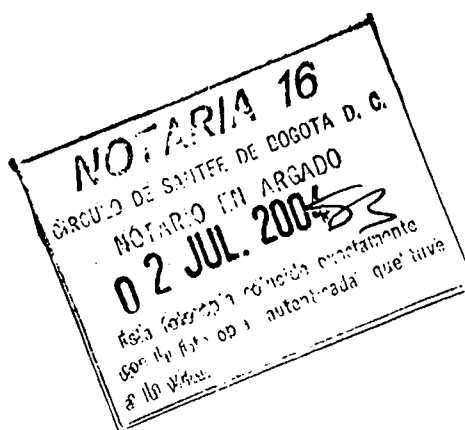
ASSIGNEE:

MICROSOFT CORPORATION
ONE MICROSOFT WAY
REDMOND, WASHINGTON 98052

SERIAL NUMBER: 10434850
PATENT NUMBER:

FILING DATE: 05/09/2003
ISSUE DATE:

JEFFREY OLSEN, EXAMINER
ASSIGNMENT DIVISION
OFFICE OF PUBLIC RECORDS



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

171

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Serial Number 10/434,850
Filing Date May 9, 2003
Inventorship PARIKH et al.
Applicant Microsoft Corporation
Attorney Docket No. 218428
MS Docket No. 300337.01
Title: SYSTEM FOR HOSTING GRAPHICAL LAYOUT/PRESENTATION OBJECTS

PATENT ASSIGNMENT

PARTIES TO THE ASSIGNMENT

Assignor(s):

Sujal S. Parikh
10804 180th Ct NE
Redmond, Washington 98052

Dmitry Titov
10401 180th Ct NE
Redmond, Washington 98052

Oleg Ovetchkine
11317 NE 117th Street
Kirkland, Washington 98034

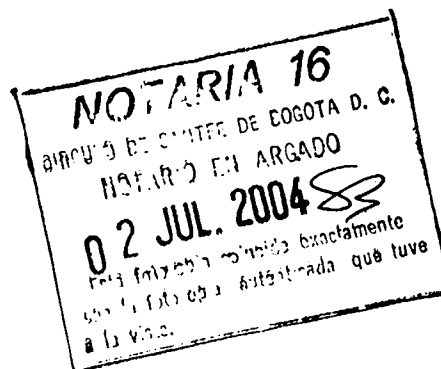
Gregory Lett
4050 W Lake Sammamish Pkwy NE, # 3
Redmond, Washington 98052

Grzegorz Zygmunt
417 226th Lane NE
Sammamish, Washington 98074

Debbie A. Newman
15307 NE 66th Ct
Redmond, Washington 98052

Assignee:

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

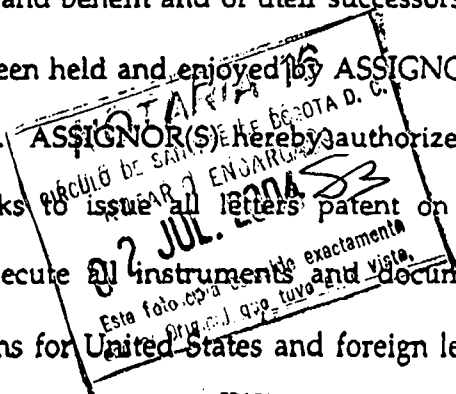


AGREEMENT

WHEREAS, ASSIGNOR(S) (listed above) are inventor(s) of an invention entitled "SYSTEM FOR HOSTING GRAPHICAL LAYOUT/PRESENTATION OBJECTS," as described and claimed in the specification forming part of an application for United States letters patent referenced above;

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

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



178

patent on said invention, for litigation regarding said letters patent, or for the purpose of protecting title to said invention or letters patent therefor.

7/23/2003
Date

S. Parikh
Sujal S. Parikh

7/23/03
Date

Dmitry Titov
Dmitry Titov

7/23/03
Date

Oleg Ovetchkine
Oleg Ovetchkine

7/23/03
Date

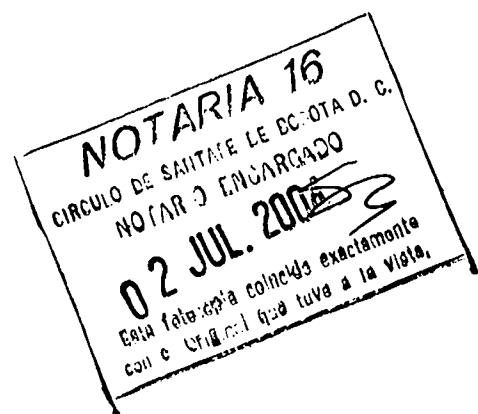
Gregory A. Lett
Gregory Lett

7/23/2003
Date

Grzegorz Zygmunt
Grzegorz Zygmunt

7/30/03
Date

Debbie A. Newman
Debbie A. Newman



Traducción Oficial No. 2004/481 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.

CERTIFICACION DE CESION DE SOLICITUD DE PRIORIDAD

Yo certifico que la anexa Cesión de Solicitud de Prioridad de SUJAL S, PARIKH, DMITRY TITOV, OLEG OVETCHKINE, GREGORY LETT, GRZEGORZ ZYGMUNT y DEBBIE A. NEWMAN a favor de Microsoft Corporation de su derecho entero, titulo e interés en y para la invención divulgada en la solicitud para Certificado de Patentes de los Estados Unidos No. 10/434,850, presentada el 15 de Mayo de 2003, y titulada "SISTEMA PARA HOSPEDAR OBJETOS GRAFICOS" es una copia exacta y verdadera de la cesión original.

(Firmado)

Elizabeth J. Eich

Paralegal Patentes

Leydig, Voit & Mayer, Ltd.

Fecha: Junio 9, 2004.



CONDADO DE :)

2004/481
 TRADUCCION OFICIAL No. 2004/481
 HILDA AGUILERA DE PIEDRAHITA
 INTERPRETE OFICIAL
 RES. No. 323 DEL MIN. DE JUSTICIA

ESTADO DE:) :SS

Suscrito y juramentado ante mi este día 9 de Junio de 2004, por Elizabeth J. Eich
como Paralegal Patentes de Leydig, Voit & Mayer, Ltd.

(Firmado)

Notario Público

Mi Comisión Expira 7/29/2007

(SELLO)

Sello a tinta: Sello Oficial, Mary A. Smith, Notario Público, Estado de Illinois, Mi
comisión expira 7/29/2007.

OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Bajo el Secretario de Comercio para la Propiedad Intelectual y Director de la
Oficina de Patentes y Marcas de los Estados Unidos, Washington D.C. 20231,
www.uspto.gov (Código de Barras *102518662A*)

Sello de Recibido de LEYDIG, VOIT & MAYER FEB 24, 2004 PAT/TM Fecha Exp.

Anotada

Febrero 19, 2004.

LEYDIG, VOIT & MAYER, LTD.

Markj Joy

Tow prudential Plaza, Suite 4900



TRADUCCION OFICIAL No. 2004/181
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

175

20

Chicago, IL 600601-6780

Oficina de Patentes y Marcas de los Estados Unidos

Notificación de registro de Documento de Cesión

El Documento anexo ha sido registrado por la División de Cesiones de la Oficina de Patentes y Marcas de los Estados Unidos. Una copia microfilmada completa se encuentra a disposición en el Cuarto de Búsqueda de Cesiones bajo el número de rollo y marco abajo en referencia.

Por favor revise toda la información contenida en esta notificación. La información contenida en esta notificación de registro refleja los datos presentes en el Sistema de Cesiones de Patentes y Marcas. Si llega a encontrar cualquier error o tiene preguntas acerca de esta notificación, usted podrá contactar al empleado cuyo nombre aparece en esta notificación al 703-308-9723. Por favor envíe petición de corrección a: Oficina de Patentes y Marcas de los Estados Unidos, División de Cesiones, Cajilla de Cesiones, CG-4, 1213 Jefferson Davis Hwy, Suite 320, Washington, D.C., 20231.

Fecha de Registro: 08/04/2003.

Rollo/Marco: 014347/0251

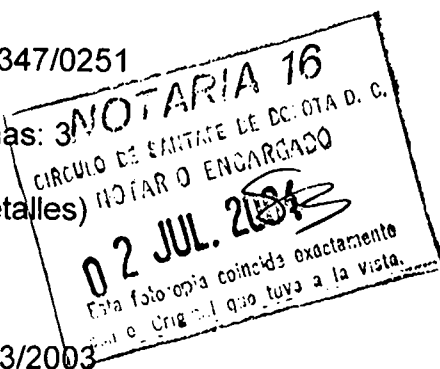
Número de Páginas: 3

Asunto: Cesión del Interés del Cedente (Ver documento para detalles)

Cedente:

PARIKH, SUJAL S.

Fecha Documento: 07/23/2003



TRADUCCION OFICIAL No. 2504/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

177

Cedente:

TITOV, DMITRY

Fecha Documento: 07/23/2003

Cedente:

OVETCHKINE, OLEG

Fecha Documento: 07/23/2003

Cedente:

LETT, GREGORY

Fecha Documento: 07/23/2003

Cedente:

ZYGMUNT, GRZEGORZ

Fecha Documento: 07/23/2003

Cedente:

NEWMAN, DEBBIE A.

Fecha Documento: 07/23/2003

014347/0251 PAGINA 2

Cesionario:

MICROSOFT CORPORATION

One Microsoft Way

Redmond, Washington 98052

número de Serie: 10434850

Fecha Presentación: 05/09/2003

Número de Patente:

Fecha Expedición:

Jeffrey Olsen, Examinador

División de Cesiones

Oficina de Registros Públicos



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

08-07-2003

Código de Barras

102518662

Departamento de Comercio de E.U.

Oficina de Marcas y Patentes de los E.U.

AL: Comisionado de Patentes y (ilegible) documento(s) o copia(s) originales.

Tipo de Presentación

☒ Nueva

☐ Representación (No-registro)

Documento ID#

☐ Corrección de Error PTO

Rollo#

Marco#

☐ Documento Correctivo

Rollo#

Marco#

Naturaleza de la Entrega:

☒ Cesión

☐ Licencia

☐ Fusión

☐ Gravamen

☐ Cambio de Nombre

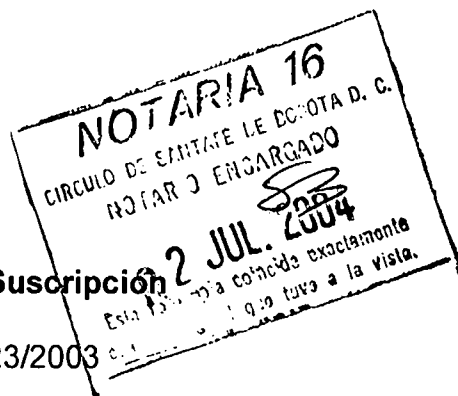
☐ Otro

Parte(s) que entregan:

1. PARIKH, SUJAL S.

Fecha de Suscripción

07/23/2003



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

2.TITOV. DMITRY	07/23/2003
3.OVETCHKINE, OLEG	07/23/2003
4.LETT, GREGORY	07/23/2003
5.ZYGMUNT, GRZEGORZ	07/23/2003
6.NEWMAN, DEBBIE A.	07/23/2003

_____ Marque si existen Nombres Adicionales de la(s) parte(s) que entregan anexas

Parte(s) que reciben:

Nombre: Microsoft Corporation.

Dirección One Microsoft Way

Dirección Ciudad: Redmond Estado: WA Código Postal: 98052

_____ Marque si existen Nombre(s) y Dirección(es) adicional(es) anexas

Nombre y Dirección Correspondencia

Marka Joy

Leydig, Voit & Mayer, Ltd.

Two Prudential Plaza, Suite 4900

Chicago, Illinois 600601-6780

Teléfono: (312)6165600

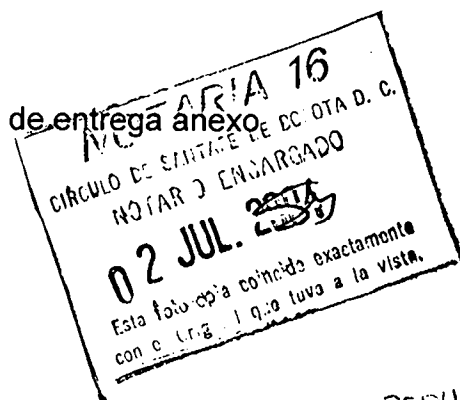
Facsimil: (312) 6165700

Número Expediente Abogado: 2218428

Páginas Entre el número total de páginas del documento de entrega anexo.

incluyendo cualquier anexo: 3

Número(s) de Solicitud o Número(s) de Patente:



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Entre bien sea el Número de Solicitud de Patente o el Número de Patente (No los dos números para la misma propiedad)

____ Marque si existen números adicionales anexos

Solicitud de Patente No.(s) 10/434.850

Patente No.(s)

Si este documento se está presentando junto con una nueva solicitud de Patente, entre la fecha en que la nueva solicitud de patente fue firmada por el primeramente nombrado inventor que suscribe. Mes Día Año

Tratado de Cooperación de Patentes (PCT)

Entre el número de solicitud PCT solamente si un número de Solicitud de EU no ha sido adjudicado.

PCT PCT PCT PCT PCT PCT

Número de Propiedades Entre el número total de propiedades involucradas: 1

Total Derechos

Total Derechos para las propiedades enumeradas (37 CFR 3.41)...\$ 40.00

Método de Pago

____ Adjunto se encuentra un cheque por la suma de de los cuales \$40.00 son para la cesión.

X cárguese a cuenta de deposito No. 12-1216 (fue autorizada a través de transmisión útil)

Autorización para cargar derechos adicionales a la Cuenta de Depósito No. 12-1216

1216 X Si ____ No

Declaración y Firma



TRADUCCION OFICIAL No. 2004/481
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. Los cargos a la Cuenta de Deposito están autorizados, como se indica en el presente.

Mark Joy

(Firmado)

Agosto 1, 2003

Nombre de la persona firmante Firma

Fecha

08/06/2003 GT0N11 00000144 121216 100434850

01 FC:0021 40.00 CH

OFICINA DE PATENTES Y MARCAS DE LOS ESTADOS UNIDOS

Número de Serie.....10/434.850

Fecha de Presentación.....Mayo 9, 2003

Inventor.....Parikh et. Al.

Solicitante.....Microsoft Corporation

Expediente Abogado.....218428

Expediente MS No.....300337.01

Título...: "SISTEMA PARA HOSPEDAR OBJETOS GRAFICOS"

CESION DE PATENTE

PARTES DE LA CESION

Cedente(s):

Sujal S. Parikh

10804 180th CtNE



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

182

Redmond, Washington 98052

Dmtry Titov

10401 180th Ct NE

Redmond, Washington 98052

Oleg Ovetchkine

11317 NE 117th Street

Kirkland, Washington 98034

Gregory Lett

4050 W Lake Sammamish Pkwy NE #3

Redmond, Washington 98052

Grzegorz Zygmunt

417 226th Lane NE

Sammamish, Washington 98074

Debbie A. Newman

1507 NE 66th Ct

Redmond, Washington 98052

Cesionario:



Microsoft Corporation

Sociedad en el Estado de Washington

One Microsoft Way

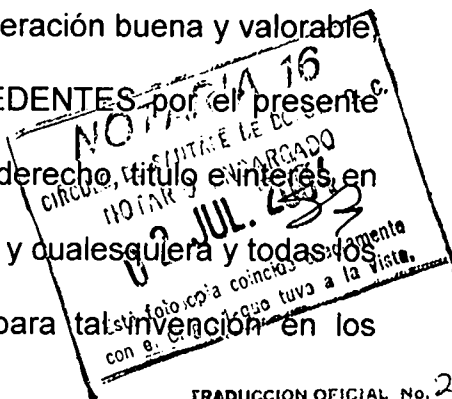
Redmond, WA 98052

ACUERDO

POR CUANTO, LOS CEDENTES (arriba enumerados) son inventores de una invención titulada "SISTEMA PARA HOSPEDAR OBJETOS GRAFICOS", 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 valorable, 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



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 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.

7/23/03

Fecha

(Firmado)

Sujal S. Parikh



TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

7/23/03

(Firmado)

Fecha

Dmtry Titov

7/23/03

(Firmado)

Fecha

Oleg Ovetchkine

7/23/03

(Firmado)

Fecha

Gregory Lett

7/23/03

(Firmado)

Fecha

Grzegorz Zygmunt

7/23/03

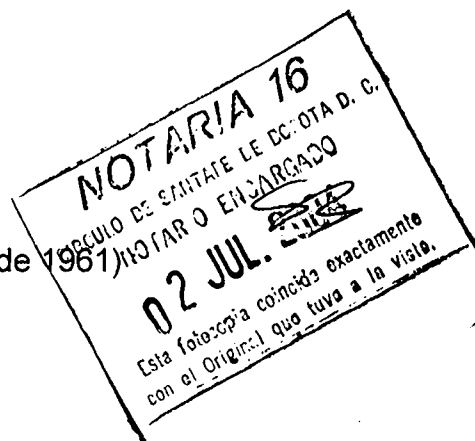
(Firmado)

Fecha

Debbie A. Newman

ESTADO DE ILLINOIS
SECRETARIO DE ESTADO
SPRINGFIELD, ILLINOIS
APOSTILLA

(Convención de la Haya de Octubre 5 de 1961)



1. País: Estados Unidos de América

TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES. No. 323 DEL MIN. DE JUSTICIA

Este documento público Documento anexo

2. Ha sido firmado por Mary A Smith
3. Actuando en calidad de Notario Público, Condado de Cook
4. Lleva el sello/estampilla del Estado de Illinois

CERTIFICADO

5. Chicago, Illinois
6. Junio 10, 2004.
- 7 Por El Secretario de Estado del Estado de Illinois
8. No. CO4JW19706
9. Sello (Sello Redondo en Relieve y de Oblea)
10. Firma (Firmado)

Jesse White

Secretario de Estado

Estado de Illinois

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., Junio 17 de 2004.

MINISTERIO DE RELACIONES
EXTERIORES

442033
Hilda Aguilera
COPIA FIDELMENTE DEL
DOCUMENTO ORIGINAL
LAS FUENTES DE INFORMACIÓN

NO SE ASUME
RESPONSABILIDAD DEL TEXTO

2004 JUN 28 A 11:57

JOSE DE LEGALIZACION
CORREO DE BOGOTA

NOTARIA 13
CIRCULO DE SANTAFE DE BOGOTA
NOTAR O ENCARGADO
02 JUL 2004
Esta foto copia coincide exactamente
con el original que tuvo a la vista.

TRADUCCION OFICIAL No. 2004/481
HILDA AGUILERA DE PIEDRAHITA
INTERPRETE OFICIAL
RES No. 323 DEL MIN DE JUSTICIA

Hilda Aguilera de Piedrahita

SYSTEM FOR HOSTING GRAPHICAL LAYOUT/PRESENTATION OBJECTS

FIELD OF THE INVENTION

5 The present invention relates generally to computing devices. More particularly, the present invention relates to computing components for arranging graphical elements displayed via a graphical document/user interface.

BACKGROUND OF THE INVENTION

10 Displaying and/or rendering graphical output for an application executing upon a computing system includes many tasks. One such task is layout/presentation management. Layout/presentation management concerns arranging and adorning a set of display elements within their assigned display spaces (e.g., rectangles). While "layout" and "presentation" may be distinguishable in other contexts, presentation and layout, as used herein, should be treated as interchangeable and equivalent terms. The layout/presentation system-processed display elements provide graphical output data to rendering components of a graphics output system and graphical display drivers. Such rendering components drive graphical display hardware such as monitors and printers.

15 The layout/presentation operations executed by computing systems comprise a wide variety of functions performed on graphical display elements. An example of such an operation is arranging text within assigned dimensions of an editor application's graphical display user interface. The layout operation determines the content and placement of the lines of text within a designated rectangular space. Another example is framing an element (e.g., a dialog box, a toolbar, a control bar, etc.) with a standardized border. The operations are also referred to herein as "features." Rendering components of a computer system produce graphical output data based upon the state of the display elements that are potentially modified by previously executed layout/presentation operations/features.

20 Various presentation engines are known. For example, MICROSOFT WINDOWS' "user module" performs layout operations with regard to top-level windows on a graphical user interface. MICROSOFT WINDOWS' dialog box manager places objects at pre-defined positions and allows grouping of objects. The positioning of the

objects occurs in a logical manner and is independent of a physical device. JAVA SWING permits basic placement operations with regard to objects within a view based on properties specified on the objects.

Layout/presentation operations have been performed in a number of manners. An application itself can fully carry out its display element layout operation. The layout (or "presentation") operation is encapsulated within the application. The application adorns/arranges display elements for a view or view(s) associated with a current state of the application. The application arranges the elements within a display region assigned to the application. The application thereafter renders display output, corresponding to the arranged display elements, to the graphics system/drivers.

Another layout processing arrangement, incorporated into the MICROSOFT WINDOWS XP operating system, provides support for a pre-defined set of layout features that are callable by applications. The set of layout features comprise specialized layout/presentation operations for user interface graphical display elements of applications. The set of pre-defined layout/presentation features, such as for example, a border generator for supplied rectangles, reduces the programming load on application developers with regard to specific aspects of organizing and presenting elements of a graphical display. The set of pre-defined features also facilitates consistency with regard to executing certain basic display features/functions such as the aforementioned borders. Applications supplement the pre-defined layout functionalities with additional layout operations embedded within the applications themselves.

The known layout processing architecture, including pre-defined layout features callable by an application, simplifies programming layout tasks encountered by applications during the course of arranging a view before rendering graphical data and commands to rendering components of a computer system. However, the known layout processing architecture does not readily support enhancements or modifications with regard to the set of pre-defined layout features callable by the applications. Thus, for example, new layout features are placed directly into the application code rather than placing the features in the pre-defined layout features provided by the presentation/layout architecture. Thereafter, during execution, the application calculates the layout using a

189
combination of the pre-defined layout functionalities and internally executed layout/presentation operations.

SUMMARY OF THE INVENTION

5 The present invention comprises a presenter system for incorporation within a graphical output layout management system. Such layout management is directed to the way in which objects are arranged, sized, and placed upon a display space. Such systems process a layout, on behalf of a program providing graphical elements containing data representing displayable content of the program (e.g., a block of text that needs to be
10 arranged on a displayed or printed page). The presenters perform a complimentary role to the graphical elements (that specify data) by defining display states for presenting the content of the graphical elements. As such, the presenters maintain a layout description for a corresponding graphical element.

15 In accordance with the present invention, the presenter system hosts and arranges the presenters associated with graphical elements within a view. The presenter system, in its role of supporting the derivation of presenters of many types, includes a presenter base class from which presenter classes are derived. Thereafter, presenters are instantiated from the derived presenter classes.

20 The presenter system also includes a presenter host interface. The host interface comprises a method for laying out graphical elements in a view in accordance with the presenters associated with the graphical elements. As such, the layout of the graphical elements within a view is specified by the associated presenters.

BRIEF DESCRIPTION OF THE DRAWINGS

While the appended claims set forth the features of the present invention with particularity, the invention and its advantages are best understood from the following detailed description taken in conjunction with the accompanying drawings, of which:

5 Fig. 1 is a block diagram depicting an exemplary computer system for carrying out an embodiment of the invention;

 Fig. 2 is a high level schematic diagram depicting the primary components of a layout management architecture including a presenter system for carrying out graphical layout maintenance in a computing system embodying the present invention;

10 Fig. 3 is a schematic drawing depicting the relationships between graphical elements and presenters defining a graphical document/user interface in a system embodying the present invention;

 Figs. 4a and 4b summarize the structure of a presenter base class from which customized presenter object classes are derived to carry out particular layout/rendering tasks in a system embodying the present invention;

 Fig. 5 summarizes the structure of a child proxy class;

 Fig. 6 summarizes the structure of a notification handler class interface;

 Fig. 7 summarizes the structure of a notification handler site class;

 Fig. 8 summarizes the structure of a view class;

20 Fig. 9 summarizes the parts of a BoxSizeInfo structure;

 Fig. 10 summarizes the parts of a page descriptor structure;

 Fig. 11 summarizes the structure of a presentation engine class;

 Fig. 12 is a flowchart summarizing exemplary steps for notification handler-based processing of a change to a graphical element;

25 Fig. 13 is a flowchart summarizing exemplary steps for re-calculating and rendering a view comprising a set of presenters in accordance with an embodiment of the present invention; and

 Fig. 14 is a flowchart illustratively depicting the steps for executing an update on a presenter (and associated affected children) in response to a change to a related graphical element.

DETAILED DESCRIPTION OF THE DRAWINGS

A user interface and document layout/presentation architecture is disclosed herein. The disclosed layout/presentation architecture includes graphical layout processing/rendering components (presenters) that carry out view aspects of associated
5 underlying data (graphical elements) for a document/user interface. A presenter system hosts the presenters. In its host role, the presenter system orchestrates and coordinates updating a view, comprising a set of presenters, in response to changes to corresponding graphical elements. Thus, the presenter system provides a mechanism that links the status (e.g., data state) of graphical elements to view states updated, cached, and rendered
10 by corresponding presenter objects.

An aspect of the presenter system-based layout/presentation architecture embodying the present invention is its high degree of extensibility with regard to the types of presenters that it hosts. A presenter base class provided by the presenter system facilitates extending a set of presenter types (object classes) available for laying out and
15 rendering information provided by a corresponding graphical element. The various presenter types derived from the presenter base class incorporate particular layout/rendering methods as overrides to default methods specified by the base presenter class. The presenter objects instantiated from the set of presenter object classes implement layout/presentation features according to the data state of a corresponding
20 graphical element.

The presenter system hosts the presenter objects instantiated from the set of presenter object classes. The presenter system performs lifetime management (i.e., creation, deletion, etc.) of presenter objects. The presenter system handles dirty tracking to selectively update only the changed presenters. The presenter system supports
25 chaining presenters together to provide adornments to main presenters. The presenter system also coordinates updating and rendering presenters.

Notification of changes to graphical elements potentially effecting the view states is facilitated, in an embodiment of the invention, by notification handlers. The notification handlers enable change notification handling to be separated from the change
30 processing performed by presenters. Each presenter element type is associated with a notification handler type. Each notification handler processes changes to a corresponding

graphical element. If updating is needed, then the notification handler initiates setting a dirtiness property on an associated presenter object. The set dirtiness property causes recalculation of a presenter object in response to the changes to the corresponding graphical element.

5 The user interface layout/presentation architecture, described by way of example herein, is incorporated into an operating system that hosts applications having graphical user/document interfaces. The applications, presenters, and device-specific rendering components of the operating system and device-specific drivers, render interface/document graphical output device control commands/data to display device
10 hardware (e.g., monitors, printers, etc.). Layout features of the rendered output are based at least in part upon layout/presentation processing and rendering performed by the presenters hosted by the presenter system.

 The presentation architecture enables disassociating a displayed element's data state from it's view state. Such disassociation provides an extensible and highly
15 customizable platform for creating new and rich display personalities for an underlying graphical element. In an embodiment of the invention, a display element is represented by a graphical element object (representing the element's data state) and at least one associated presenter object (representing a view state) for laying out the element in a particular view. The presenter carries out customized updating of a layout for a related
20 element. The presenter also renders the view of the element after updating the layout.

 Utilizing presenter objects disassociates the data state of a displayed element and its display in a view. As a consequence, each graphical element (display data source) is potentially capable of having multiple presenters providing distinct views of the graphical element (e.g., a full scale and thumbnail view of a photograph element). The ability to
25 associate multiple presenters with a single graphical element also enables a single element to be associated with multiple locations within a field (e.g., splitting a single graphical element between multiple columns). Also, the presenter system supports chaining multiple distinct presenters of differing types to a single element thereby supporting attaching a variety of customized adornments to a basic view for an element.
30 These, as well as other aspects of an exemplary display layout/rendering architecture are described herein below with reference to the figures.

FIG. 1 illustratively depicts an example of a suitable operating environment 100 for carrying out the presenter system-based display layout/rendering architecture embodying the present invention. The operating environment 100 is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Other well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, laptop/portable computing devices, hand-held computing devices, multiprocessor systems, microprocessor-based systems, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like. The disclosed layout management architecture including a presenter system serves a number of diverse interests including extensibility, integration, and uniformity with regard to the display functionality of a computing system, such as the one depicted in **FIG. 1**.

The invention is described in the general context of a set of steps and processes carried out by computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Though the exemplary embodiment is described with reference to locally executed processes on a single computer system, the invention is potentially incorporated within network nodes operating in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules are generally located in both local and remote computer storage media including memory storage devices.

With continued reference to **FIG. 1**, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be

any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video
5 Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By
10 way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not
15 limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by computer 110. Communication media typically embodies computer readable instructions,
20 data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such
25 as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access
30 memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as

195

during start-up, is sometimes stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

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

The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 20 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often

196

connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device may also be connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195.

The computer 110 potentially operates in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on memory storage device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Figure 2 is a high level schematic diagram identifying software modules and programs as well as instantiated objects and data components of a layout/presentation

147

management system architecture embodying the present invention. The exemplary layout/presentation management system architecture represented in FIG. 2 includes a presenter system 200 that supports layout/presentation of displayable objects based upon a set of hierarchically arranged presenter objects associated with a view 202 created, from
5 a view object class (see, FIG. 8 described herein below) at the request of an application 204 executing on an operating system 205.

In an embodiment of the invention, each view (e.g., view 202) has a corresponding presenter system (e.g., presenter system 200) that is responsible for maintaining the displayable objects within the view. In an embodiment of the invention
10 the view 202 is the visual root for all the visuals that are created by the presenter system 200. The displayed content within the view 202 is based upon graphical elements 206 within a backing store 208. The application 204 is a source of the graphical elements 206 within the backing store 208. An illustrative example depicting the relationships between views, presenter systems, presenter trees for views, and graphical element trees
15 associated with an application is depicted in FIG. 3 described herein below.

The layout/presentation management system architecture depicted in FIG. 2 is divided into a set of functional components. The lines between the depicted components represent interaction paths between the components. The present invention is not limited
20 to the particular arrangement of the components and their indicated exemplary interactions in the illustrative embodiment depicted in FIG. 2. Rather, the functionality of the components described herein below is grouped in different manners in accordance with alternative embodiments of the invention.

In an embodiment of the invention, the application 204 calls a constructor method
25 on a view object class application program interface (See view class structure in FIG. 8 described herein below) to create a root object for the view 202 object instance. The method call to create the view 202 passes a reference to a root graphical element of the graphical elements 206 contained in the backing store 208. After creating the view 202, a call to a DoLayout method on the view 202 creates a presenter tree for the view 202 with
30 a root presenter object corresponding to the root graphical element passed initially to the constructor method that created the view 202. The presenter tree (not shown in FIG. 2)

for the view 202 comprises a set of presenters that are responsible for organizing and rendering visual output for the view 202.

In an embodiment of the invention each view (e.g., view 202) owns a presenter system (e.g., presenter system 200). Thus, in response to the DoLayout method, the view
5 202 initiates creating a new presenter tree and returns it to the caller (e.g., application 204). Initially, the view 202 calls a constructor method on a presentation engine object class to create a presentation engine 212 instance including a presenter host application program interface (presenter host API). The presentation engine 212 includes executable
10 code for coordinating and orchestrating execution of layout/presentation operations in a computing system supporting graphical output. Functions performed/coordinated by the presentation engine 212 include: creating presenter objects (both root and child versions), coordinating re-calculating presenters' display states, rendering, lifetime management of presenters, dirty tracking, chaining multiple presenters to a single element, and performing incremental layout. Examples of the callable methods on the
15 presentation engine 212's presenter host API are described herein below with reference to FIG. 12. These methods are high level requests that create and maintain the presenter object tree for the view 202. The identified tasks performed by the presentation engine 212 are exemplary, and those skilled in the art will appreciate that additional/alternative tasks can be carried out by the presentation engine 212 in alternative embodiments of the
20 invention.

The presentation engine 212 builds the presenter system 200 to handle presenter objects associated with the view 202. In an embodiment of the invention, the presenter system 200 includes a presenter base class 216. The presenter base class 216 is a full featured base class for creating presenter subclasses and presenter objects capable of
25 exploiting the layout management functionality of the system described herein. The presenter base class 216 comprises a template, including virtual (overridable) program code and data structures, from which a set of presenter classes 214 are derived. The presenter classes 214 comprise a set of specialized presenter types corresponding to views, of graphical elements, having particular display properties/behaviors. Customized
30 presenter classes 214 derived from the presenter base class 216, by way of example, lay out content, arrange children, create visuals for attached elements, paginate content for

typographic documents, and incrementally update visuals including the graphical elements 206. The specialized functionality of each presenter class within the set of presenter classes 214 is established by overriding and supplementing the virtual methods defined by the presenter base class 216. Presenter objects instantiated from the set of presenter classes 214 represent and handle display aspects of the graphical elements 206.

In an embodiment of the invention the set of presenter object classes 214, including both predefined presenter classes 218 and external presenter classes 220, is extensible. The set of predefined presenter classes 218 are provided with the presenter system 200. The external presenter object classes 220 are generally installed independently of the presenter system 200. However, properly integrating the external presenter object classes 210 with the presenter system 200 is facilitated by a published interface specification for the presenter base class 216 (described herein below with reference to FIGs. 4a and 4b). A developer of external presenter classes 220 overrides/modifies program code and data structures associated with the customizable layout/rendering methods of the presenter base class 216 to achieve custom sizing, child positioning, and rendering functions. Thereafter, the new presenter class is installed in the set of external presenter object classes 220, by for example, installing a file containing the new object class in a designated directory within a computer's file system.

In an embodiment of the invention, the presenter classes 214 include presenter classes for carrying out layout features including, by way of example: content layout, child object arrangement, visuals creation, document content pagination, and incremental update of visuals (e.g., HTML pages that span multiple display screens). Examples of specific presenter types include: a text presenter that measures and displays text formatted in various ways (e.g., bold, italic etc), an image presenter that measures and displays a specified image, and a dock presenter that partitions a space among multiple children presenters.

After creating the presenter system 200 and the presenter classes 214, a set of presenter objects associated with the view 202 and the presenter system 200 are instantiated for the view – beginning with a root presenter object. It is noted that the constructor method call to the presentation engine object class to create the presentation engine 212 identifies the root graphical element for the view 202. After instantiation, the

200

presentation engine 212 calls the graphical elements 206 to determine a presenter object type associated with the root graphical element for the view 202. After determining the graphical element's presenter type, the presentation engine 212 calls a method, on one of the presenter object classes 214 corresponding to the presenter object type identified by
5 the graphical elements 206, to instantiate the root presenter object for the view 202.

As indicated by the path between the presenter classes 214 and the backing store, presenter objects, including the root presenter, access the graphical elements 206 of the backing store 208 to determine the properties of particular graphical elements. By way of example, a presenter object for a particular graphical element is able to identify any
10 graphical element children of the particular graphical element and the presenter type for each child graphical element. Thereafter, the presenter object calls, for each identified child graphical element, a method on itself (e.g., GetChildProxyForElement 486) to instantiate a child proxy object from a child proxy class 222 of the presenter system 200. Child proxy objects are wrapper objects for presenter objects (instantiated from the
15 presenter object classes 214) that are nested within the display space allocated to a parent presenter object. The child proxy class 222 meets encapsulation and security needs (e.g., a child presenter object limits access by a parent presenter object, or the application 204, to the contents of a presenter contained within the child presenter). Alternatively, the encapsulation layer provided by the child proxy objects is bypassed and the parent
20 presenters directly access their child presenters.

The method on the presenter object for creating a child proxy object for an element, in an embodiment of the invention, calls a constructor method on the child proxy class 222 and passes the identity of the child graphical element to instantiate a new child proxy object and presenter object for the child graphical element. Once
25 instantiated, the new child proxy object creates a presenter object for the child graphical element via a presenter constructor corresponding to the presenter type specified on the identified child graphical element. The child proxy object, in turn, returns its own reference to the parent presenter object. The parent presenter object maintains the returned child proxy object reference in an array of references to child proxy objects. As
30 such the child proxy object array entries for a presenter object correspond to the graphical

201
element children for a parent graphical element with which the parent presenter is associated.

Another aspect of the graphical element arrangement/presentation architecture depicted in FIG. 2 is change notification. In an embodiment of the present invention, a
5 set of notification handler classes 224 are provided for presenter objects instantiated from the presenter classes 214. The notification handler classes 224 are defined according to a notification handler base class 225. The notification handler base class 225 is an abstract class that defines an interface for the notification handler classes 224. The notification
10 handler base class 225 provides the interface definition of two virtual methods (see FIG. 6), the implementation of which is provided in the notification handler classes 224 in accordance with their corresponding presenter classes 214. The notification handler classes 224 aid incremental layout updating – recalculating only the presenter objects effected by changes to a set of corresponding graphical elements. In an embodiment of
15 the invention, a notification handler is specified for graphical element and view combinations. In cases where multiple presenters are created for a single graphical element/view combination, the single notification handler operates as the single source for tracking changes to the element for purposes of updating the multiple presenters for the element in a view. Each presenter type specifies a particular corresponding
20 notification handler type. When a presenter object for an element is instantiated that requires a notification handler, the presenter system creates a corresponding notification handler of the specified type and links the notification handler to the presenter object.

Instances of the notification handler classes 224 determine which portions of the presenter objects are dirty and therefore require updating. Initially, a notification
25 relationship is established between a graphical element and a notification handler created for a presenter in a view (e.g., view 202 for application 204). The notification handler receives change notifications associated with the graphical element in the backing store 208 through the presentation engine 212 of the presenter system 200. The notification handler determines a portion, if any, of the presenter that has been affected by the change
30 to the element. The notification handler returns the affected portion information to the presenter system 200. The presenter is marked as "dirty" by the presentation engine 212.

202

The presenter system 200 accumulates the "dirty" presenter information for all presenters in a data structure maintained for a view (e.g., view 202). At a later time, in response to a call to re-calculate the layout for a view, the presentation engine 212 of the presenter system 200 invokes an update method/operation only on the dirty presenters. Thus, the notification handler classes 224 enable limiting layout updates to the affected ones of the presenter objects created from the presenter classes 214.

The layout management system architecture embodying the present invention operates independently of any particular output hardware devices upon which the laid out elements are ultimately displayed according to their defined views. The layout/presentation management system architecture depicted in FIG. 2 performs a preprocessing step with regard to graphical display output. In an embodiment of the invention, the arranged output associated with the layout/presentation management system architecture is device independent. Thereafter, during a rendering stage the presenter objects render the re-calculated view to graphical output components of a computing system. However, in other embodiments of the invention, the output of the layout/presentation system is returned by either the presenter objects 214 or the presenter system 200 to the calling application 204, and the application 204 performs the rendering task. In an embodiment of the invention, the rendering commands, whether issued by the presenter objects derived from the presenter classes 214, the view 202, or the application 204, are passed to a graphics subsystem and graphics device drivers 234 on the user's computer. The graphics subsystem and graphics device drivers 234 transform the rendering instructions and data to a particular selected output device such as a monitor 236 or a printer 238.

In summary of the layout/presentation management system architecture described above, a presenter system 200 provides a device-independent layout processing platform for applications such as the application 204. Layout processing is performed by presenter objects instantiated from a set of presenter classes 214 hosted by the presenter system 200. The presenter system 200 is owned by the view 202. The presenter objects correspond to particular view states/renderings of the graphical elements 206. Separating view states (presenter objects) from element states (graphical elements 206) enables multiple view states/presenters to be independently designated for a single element state.

The layout is separated into views, such as view 202, wherein each view corresponds to a display region such as a particular rectangle on an output screen allocated to the application 204 or a document page in graphical printer output. The separation of data and view states enables a single element to drive multiple views of that graphical element and thus enable a single graphical element to be displayed in a variety of views/ways.

Another aspect of the disclosed layout management system architecture is its high degree of extensibility. Extending the set of presenter classes 214 is facilitated by the presenter base class 216 provided by the presenter system 200 and the presenter host API of the presentation engine 212.

The above-described layout management system architecture supports a variety of advanced layout processing features for applications rendering graphical output to graphical user interfaces and printers. The ability to separate a single element (graphical element) into multiple view states (presenters) aids pagination and dividing elements into multiple columns (a form of pagination) in applications when rendering printer output or a printer output view of a document on a display screen. Such applications include web browsers and word processing programs.

Another benefit of the multiple presenters is the ability for applications to perform multiple attempts at fitting their content within a given display space. The applications can instantiate multiple tries on different presenter objects, for a single element, with different input parameters, and then allow a user to select a best one of the displayed views of the graphical element(s).

Yet another benefit arising from the above-described layout management architecture is the ability to perform incremental updates on a view comprising a set of presenters for a set of corresponding elements. Change is implemented on an element-by-element basis in the backing store (See, e.g., backing store 308 in FIG. 3). The notification handlers and the presenter system 200 provide an infrastructure for limiting update of a corresponding view to the presenters that have been affected by changes to corresponding elements in the backing store.

Still another benefit of the exemplary layout management architecture is the ability to adorn view states of elements embodied in previously created presenters. This is accomplished by chaining additional presenters onto a presenter of a type specified for

204

a particular element type. Chaining enables developers to enhance user interfaces and document output by adding borders, backgrounds, etc. to a view state specified by a particular presenter type for an element.

5 Having described the general architecture of the layout management system embodying the present invention, attention is directed to **FIG. 3** that depicts relationships between views, presentation engines, presenter objects within views, graphical elements, and notification handlers for a particular exemplary view.

Conceptually, a view defines a particular way for laying out a set of graphical
10 elements represented in a backing store 208 according to specified presenters. In an embodiment of the invention, a view is defined by a rectangle in an output field (e.g., a display screen). In addition to a rectangle and its position, a view also specifies graphical elements and associated presenters contained in the view. Furthermore, it is noted that in an embodiment of the invention each view is associated with its own presenter system
15 instance (e.g., presenter system 200a for view 202a and presenter system 200b for view 202b).

With continued reference to **FIG. 3**, the backing store 308 comprises an organized collection of graphical elements (Ex) associated with views 202a and 202b. The graphical elements (Ex) are entities of organized user content. Examples of elements
20 include: a graphical user interface (GUI) button or text panel, a text editor window, a bitmap, etc. In an embodiment of the invention, the graphical elements (Ex) in the backing store 208 are arranged in the form of a tree. Each element (Ex) in the set of graphical elements 206 maintained within the backing store 208 is associated with a set of properties/placeholders (which can be either explicit or implied from other explicit
25 properties/placeholders). One or more such properties/placeholders on the graphical elements specify a presenter type that is used to lay out and render the content of the graphical element in cooperation with other components of the computer system. In embodiments of the invention, the chained presenters are explicitly defined, or alternatively, are implied from other properties specified on an element (e.g., a "border"
30 property of two pixels implies a border chained presenter). Property types specified for

graphical elements depend upon the particular graphical element and include: color, font name, height, width, etc.

The backing store 208 is associated with potentially multiple views (e.g., views 202a and 202b). Each one of the potentially multiple views exclusively owns a set of
5 presenter object instances corresponding to a set of graphical elements within the backing store 208. Thus, in the case of a graphical element being present in multiple views, a presenter object is created for the element within each view (e.g., presenter P2 and presenter P2' for element E2). The graphical element with which the presenter is associated is specified in an element owner field and corresponds to the dashed lines from
10 the presenters to their corresponding graphical elements. This is just one example of the presentation architecture represented in FIGs. 2 and 3 supporting multiple presenters operating upon information provided by a single graphical element.

As a consequence of a presentation architecture embodying the above-described ability to attach multiple views (and presenters) to a same collection of graphical
15 elements in the backing store 208, a graphical user interface is capable of displaying multiple views of a same set of information provided by graphical elements. For example, in the an application such as MICROSOFT's POWERPOINT, two simultaneous views of a slide are supported, the main view and the preview (or thumbnail) view, for a same slide graphical element.

20 The ability to create multiple presenter object instances for a single graphical element also facilitates pagination of a single graphical element (e.g., document text) across multiple pages/columns (i.e., where one graphical element is split across two pages, two presenter objects are created for the element – one for each page). With reference to FIG. 3, a parent presenter object (e.g., P5) receives a passed property from a
25 graphical element (e.g., E5) indicating that content within the element (e.g., text element E8) is presentable across two columns. Two child presenter objects P8a and P8b, corresponding to two separate and distinct column/page rectangles on the document, are instantiated under presenter P5 for displaying text provided by graphical element E8.

Yet another case of instantiating multiple presenters for a single graphical element
30 arises in the case of adornments (e.g., borders, backgrounds, frames, etc.) assigned to a graphical element. An example of a chained presenter is provided by chained presenter

Q4 for element E4 that provides a border around a rectangle defined by presenter P4 to hold the text supplied by a data field within element E4. Both presenter Q4 and presenter P4 are linked to/associated with a graphical element E4 by specifying the graphical element E4 in their element owner fields. In an embodiment of the invention, a parent

5 presenter detects from an element corresponding to one of its children that a chained presenter is needed (e.g., a border of two pixels is specified on the element). In the case of the chained presenter Q4, when presenter P1 calls a method on itself to create a child (proxy) presenter for a child element E4, it recognizes from properties on E4 that a chained presenter (Q4) is needed. First the child proxy/presenter Q4 is created and is

10 identified as a child presenter of presenter P1. Q4, in turn, creates the child proxy/presenter P4. P4 is thus a child presenter of presenter Q4. Information is stored within the child proxy for chained presenter Q4 identifying the next presenter (P4) in the chain.

Having described particular exemplary cases involving multiple presenters being

15 linked to a same graphical element and creating multiple views (and presenter systems) for a set of graphical elements in a backing store 208, the relationships between the displayed entities in FIG. 3 are now described. As mentioned above, the views 202a and 202b own presenter systems 200a and 200b, respectively. The presenter system 200a created and supports a set of presenters 302 for the view 202a through code and a base

20 class including an API to which the presenter classes are written. The presenter system 200b created and supports a set of presenters 303 for the view 202b.

As indicated by the lines between presenters (Px) and graphical elements (Ex) in the backing store 208, the presenters (Px) are associated with corresponding graphical elements (Ex). The presenters (Px) represent view states for corresponding graphical

25 elements (Ex). The graphical elements (Ex) in the backing store 304 are maintained, by way of example, in a tree structure built by an application in one of many possible ways. Examples of such ways include running a text markup file through a parser, using code to instantiate the graphical elements, etc. The set of presenters 302 tree structure is built by the presenter system 200a in response to a call to a particular method on the interface

30 (e.g., DoLayout) provided by its presentation engine. As explained previously herein above with reference to FIG. 2, the nodes of the presenter object tree are determined by

the nodes of the graphical element tree associated with a particular view. The graphical elements within a view are determined by the root graphical element specified for the view, its children, grandchildren, etc..

The set of presenters 302 in view 1 are arranged hierarchically. A root presenter P1 has two children presenters P2 and P4 (Q4 is an adornment presenter on presenter P4). Presenter P2 has no children. Presenter P4 however has two children presenters P5 and P7. Presenter P5, in turn, has child presenters P8a and P8b that are associated with a single graphical element E8. Such sharing/dividing of a single graphical presenter between two presenter objects arises, by way of example, when a text graphical element E8 is divided between two columns of a parent text graphical element E5, which has a presenter (P5) of type TextPresenter. Presenter P5 initiates a column division process when the presenter P5 notes a property on E5 requesting E5's content (E8) to be divided across two columns. The presenters P8a and P8b correspond to the graphical existence of parts of graphical element E8 in the two columns.

It has been previously noted that a single graphical element can have multiple associated presenter objects. Three exemplary cases have been identified. In a first scenario, depicted by the multiple views 202a and 202b and corresponding presenter systems 200a and 200b in FIG. 3, a single graphical element (e.g., E2) has corresponding presenters P2 and P2' associated with views 202a and 202b, respectively. Such views correspond, by way of example, to a thumbnail view and a full size view of a photograph image (E2) displayed at different regions of a user interface generated by a photographic image processing application. In this case, presenters P2 and P2' are created for the element E2 in each view 202a and 202b containing the element. In a second scenario involving multiple presenters for a single element (e.g., text element E8), presenter elements P8a and P8b correspond to two columns within a rectangle defined by a single presenter element P5. Alternatively, such cases of multiple presenters in a view referencing a same graphical element arise, for example, when a parent performs multiple attempts at laying out a view and saves the tries. In a third scenario, a presenter (e.g., Q4) that is dependent upon another presenter (e.g., P4) – as opposed to the data state of a graphical element (e.g., E4) – is chained to the other presenter thereby enriching the

display features provided by the original presenter. An example of presenter chaining is a border presenter providing a border adornment to a main presenter.

It has been previously noted above, with reference to FIG. 2, that child proxy objects instantiated from the child proxy class 222 are wrappers for each child presenter object of a parent presenter object. In the graphical depiction of the presenter tree structure for view 202a, the child proxy objects are represented by the arrowheads in the lines connecting a parent presenter to its child presenter objects. The child proxy objects limit access by a parent presenter object to its child presenter objects.

Notification handlers (Nx) are responsible for notifying corresponding presenters (Px) of data state changes to their graphical elements (Ex). Notification handlers Nx are not essential to the operation of the presentation/layout management system. Rather, they facilitate incremental updates to less than all the presenters when a change occurs in an element within a view.

With reference to FIGs. 4a and 4b, the presenter base class includes a set of methods, properties and fields. In an embodiment of the invention, the properties are accessed on a particular referenced presenter object by invoking get/set operations on the object. A background property 400 specifies a paint value for the presenter background. A background opacity property 402 stores a value specifying an opacity for the background. A border property 404 stores a value specifying the thickness of a border drawn around the edge of the presenter's layout space. The border property 404 stores, for example, four values indicating the border widths for the top, bottom, right and left sides of a rectangle assigned to a presenter. A border color property 406 specifies a value designating the color of the border around a presenter. A border style property 408 stores a value indicating one of a set of border display styles. Examples of border styles include: solid, groove, ridge, inset and outset.

A bounding box property 410 specifies the area (e.g., rectangle) in which the presenter graphics (or children) are drawn. The default size of the bounding box property 410 is the same as a size specified by a layout size field 460. However, the bounding box size can exceed the layout size thereby enabling the presenter to draw outside the layout size.

On the other hand, if a clip property 414 is set to true, then the presenter (and its children presenters) can not be drawn outside a rectangle specified by a layout size field 460. If the clip property 414 is set to true, then the bounding box property 410 is no larger than the layout size. This ensures that the drawn presenter does not render
5 graphics outside its assigned layout space.

A children property 412 is an array, or alternatively any other suitable multi-element data structure, that stores references (e.g., handles, pointers, direct/indirect etc.) to child presenters nested within a parent presenter and their transformed locations within the parent presenter's layout space. The children property 412 is populated on first use in
10 the context of an OnUpdate method call (discussed herein below) on the presenter. "On first use" refers to when a parent presenter sequentially requests the child collection and the children collection is populated with presenters corresponding to child graphical elements, of the graphical element corresponding to the parent presenter, from the tree of graphical elements in a backing store (e.g., backing store 208). The children property
15 412 facilitates establishing/maintaining a hierarchy reflecting nested relationships with regard to presenters in a view. In an embodiment of the invention, the array entries reference child proxy objects. The child proxy objects are wrapper objects for child presenter objects that are nested within the display space allocated to a parent presenter object. The child proxy object wrapper meets encapsulation and security needs (e.g., a
20 child presenter limits access by a parent presenter to its contents).

The clip property 414, as explained above, controls whether the contents of a presenter will be confined, by clipping, to the layout size of a presenter as specified by the layout size field 460.

A set of properties designate the space allocated to content within the space of a
25 presenter. In addition to the border property 404, a padding property 416 specifies an amount of space inside the border that is left unoccupied by content. A content height property 418 specifies a value indicating the height of content within a presenter – the height of the portion of the presenter that remains after accounting for the values specified in the border property 404 and the padding property 416. A content width
30 property 420 specifies a value indicating the width allocated to content within a presenter after accounting for the border property 404 and padding property 416 dimensions. The

content height property 418 and content width property 420, in combination, specify the area of a presenter that is potentially occupied in a presenter by content.

5 A height property 422 and a width property 424 specify the actual height and width of a presenter, and these two values include the content, padding and border dimensions. A default height property 422 and a default width property 424 specify default values assignable to the presenter when the height property 422 and width property 424 are not provided for a specific presenter object. A maximum width property 430 and minimum width property 432 specify bounds for the value assigned to the width property 424.

10 Yet another dimensional property for presenters is a margin property 434. The margin property 434 specifies a buffer space outside the presenter's border. The margin property 434 is specified by a child presenter and is accessed by its parent presenter when arranging a layout comprising a hierarchy of nested presenters.

15 A visibility property 436 specifies the visibility state of the presenter. The visibility property 436 is referenced to determine how (or whether) a graphical image corresponding to the presenter and its children is rendered. In an embodiment of the invention potential visibility states include: visible, collapsed, and hidden.

20 A number of presenter properties are utilized to link presenters to other entities in the layout/presentation management system. A presenter property 438 specifies the type of presenter that is used to layout and render the graphical element. The presenter type is initially set on a graphical element, and the presenter system 200 reads the presenter property 438 from the graphical element and instantiates a presenter object corresponding to the present type identified by the presenter property 438. The presenter property 438 is passed to an instantiator that, in turn, creates an instance of the identified presenter type.

25 An IsMainPresenter property 440 specifies whether the presenter is the only presenter or the primary (e.g., last) presenter in a set of chained presenters associated with a graphical element object. Referring to FIG. 3, presenter P4 is the main presenter for the chain including B4 and P4. In an embodiment of the invention, the IsMainPresenter property 440 is a Boolean type property.

30 An element owner property 442 identifies the graphical element for which the presenter was instantiated. Referring again to FIG. 3, the ghost lines connecting the

presenters (Px) to corresponding graphical elements (Ex) represent the associations designated by the content of the element owner property 442. A layout context property 444 specifies a global context that facilitates inter-presenter messaging during layout processing. The values stored in the layout context property 444 enable presenters to
5 pass messages independently of the presenter system 200. A notification handler property 446 specifies a reference to a notification handler for the presenter. A notification handler type property 448 specifies a type of a notification handler referenced by the notification handler property 446.

A dirtiness property 450 specifies whether a particular presenter has been changed
10 and therefore requires update processing. The presenter system 200 and associated components of the layout management architecture support incremental processing of a layout. The dirtiness property 450 maintained by each presenter indicates on a presenter-by-presenter basis whether a presenter needs to be updated.

With continued reference to FIG. 4a, an exemplary set of fields of the presenter
15 base class are summarized. In an embodiment of the invention, the fields are Common Language Runtime (CLR) fields (similar to well known member variables in the C++ classes). In an embodiment of the invention, the properties described above are similar to fields. However, properties on the graphical elements (e.g., graphical elements 206) are owned and managed by the backing store (e.g., backing store 208). When a property
20 changes, the backing store notifies the presentation engine of the change. The presentation engine (e.g., presentation engine 212 for presenter system 200) in turn notifies the appropriate notification handlers 224. Thus, in the case of a property, a change notification is received via one of the aforementioned notification handlers. Fields, on the other hand, are not managed by the backing store 208 (and are instead
25 managed for example by the CLR). Hence, no change notifications are fired to the presentation engine when a field changes.

The layout size field 460 maintains dimensions of a presenter's allocated layout space. The layout size field 460 is utilized to arrange neighboring presenters in a page/user interface layout. An affects layout field 462 is a collection of properties (e.g.,
30 border, padding, etc.) of a presenter that, when changed, affect the layout of the presenter. An affects parent layout field 464 is a collection of properties (e.g., margin, visibility,

etc.) of a presenter that, when changed for an element (e.g., Element A), dirties (i.e., potentially affects) the layout of a first parent of the element (Element A) having a specified presenter. An effects render field 466 is a notification mechanism that triggers invalidation and re-rendering of the presenter.

5 Referring now to the methods of the presenter base class 206 listed in FIG. 4b, an OnUpdate method 470 performs measuring and positioning tasks on a presenter for an element. Presenter authors overwrite a base class version of the OnUpdate method 470 to create/provide customized layout sizing/positioning behavior for an element's corresponding view.

10 A first default action of the OnUpdate method 470 is sizing the presenter. The OnUpdate method 470 receives presenter size dimensions in a passed BoxSizeInfo parameter. The OnUpdate method 470 sets the dimensions specified in the layout size field 460 according to the specified BoxSizeInfo and default width property 428 and default height property 426 values. Additional sources of sizing/positioning information
15 for a presenter are incorporated into the disclosed presenter-based layout/presentation processing architecture, and such sources are utilized in alternative embodiments of the invention.

A second default task of the OnUpdate method 470 is positioning child presenters within the local dimension space of the presenter. The base class version of the
20 OnUpdate method 470 iteratively calls the OnUpdate method 470 on children presenters identified in the children property 412 to discover their dimensions and children. Upon discovering and measuring the child presenters, the children are positioned within the parent presenter by invoking a transform method (discussed herein below) on the individual children. The iterative calls to the OnUpdate method 470 on children
25 presenters end when no further children are specified on a presenter. In addition to registering its size in the layout size field 460 and iteratively calling/positioning children presenters, the OnUpdate method 470 returns a value to the caller indicating whether re-rendering is desired in view of the call to the OnUpdate method 470.

A register atom method 471 registers a string name for the presenter for the layout
30 context 444 as well as any other appropriate lists of presenters including those maintained by extensions to the presenters. The register atom method 471 is called, for example, to

communicate with another presenter object in a non-standard manner (e.g., a manner not currently supported by the presenter application program interface). Presenters register an atom and then set the object to whom they wish to communicate using the atom. For example, a child presenter wishes to communicate a text descent to a parent presenter.

- 5 The child presenter registers an atom labeled/tagged "Descent" and receives an identification for the atom. The child computes the descent and sets the value in the atom using the identification. The parent presenter then uses the identification (that is also received by registering "Descent") to retrieve the stored information.

- An OnUpdateBoundingBox method 472 produces an updated bounding box
- 10 property 410 for a presenter. The OnUpdateBoundingBox method 472 is called if the OnUpdate method 470 returns a value (e.g., "true") indicating that re-rendering is required. The default implementation of the OnUpdateBoundingBox method 472, provided in the presenter base class, scans the children presenters identified in the children property 412 for the presenter and combines (i.e., performs a mathematical
- 15 union of rectangles for) transformed children's Bounding Box properties with the dimensions of the presenter stored within the layout size field 460. The base class implementation of the OnUpdateBoundingBox method 472 also accesses the clip property 414, and if it is "true," ignores children's bounding boxes. Instead the children presenters will be clipped to the layout size field 460 dimensions of their parent
- 20 presenter. Presenters that draw outside of their layout size field 460 dimensions overwrite this method. Such customized presenter classes call the base presenter class and add additional "ink area" in their overwrite code. In an embodiment of the invention, presenters are allowed to estimate an "ink area" to be bigger than actually required. If the bounding box property 410 area is smaller than the area needed to draw the presenter,
- 25 then the drawing is clipped based upon the bounding box property 410 dimensions specified for the presenter.

- An OnRender method 474 is invoked upon presenters that request/require re-rendering. The OnRender method 474 is the primary call on a presenter to re-render the presenter and any of its children presenters. The OnRender method 474 is preferably
- 30 invoked after the OnUpdate method 470 is invoked on all "dirty" presenters in a layout (e.g., a scene). In an embodiment of the invention, the OnRender method 474 is invoked

on presenter objects based upon whether the OnUpdate method 470 call on a presenter object returned an indication that the presenter requests re-rendering. In an embodiment of the invention, the base class OnRender method 474 performs device-independent presentation/layout arrangement operations on a presenter. Other components, including the graphics subsystem and graphics device drivers 234, render the presenter's view in a device-specific format. The customized derivations of the OnRender method 474 include calls to graphics generation components of the computer system to render graphical document/user interface output.

In an embodiment of the invention, a base class version of the OnRender method 474 renders the contents of a presenter including any children in three layout processing stages. During an OnRenderBeforeChildren stage, the presenter object invokes operations designated for processing before processing any of its child presenters. In an embodiment of the invention, the base class implementation of the OnRender method 474 provides a hook for customization and does not specify any preprocessing operations during the OnRenderBeforeChildren stage. However, an exemplary customized replacement for this default behavior is establishing/drawing a custom background for the presenter's display area.

The base class version of the OnRender method 474, during a children rendering stage, accesses a presenter's children presenters identified in the children property 412. The OnRender method 474, is invoked in turn on each of the children presenters (to the extent requested by the children when the OnUpdate method 470 was performed). Thus, a top-level call to the OnRender method 474 cascades down to children presenters, and their children, until the bottom of a tree is reached (i.e., the presenter has no children). When the top-level call is complete, all the children presenters in a tree that requested re-rendering have been re-rendered.

During an OnRenderAfterChildren stage, the presenter object invokes operations that are to be executed after calling the OnRender method 474 on the children presenters identified in the children property 412. The base class version of the OnRender method 474 provides a hook for customization and does not specify any post-processing operations during the OnRenderAfterChildren stage. However, customization examples include annotations, adornments, etc.

If a presenter does not re-render anything other than its children, then there is no need to overwrite the base class version of the OnRender method 474. However, if anything is to be drawn by the presenter itself, then the base class for the OnRender method 474 is overwritten to render graphical data within the space assigned the presenter. The replacement code includes, by way of example, calls to the applications 230 and/or the graphics subsystem and graphics device drivers. A chained presenter is in the tree of presenters for a particular view. If the chained presenter requires re-rendering, then its OnRender method will be invoked by the presenter system in a manner similar to OnRender for a main presenter.

10 In the context of this application a "hit-test" refers to a process for identifying which presenter falls under a particular grid point of a graphical user interface grid space. For example, a hit-test is used to locate where a mouse pointer is located (e.g., over which presenter it sits) when a user selects one of the mouse buttons. An OnHitTestDetail method 476 returns data stored within a HitTestDetail structure for a
15 presenter. A customized presenter class overwrites the base class implementation of the OnHitTestDetail method 476 to fill in the HitTestDetail structure to enable the presenter to return data other than the element and presenter that was hit, thereby providing, for example, more detailed information regarding the presenter such as a part of the presenter that was actually hit (e.g., a character position for text).

20 An OnCreateViewResult method 478 exposes parameters describing the calculated state of a presenter in a view result object. Users request the view result object for any element in the context of a designated view. The base class version of the OnCreateViewResult method 478 provides, through a view result object, the height, width, and top-left corner position of a presenter. If additional/alternative information
25 regarding a presenter's state is desired, such as how many lines of text are contained in the presenter, then the base class version of the OnCreateViewResult method 478 is overridden to provide the desired information through the view result object.

A hit test method 480 initiates a hit test upon the sub-tree rooted at a specified presenter. Coordinates of a point (in the local coordinates of the presenter) are passed to
30 the hit test method 480 upon which the hit test is to be performed. The hit test method 480 returns hit test data containing the results of the hit test for the designated point and

presenter. The results contain all visuals hit by the point. A visual is a base object (e.g., a line) from which a presenter derives a display image. A visual is a graphics object having the capability to actually draw an image on the graphical display output. In a presenter's OnRender call, several calls are issued to a visual's API to render, for
5 example, text, images, video, etc.

An OnGetBypassList method 482 provides a list of child presenters previously calculated for the identified presenter. The list of child presenters facilitates bypassing child presenters during layout calculation (e.g., comparing a new layout child with previously calculated layout children presenters and re-using a previously calculated one
10 if there is a match to one in the list). If a presenter discovers its children by referencing the children property 412 collection, it does not have to override this method. However, if the presenter uses a GetChildProxyForElement method 486 (described below), the OnGetBypassList method 482 is generally overridden.

In an application of a layout system embodying the present invention, in the case
15 where the GetChildProxyForElement method 486 is called, the OnGetBypassList method 482 is called before the OnUpdate method 470, and a presenter object creates and populates an array list with child proxy objects (described herein below with reference to FIG. 5) that it cached from a previous layout calculation. If a presenter object does not have a cache for storing the recalculated child proxy objects, all child presenters of the
20 presenter object are recalculated on every call to the OnUpdate method 470 on the presenter.

A calculate bounds method 484 calculates and returns the bounds of a sub-graph rooted at an identified visual.

A child proxy object, as explained above, is a wrapper object that acts as an
25 intermediary between a parent presenter and a child presenter for a particular graphical element. The child proxy objects for the child presenters of a particular presenter object are referenced as a collection within the children property 412 of the presenter object. The GetChildProxyForElement method 486 is a method on the parent presenter object that creates a child proxy object and presenter object for a specified element. The
30 method, returns "null" if the specified element does not have a presenter.

An OnBeforeBypass method 488 is called when a presenter system 200 wants to "bypass" a particular presenter (i.e., to skip a call to the presenter's OnUpdate method 470). Such bypass occurs, for example, when the presenter is not dirty and the OnUpdate method 470 call input presenter dimension parameters (BoxSizeInfo) have not changed
 5 from the last OnUpdate method 470 call to the presenter. The OnBeforeBypass method 488 is a performance enhancing mechanism enabling presenters, that have their own custom collection of child proxy objects (that that use the GetChildProxyForElement method 486), to reuse the child proxy objects of their children when an update is performed on a layout (e.g., when only a small portion of a layout is affected by a
 10 change).

An OnDisconnectChildren method 490 removes all children presenter objects from a set of children presenters referenced in the children property 412 for a specified presenter. If the default children property 412 is overridden for a presenter class derived from the presenter base class 216, then the OnDisconnectChildren method 490 is also
 15 overridden to ensure proper disposal of children presenters.

A Min/Max pass is a special sizing pass used by partitioning presenters (e.g., DockPresenter) when sizing children to content. Knowing the minimum and maximum widths of children allow a parent to intelligently distribute space among multiple children sharing a particular width dimension. Without the min/max information provided by the
 20 MinMaxPass function, a partitioning presenter would require multiple passes to intelligently size to content. An OnMinWidth method 492 is called after a Min/Max pass. The OnMinWidth method 492 collects a MinWidth value that is a byproduct of a Max Width calculation. The base class version of the OnMinWidth method 492 issues an "I don't care" message which invokes a calculation pass with a ProposedSize =
 25 (0,infinity) and both directions "NotFixed."

A presenter method 494 is a constructor method for a presenter instance. The presenter method 494 has no parameters. The presenter method 494 is invoked when the presenter is instantiated by the presenter system 200. The presenter method 494 sets the presenter instance up to enable invoking other methods (e.g., OnUpdate) on the presenter.

30 A QueueLayoutTask method 496 adds a specified layout task, to be performed with regard to the presenter object, to a queue of layout tasks to be executed by the

presenter system 200. When the specified layout task reaches the front of the queue, the presenter system 200 calls an OnLayoutTask method 604 (described herein below) on the notification handler for the presenter.

5 An OnQueryValue method 498 returns a value corresponding to a specified calculated value for the presenter.

Having described the presenter base class 216, attention is directed to **FIG. 5** that provides a summary of the properties and methods of an exemplary embodiment of the child proxy class 222. As previously explained, in an embodiment of the invention, the child proxy objects are wrappers for presenter objects that are children of a parent presenter object, and support limiting access by the parent to the resources of the child presenter objects. All communications (requests/responses) between a parent presenter object and a child presenter object are routed through the child proxy for the child presenter. The child proxy object holds a reference to the child presenter and routes
15 calls/requests by the parent presenter to the child presenter.

In an embodiment of the presenter system 200, the child proxy class 222 includes an element owner property 500 that specifies the element in the graphical elements 206 with which the child proxy's referenced presenter is associated.

A transform property 502 positions the child presenter object (and its associated
20 visuals) inside its parent presenter object. A transform is a standard graphics 3x3 transformation matrix on a visual object base class. When the transformation matrix is set up, a visual object, associated with the presenter's graphical element, places its child at an offset specified by the transformation matrix.

A bounding box property 504 specifies the rendering bounds of a child presenter.
25 The rendering bounds of the child presenter can thus differ from the layout size of the child presenter.

A dirtiness property 506 specifies the dirtiness state of the child proxy's presenter. In an embodiment of the presenter system 200, the dirty state is either: clean, dirty bystander, or dirty. A clean state indicates that no changes have occurred to the child
30 presenter since the last update of the view. A dirty state indicates that a change has occurred to the child presenter since the last update of the view. A dirty bystander state

indicates that at least one of the child presenter's own children is dirty, but the child presenter itself is not dirty. Such states enable parent presenters to optimize update operations.

5 A QueryDefaultSizeInfo method 510 initializes the dimensions of a child presenter from default values supplied by the child presenter's parent presenter.

10 An Update method 520 requests a child proxy to invoke the OnUpdate method 470 on the child presenter object. The parent presenter calls the Update method 520 on the child proxy when the parent presenter seeks calculation of the child presenter's layout. The Update method 520 call includes two passed parameters. Box size information specifies the dimensions of the box within which the child is to calculate a layout, and, if pagination is activated, the parent presenter passes page information in a page descriptor parameter (See FIG. 10 described herein below). The Update method 520 thus provides a layer between a caller and a child presenter object that enables screening/filtering of such requests to a child presenter object. The Update method 520 provides a way to
15 bypass calls normally routed to the OnUpdate method on the child presenter. For example, the Update method 520 call is invoked in cases where a child presenter has not changed and the input parameters (box size information and page description) are unchanged.

20 An attach method 530, issued by a parent presenter to a child proxy, invokes an OnRender command on a child presenter associated with the child proxy.

A query value method 540 provides a presenter value corresponding to a calculated value type (property) identified in the method call. The query value method 540 call is delegated to the corresponding OnQueryValue method 498 on the child
25 presenter object.

A query layout size method 550 provides calculated layout size parameters. The query layout size method 550 call is delegated to the corresponding OnQueryValue method 496 on the child presenter object. A compute MinMax method 560 provides a minimum and maximum width of a child presenter.

30

Referring to FIG. 6, the set of base classes associated with the presenter system 200 include a notification handler base class. As explained previously herein above, the notification handler objects receive all notifications regarding changes potentially effecting a presenter associated with a particular view, accumulate dirtiness information, and facilitate determining whether an associated presenter object requires updating. Both methods on the notification handler base class receive, as a parameter, an object including a method that provides relevant information from the element in the graphical elements (e.g., graphical elements 206) with which the notification handler is associated.

The notification handler base class includes an OnNotify method 600 that provides notification of dirtiness of a particular associated presenter object. The OnNotify method 600 potentially includes additional, customized derivative notification tasks (by overriding the default action of providing a dirtiness property value) that are executed in response to receiving a notification. The OnNotify method 600 receives as an input parameter information about a change that resulted in a notification. The OnNotify method returns a Boolean value indicating whether or not the notification handler can process the change by itself (or whether it is necessary to notify the notification handler for an ancestor presenter in the presenter tree for a view). The method also returns a dirtiness value indicating whether the presenter is "clean", "dirty", or a "dirty bystander".

An OnLayoutTask method 602 executes any tasks previously queued by an associated presenter through the QueueLayoutTask 496.

Referring to FIG. 7, the set of base classes associated with the presenter system 200 further includes a notification handler site class. The notification handler site class is a site object passed to a notification handler object that enables the notification handler to invoke notification services supported by the presenter system 200.

The notification handler site class includes an element owner property 700 specifying the graphical element object with which the notification handler site object is associated.

A register atom method 702 registers a provided string name within the layout context 444 for a notification handler. The functionality of the register atom method 702

is similar to register atom method 471 described herein above with reference to the presenter base class. The register atom method 702 is thus a means for communicating requests by a notification handler to other objects (e.g., presenter objects).

5 A create notification method 704 creates a new presenter notification object and initializes its fields based upon passed parameters. The passed parameters include an identification of an atom type (indicating the presenter notification object type) and notification data.

A notify descendants method 706 invokes the OnNotify method 600 on the notification handlers associated with the descendants of a presenter, with which the notification handler site is associated, to pass a specified presenter notification. The notify descendants method 706 performs the function of notifying descendant presenters of any changes to a parent presenter (e.g., re-sizing). Similarly, a notify ancestors method 708 invokes the OnNotify method 600 on the notification handlers associated with the ancestors of a presenter, with which the notification handler site is associated, to pass a specified presenter notification. Ancestors are determined by traversing the graphical element tree (see, FIG. 3) in the backing store 208. The graphical elements represented in the backing store include a property that specifies the parent (immediate ancestor) of the graphical element. A notify all method 710 invokes the OnNotify method 600 on the notification handler objects for all presenters in a presenter tree owned by a view to pass a specified notification. Finally, a notify self method 712 provides a means for a presenter to invoke the OnNotify method 600 to mark itself dirty.

Turning to FIG. 8, an exemplary view object class is depicted from which the view 202 is instantiated. View objects possess an instance of the presenter system (e.g., presenter system 200) and are the visual root of all visuals created within the created presenter system instance. The view object initiates all calculations on the presenters within the view. A view method 800 is a constructor method that is used to create a new view object. The view receives as a passed parameter a reference to an element that is the root graphical element of the view. A RootElement property 802 returns a reference to a root graphical element for the view (that was established through the above-described view method 800).

A DoLayout method 804 creates a new presenter tree for the view and returns a reference to the new tree. The DoLayout method 804 receives as input parameters: a proposed height and width for the view rectangle as well as two Boolean values (fixedwidth and fixedheight) that establish whether the proposed dimensions can be
 5 altered during the DoLayout method 804 execution.

A ViewSize property 806 sets the dimensions of the view. The ViewSize property 806, a write-only property, is utilized when the view object is hosted within a window, and the ViewSize property 806 represents an initial size from which a layout operation is performed.

10 A GetViewResult method 808 receives a graphical element reference as an input, and the GetViewResult method 808 returns the resulting view for the graphical element. A value of null is returned in the event that the graphical element is not located within the set of graphical elements for the view.

A HitTest method 810 performs a hit test at a passed point within a coordinate
 15 system of the view. The HitTest method 810 returns a first non-transparent hit test target. If no graphical element is hit, then the method returns "false."

A CreatePage method 812 receives as input parameters BoxSizeInfo (see, Fig. 9 described herein below) and PageDescriptor (see, Fig. 10 described herein below). The CreatePage method 812 creates a new page, based upon the input parameters, and returns
 20 the child proxy object corresponding to the root graphical element for the page.

Turning to FIG. 9, an exemplary BoxSizeInfo structure is depicted. The BoxSizeInfo structure specifies a set of dimensions for presenters and their child presenters within a view, and how those dimensions should be treated by the recipient. A
 25 proposed size property 900 sets forth the proposed size for a child presenter within the view. Initially, the value of the proposed size property 900 comes from properties for a child graphical element in a backing store. It can be modified by a parent presenter before passing to a child. A parent size property 902 is a proposed size for a parent presenter. The parent size property 902 is used, by way of example, to calculate
 30 dimensions, from percent values of the parent, for child presenters. The BoxSizeInfo structure also includes two Boolean value properties, FixedWidth 904 and FixedHeight

906 that instructs the recipient presenter whether the supplied dimension as an imposed (fixed) value, or whether the dimension value is merely a suggested height or width.

Turning to FIG. 10, parts of an exemplary page descriptor structure are depicted.

5 A page descriptor includes a PageSize property 1000. The PageSize property 1000 is a read only property specifying the size of a page. It is initialized when the page descriptor structure is created. A break record property 1002 is a structure including: a character position start that identifies the first character of the current presenter that is on a next page (with respect to the graphical element that is the root of a view that is being
10 paginated), a number of characters (counting back from the character position start) that can invalidate a previous page; and an IsDirty Boolean value that indicates whether a break record (covering a particular character range within a paginated graphical element) is invalidated by a change to the backing store.

The page descriptor structure also includes an AvailableSize property 1004. The
15 complete page may not be available to a presenter upon commencing an OnUpdate method on the presenter. The AvailableSize property 1004 indicates the space remaining on the page for a presenter. The AvailableSize property 1004 is initialized to a value for a page size.

20 Turning to FIG. 11, a set of methods are identified that are carried out by the presentation engine 212 of the presenter system 200 created by the view 202 (see, Fig. 2). A constructor method 1100 on the presentation engine 212 accesses the view 202 that contains a reference to the root graphical element for the view (provided by the graphical elements 206 tree. A DoLayout method 1102 receives a width and height as passed
25 parameters representing the dimensions of the view 202 and, in response, commences executing a layout process with regard to the contents of the graphical elements 206. An UpdateLayout method 1104, executed after executing the DoLayout at least once, commences applying incremental changes to a previously calculated layout for the view 202. A CreatePage method 1106 receives, as an input parameter, box size information
30 (BoxSizeInfo) that contains the dimensions of a page to be laid out, and a page descriptor containing information regarding how to start the particular page. Rather than lay out all

content under a root of the graphical elements 206 for the view 202, the CreatePage method 1106 lays out only a single page of interest.

An exemplary layout processing architecture has been described with reference to
5 **FIGs. 1-11.** The exemplary architecture includes separate view states (presenter objects) linked to element data states (graphical element objects). The exemplary architecture includes a presenter system 200 that facilitates extending the view layout/presentation capabilities through a presenter base class 216 from which new presenter object classes, embodying new layout/presentation functionality, are derived. The system and
10 architecture disclosed herein also include a notification mechanism linking elements to their corresponding presenter objects to selectively invoke recalculating portions of a view in response to changes. The illustrative object base classes are illustrative of many potential alternative ways of carrying out the extensible architecture embodying the present invention and should not be viewed as limiting the scope of the invention.

15

Having described a set of components making up an exemplary presentation/layout processing architecture embodying the present invention, attention is directed **FIG. 12** that depicts the general progression of processing changes to a graphically displayed element. Initially, at 1200 the backing store 208 changes. For
20 example, the application 204 executes an operation that results in a status and/or content change to one or more of the displayed components of a graphical user interface for the application. In response, at step 1202 a change notification, arising from a change to a graphical element in the backing store, is received by the presentation engine 212 of the presenter system 200.

25 By way of background, with regard to step 1204, the presenter system 200 maintains associations between elements and presenters in element-to-presenter information (EPI). Whenever a presenter is created for a graphical element, a reference to that presenter is created on the element and is stored in EPI maintained by the presenter system. If a graphical element has more than one presenter, then there will be
30 multiple references (one per presenter) in the graphical element's EPI. When a property changes on a graphical element, the following steps occur:

(a) The backing store 208 notices the change on the graphical element,

(b) The backing store 208 notifies the presentation engine 212 of the change, with reference to the graphical element that has experienced a change to one of its properties,

(c) The presentation engine 212 walks the EPI of the graphical element that
5 changed and determines the presenters that have been created for the graphical element,
and

(d) Once the presenter is determined, the presentation engine 212 obtains a reference to the presenter's corresponding notification handler (through the notification handler property 446 on the presenter).

10 Thus, in view of the above, at step 1204, the presentation engine 212 of the presenter system 200 determines the identity of a presenter to which the change notification applies. As explained above, the presenter system maintains a list of presenters that were instantiated for the element in its EPI. After determining a presenter to which the change notification applies (assuming for this example that only a single
15 presenter exists for the element), the presentation engine 212 of the presenter system obtains a reference to a notification handler for the presenter through the notification handler property 446 on the identified presenter.

Thereafter, during step 1206 the presentation engine 212 of the presenter system 200 forwards the change notification to the notification handler using the reference
20 obtained during step 1204. In an embodiment of the invention, step 1206 is carried out by invoking the OnNotify method 602 on a particular notification handler object.

After receiving the change notification from the presenter system, during step 1208 (e.g., during execution of the OnNotify method 602) the notification handler determines the actions, if any, required by the presenter in response to the changes,
25 during step 1200, to the graphical element in the backing store 208. In an embodiment of the invention, the notification handler determines what portion, if any, of a corresponding presenter requires updating in view of the received change notification. Such portions include, by way of example, the children of the presenter and the presenter itself.

Thereafter, during step 1210 the notification handler (e.g., OnNotify method 602)
30 returns a result that enables the corresponding presenter to determine and execute actions in view of the change notification received by the presenter system 200 during step 1202.

In an embodiment of the invention, the form of the returned result is notification handler/presenter-specific. In some cases, the returned result simply labels the presenter as "dirty" – thereby invoking a complete re-calculation of the corresponding presenter. In other cases, information is provided to selectively invoke only portions of the presenter's view re-calculation capabilities.

In an embodiment of the invention, the presentation engine 212 of the presenter system 200 receives the returned result from the notification handler for a presenter during step 1210. Thereafter, at step 1212 if the changes to the backing store did not raise a need to recalculate the presenter, then control passes to decision block 1216. However, if at step 1212 the returned result indicates that the presenter needs to be re-calculated, then control passes to step 1214 wherein the presenter system 200 adds the presenter to the set of "dirty" presenters that require re-calculation. In an exemplary embodiment of the invention, the presenter system sets the dirtiness property 450 on the corresponding presenter (indicating a need to re-calculate the presenter) and adds the presenter to a set of presenters that need to be re-calculated. The presenters also specify information describing the extent of changes that led to the setting of the dirtiness property 450. Control then passes to step 1216.

At step 1216, if a parent presenter does not exist for the presenter associated with the notification handler of present interest, then control passes to the End 1218. If a parent presenter does indeed exist, then control passes from step 1216 to step 1220 wherein a notification is created for (passed to) the parent presenter's notification handler (resulting in the notification handler for the parent processing the notification) as indicated by the dotted return line to step 1208.

Thus, the set of steps depicted in FIG. 12 enables the presenter system 200 to accumulate a list of the presenters that have been "dirtied" as a result of changes to the backing store 220 since last calculating a layout for a document/user interface.

Turning now to FIG. 13, a flowchart describes an exemplary process for re-calculating/re-rendering a document/user interface according to changes to presenters in a particular view (see, FIG. 12 described herein above). In an embodiment of the invention, the application 204 controls updates to its view 202 and invokes view update

procedures, such as the one summarized in FIG. 13, in response to events such as: a period expiring, a document print request, or a change in status of a graphical user interface.

Initially, during step 1300 the presenter system 200, and particular the
5 presentation engine 212 receives a call to the DoLayout method 1102 (described herein above with reference to the presentation engine 212 API description). In response to receiving the DoLayout method 1102 call by the application 204 the presenter system 200 re-calculates presenter objects contained within the view 204 to accommodate changes to elements in the backing store 208 since a last DoLayout method 1102 call.

10 In an embodiment of the present invention wherein the presenters for a view are hierarchically arranged (see, FIG. 3), the presenter system 200 initially determines the root presenter for the view in step 1302. In an embodiment of the invention, the root presenter is identified by the presenter property on the root element that is submitted to the view (e.g., view 204a) and hence the presenter system (e.g., presenter system 200a).
15 After determining the root presenter, control passes to step 1304 wherein the presenter system 200 commences traversing the tree of presenters for the view and re-calculating each dirty presenter.

During step 1304 the presenter system 200 invokes the OnUpdate method 470 on each dirty presenter in the view. The called OnUpdate method 470 is passed a height and
20 width of a rectangle within which the presenter is to re-calculate its layout. After re-calculating its layout, the presenter caches its result. Thus, upon completion of step 1304 the previously dirty presenters have been re-calculated and their results cached for easy access during subsequent re-rendering. The dirty bit is cleared on each of the called presenters. A set of steps performed during an exemplary OnUpdate method 470 call are
25 described herein below in greater detail with reference to FIG. 14.

There are a number of ways in which to clean dirty presenters during step 1304. However, in an embodiment of the invention, the presenter system 200, and in particular the presentation engine 212 UpdateLayout method 1104, leverages the hierarchical relationships between presenters in a view to simplify its role in completing step 1304. In
30 particular, during execution of the OnUpdate method 470 each called presenter iteratively calls (or alternatively requests the presenter system 200 to call) the OnUpdate method

470 on all children of the called presenter. Thus, the presenter system 200 starts the traversal of the presenter tree at the root presenter's OnUpdate method 470 (e.g., presenter P1 of FIG. 3) and supports subsequent calls to child presenters as the OnUpdate method 470 call traverses the branches of the hierarchical presenter tree to re-calculate dirty
5 presenters. The iterative calls to the OnUpdate method 470 on the children presenters ensures that all presenters in the view are traversed, and if the presenters are dirty they are re-calculated.

Upon completion, each OnUpdate method 470 called during step 1304 resets its dirtiness property 450 indicating that it has been re-calculated or "cleaned." Each
10 OnUpdate method 470 also returns a value to the caller indicating whether the presenter requires re-rendering (i.e., re-drawing). In an embodiment of the invention the returned value is a Boolean value. If "true", then re-rendering is required. If "false" is returned then re-rendering is not necessary. After processing each of the dirty presenters for a view during step 1304, control passes to a rendering step 1306 wherein presenters
15 identified in the list of presenters requiring re-rendering are processed.

During step 1306 the presenter system 200 invokes the OnRender method 474 on each presenter identified in the list of presenters requesting re-rendering. The operation of an exemplary embodiment of the OnRender method 474 is described with reference to FIG. 14. In the base class implementation of the OnRender method 474, the presenter
20 does not draw anything, and instead iteratively calls the OnRender method 474 on its children and/or chained presenters to fill its allocated space (e.g., rectangle). Customized versions of the OnRender method 474 render bitmaps through calls to the application 204 and/or the APIs of the graphics subsystem and graphics device drivers 234. After processing all the presenters requesting re-rendering during step 1306, control passes to
25 the End 1308, and control returns to the caller of the DoLayout method call 1102 on the presentation engine for the particular view.

Turning to FIG. 14, a flowchart summarizes the steps of the OnUpdate method 470 performed to re-calculate a presenter of a view. The OnUpdate method 470
30 calculates a new layout for a presenter, including any children presenters, and caches the results for access during a re-rendering stage. In the example provided herein below, the

OnUpdate method 470 call to each presenter passes a set of dimensions specifying the limits to the view space (e.g., rectangle) that can be occupied by the layout of the presenter. The set of dimensions specifies, for example, the height and width and whether those values can be modified during execution of the OnUpdate method 470 call.

- 5 However, the invention contemplates re-calculating presenters based upon any one or more of a variety of passed parameters, including view space dimensions, that affect a layout.

During step 1400, the presenter performs any needed sizing operations, arising from the passed parameters, before calling any child presenters (via child proxy objects) to perform their updates. The content of such work is dependent upon the design of a particular presenter type. Thereafter, the presenter calls upon the presenter system 200 to locate and update any children presenters of the current presenter. During the pre-processing stage, the presenter potentially detects the need to create a new page/column and creates a new child proxy/presenter to handle the new page/column for the view.

- 15 Next, during step 1402, the OnUpdate method 470 on the current parent presenter determines a next child presenter to call to update its layout. It is noted that this step, at least in a first instance requires the parent presenter to determine the existence of a child element and create a corresponding child proxy/presenter. In later iterations, the parent presenter can use the cached child proxies identified in its children field 412 to identify a next remaining (unprocessed) child proxy/presenter—i.e., a child presenter that has not yet been called during the current iteration of the OnUpdate method on the current parent presenter.

The parent presenter receives a response from the backing store or determines on its own from its children property 412 whether a child presenter remains to be processed, and at step 1404, if no unprocessed child remains to be updated, then control passes to step 1420 (described herein below). Otherwise, if the presenter does have an unprocessed child presenter, within its children property 412, that has not yet been updated, then control passes to step 1406.

- 30 At step 1406, the presenter calls the Update method 520 on the returned child proxy, that is interposed between the parent presenter and a current child presenter of interest, to obtain a height and width of a child presenter with which the child proxy is

associated. The child presenter is associated with a particular graphical element (specified in its element owner field 442).

At step 1410 the child proxy calls the OnUpdate method 470 on the child presenter with which the child proxy is associated. The OnUpdate method 470 calculates
5 an updated height and width of the child presenter (and whether re-rendering is required). It is noted that during step 1410 the OnUpdate method 470 is recursively invoked by child presenters until a called child presenter determines, at step 1402 that it has no children and returns to its calling parent presenter (through its child proxy). After invoking the OnUpdate method 470 on the child presenter, control passes to step 1414.

10 Thus, at step 1414 the calling presenter parent (from step 1406) receives a response to the Update method call 520 to the identified child proxy. The response includes the height and width of the child presenter as returned parameters. The response also indicates whether the called child presenter requires re-rendering. Next, at step 1416 the parent presenter positions the called child presenter within the parent presenter's
15 allocated layout space based upon a placement algorithm/policy. In an embodiment of the invention, during step 1416 the parent presenter places the child presenter within its layout based upon the height and width parameters of the child presenter received by the presenter during step 1414.

Control then returns to step 1402 where the OnUpdate method 470 of the parent
20 presenter continues to update any remaining child presenters (through their child proxy wrapper objects).

If at step 1404, no child (element or proxy/presenters) remain to be processed/updated, then control passes to step 1420. At step 1420 the OnUpdate method 470 of the called presenter performs customized layout functions including potentially
25 further calls to the OnUpdate method 470 on the children presenters to tune the layout for the presenter.

At step 1422, or potentially any point where new information is derived by the OnUpdate method 470, the results of the layout processing are cached for later use during a re-rendering stage of the graphical view processing.

After update processing is complete for the presenter, including all children of the called presenter, at step 1424 the presenter returns its dimensions to the caller (presenter system 200) as well as a value indicating whether the presenter needs to be re-rendered.

5 It will be appreciated by those skilled in the art that a new exemplary platform and exemplary methods have been described for managing layout/presentation processing of graphical document/user interface output within a computing environment including graphical output devices such as a graphical user interface display or a printer. In view of the many possible environments to which the principles of this invention may be applied
10 and the flexibility of designing and carrying out software utilities and tools, it should be recognized that the embodiments described herein are meant to be illustrative and should not be taken as limiting the scope of invention. Those skilled in the art to which the present invention applies will appreciate that the illustrated embodiments can be modified in arrangement and detail without departing from the spirit of the invention. Therefore;
15 the invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

WHAT IS CLAIMED IS:

1. A graphical output layout management system for arranging and maintaining graphically displayed output of programs executing on a computing system, the graphical output layout management system comprising:
 - 5 graphical elements containing data representing displayable content of a program;
 - presenters defining display states for the graphical elements, wherein a presenter of a particular type maintains a layout description for a corresponding graphical element; and
 - 10 a presenter system, including a presenter host interface comprising a method for preparing a layout for a view, according to a set of presenters associated with graphical elements contained within the view.
2. The graphical output layout management system of claim 1 wherein the presenter system further comprises a presenter base class, and wherein the set of
15 presenters are instantiated from presenter classes derived from the presenter base class.
3. The graphical output layout management system of claim 2 wherein the presenters include an update method for calculating layout properties for the presenter.
- 20 4. The graphical output layout management system of claim 3 wherein the presenters include a render method for generating instructions issued to a graphics output subsystem of the computer system.
5. The graphical output layout management system of claim 2 wherein the
25 presenter classes include predefined presenter classes and external presenter classes.
6. The graphical output layout management system of claim 1 wherein each of the presenters calculates a layout state for a corresponding graphical element based upon layout parameter values specified on the presenter.

30

7. The graphical output layout management system of claim 1 further comprising a set of notification handlers, each notification handler being associated with a particular one of the set of presenters in the view, and wherein a notification handler processes a change to a graphical element to determine whether to update a
5 corresponding presenter.

8. The graphical output layout management system of claim 7 wherein each presenter specifies a type of notification handler that provides change notifications to the particular presenter regarding an associated graphical element.
10

9. The graphical output layout management system of claim 8 wherein a notification handler determines a part of a corresponding presenter that requires updating in view of changes to an associated graphical element.

10. The graphical output layout management system of claim 7 wherein the presenter system registers, through the notification handlers, presenters that require updating based upon changes to corresponding graphical elements.
15

11. The graphical output layout management system of claim 1 wherein a graphical element instance of the graphical elements is associated with multiple presenters.
20

12. The graphical output layout management system of claim 11 wherein the multiple presenters with which the graphical element instance is associated are of a same type.
25

13. The graphical output layout management system of claim 12 wherein the multiple presenters correspond to distinct columns within a same view.

14. The graphical output layout management system of claim 12 wherein the multiple presenters correspond to distinct pages in a document view of a single graphical element.

5 15. The graphical output layout management system of claim 11 wherein a first presenter and a second presenter, of the multiple presenters associated with the graphical element instance, are associated with respective first and second distinct views.

10 16. The graphical output layout management system of claim 15 wherein the first and second distinct views are hosted by respective first and second presenter systems.

15 17. The graphical output layout management system of claim 1 wherein the view corresponds to a rectangular region within a graphical display space.

18. The graphical output layout management system of claim 11 wherein the multiple presenters associated with the graphical element instance are of different types.

20 19. The graphical output layout management system of claim 18 wherein one of the multiple presenters comprises an adornment presenter to a main presenter.

25 20. The graphical output layout management system of claim 1 wherein presenters within a view are hierarchically arranged such that child presenters are contained within a region specified by a corresponding parent presenter.

21. A presenter system for incorporation within a graphical output layout management system for processing a layout, on behalf of a program providing graphical elements containing data representing displayable content of the program, through presenters defining display states for the graphical elements, and wherein a presenter
5 maintains a layout description for a corresponding graphical element, and wherein the presenter system hosts and arranges the presenters associated with graphical elements within a view, the presenter system comprising:

a presenter base class from which presenter classes are derived, and wherein the presenters are instantiated from the presenter classes; and
10 a presenter host interface comprising a set of methods including at least a method for preparing a layout embodied in the set of presenters associated with graphical elements within the view.

22. The presenter system of claim 21 wherein the presenter base class
15 specifies a customizable update method, overridden by derived presenter classes, for calculating layout properties for a presenter.

23. The presenter system of claim 22 wherein the presenter base class
20 specifies a customizable render method, overridden by derived presenter classes, for generating instructions issued to a graphics output subsystem of the computer system.

24. The presenter system of claim 21 wherein the presenter system supports a set of presenter classes that includes predefined presenter classes and external presenter
25 classes.

25. The presenter system of claim 21 wherein the presenter base class specifies a method for calculating a layout state for a graphical element based upon layout parameter values specified on an associated presenter.

30 26. The presenter system of claim 21 further comprising a notification handler base class from which notification handler classes are derived, and wherein each

notification handler instantiated from a notification handler class is associated with a particular presenter in a view, and wherein a notification handler processes a change to a graphical element to determine whether to update a corresponding presenter.

5 27. The presenter system of claim 26 wherein the base presenter class includes a placeholder for a presenter to specify a type of notification handler that provides change notifications to the particular presenter regarding an associated graphical element.

 28. The presenter system of claim 26 wherein the presenter system registers,
10 through notification handlers, presenters that require updating based upon changes to corresponding graphical elements.

 29. The presenter system of claim 21 wherein the presenter system is
associated with a view characterized by a root graphical element and an assigned area
15 within which to display the root graphical element and its child graphical elements.

 30. The presenter system of claim 29 wherein the view corresponds to a rectangular region within a graphical display space.

20 31. The presenter system of claim 21 wherein the presenter host interface includes a constructor for instantiating the presenter system for laying out a display for a view comprising a set of graphical elements.

 32. The presenter system of claim 31 wherein the view is specified at least in
25 part by a root graphical element passed to the constructor.

 33. The presenter system of claim 21 wherein the presenter system supports chaining multiple presenters such that a first presenter provides an adornment for a second presenter associated with a graphical element.

30

34. The presenter system of claim 21 wherein the presenter base class includes a placeholder for specifying child presenters, thereby supporting creation of a hierarchical tree structure of presenters within a view such that child presenters are contained within a region specified by a corresponding parent presenter.

5

35. A method for processing a layout, on behalf of a program providing graphical elements containing data representing displayable content of the program, through presenters defining display states for the graphical elements, and wherein a presenter maintains a layout description for a corresponding graphical element, and
10 wherein the presenter system hosts and arranges the presenters associated with graphical elements within a view, the method comprising:

providing a presenter system comprising a presenter base class and a presenter host interface comprising a method for creating a layout embodied in a set of presenters associated with graphical elements within a view;

15 receiving a request, by the presenter host interface, to generate a layout for a set of graphical elements within a view; and

instantiating, for the set of graphical elements, corresponding presenters from a set of presenter classes derived from the presenter base class and invoking a method on each instantiated presenter to calculate a layout state for a graphical element

20 corresponding to the presenter.

36. The method of claim 35 further comprising the step of:

providing a notification handler base class specifying an interface for a set of notification handlers that facilitate incremental update to a layout based upon changes to
25 corresponding graphical elements.

37. The method of claim 35 further comprising organizing the presenters for the view in the form of a hierarchical tree.

38. The method of claim 35 wherein an instance of the presenter system is created in response to a call to create a layout for a view, thereby creating a distinct presenter system, and corresponding presenters, for each view.

5 39. The method of claim 35 wherein the presenter base class specifies an update method for calculating layout properties for a presenter instance associated with a particular graphical element.

10 40. The method of claim 35 wherein the presenter base class specifies a render method for generating instructions issued to a graphics output subsystem.

41. The method of claim 35 wherein the presenter system supports derivation of predefined presenter classes and external presenter classes.

15 42. The method of claim 35 wherein the presenter classes correspond to differing types of presenters that layout different types of graphical element content.

20 43. The method of claim 35 wherein presenters within a view are hierarchically arranged such that child presenters are contained within a region specified by a corresponding parent presenter.

44. A computer-readable medium including computer-executable instructions for facilitating processing a layout, on behalf of a program providing graphical elements containing data representing displayable content of the program, through presenters defining display states for the graphical elements, and wherein a presenter maintains a layout description for a corresponding graphical element, and wherein the presenter system hosts and arranges the presenters associated with graphical elements within a view, the computer-executable instructions facilitating performing a method comprising:

5 providing a presenter system comprising a presenter base class and a presenter host interface comprising a method for creating a layout embodied in a set of presenters associated with graphical elements within a view;

10 receiving a request, by the presenter host interface, to generate a layout for a set of graphical elements within a view; and

instantiating, for the set of graphical elements, corresponding presenters from a set of presenter classes derived from the presenter base class and invoking a method on

15 each instantiated presenter to calculate a layout state for a graphical element corresponding to the presenter.

45. The computer-readable medium of claim 44 further comprising computer-executable instructions for:

20 organizing the presenters for the view in the form of a hierarchical tree.

46. The computer-readable medium of claim 44 wherein computer-executable instructions facilitate creating an instance of the presenter system in response to a call to create a layout for a view, thereby creating a distinct presenter system, and corresponding

25 presenters, for each view.

47. The computer-readable medium of claim 44 wherein the presenter base class specifies an update method for calculating layout properties for a presenter instance associated with a particular graphical element.

30

48. The computer-readable medium of claim 44 wherein the presenter base class specifies a render method for generating instructions issued to a graphics output subsystem.

5 49. The computer-readable medium of claim 44 wherein the presenter system supports derivation of predefined presenter classes and external presenter classes.

10 50. The computer-readable medium of claim 44 wherein the presenter classes correspond to differing types of presenters that layout different types of graphical element content.

15 51. The computer-readable medium of claim 44 wherein the presenter base class specifies a placeholder for child presenters, thereby facilitating hierarchically laying out presenters within a view such that child presenters are contained within a region specified by a corresponding parent presenter.

48. The computer-readable medium of claim 44 wherein the presenter base class specifies a render method for generating instructions issued to a graphics output subsystem.

5 49. The computer-readable medium of claim 44 wherein the presenter system supports derivation of predefined presenter classes and external presenter classes.

50. The computer-readable medium of claim 44 wherein the presenter classes correspond to differing types of presenters that layout different types of graphical element
10 content.

51. The computer-readable medium of claim 44 wherein the presenter base class specifies a placeholder for child presenters, thereby facilitating hierarchically laying out presenters within a view such that child presenters are contained within a region
15 specified by a corresponding parent presenter.

ABSTRACT

A presenter system framework and methods performed by such framework are described for hosting a set of presenters that facilitate maintaining layouts for application views assigned a set of graphical elements. The presenter system provides a base
5 presenter class and a set of interface methods, performed by a presentation engine, for creating and integrating an extensible set of presenter classes for handling various graphical element data types during a layout operation within a specified view. The presenter system enables application user interface developers to implement complex display layout operations through calls to the presentation engine. Examples of such
10 complex display layout operations include: pagination, partial calculation, incremental calculation, multiple tries, chaining of layout features/operations.

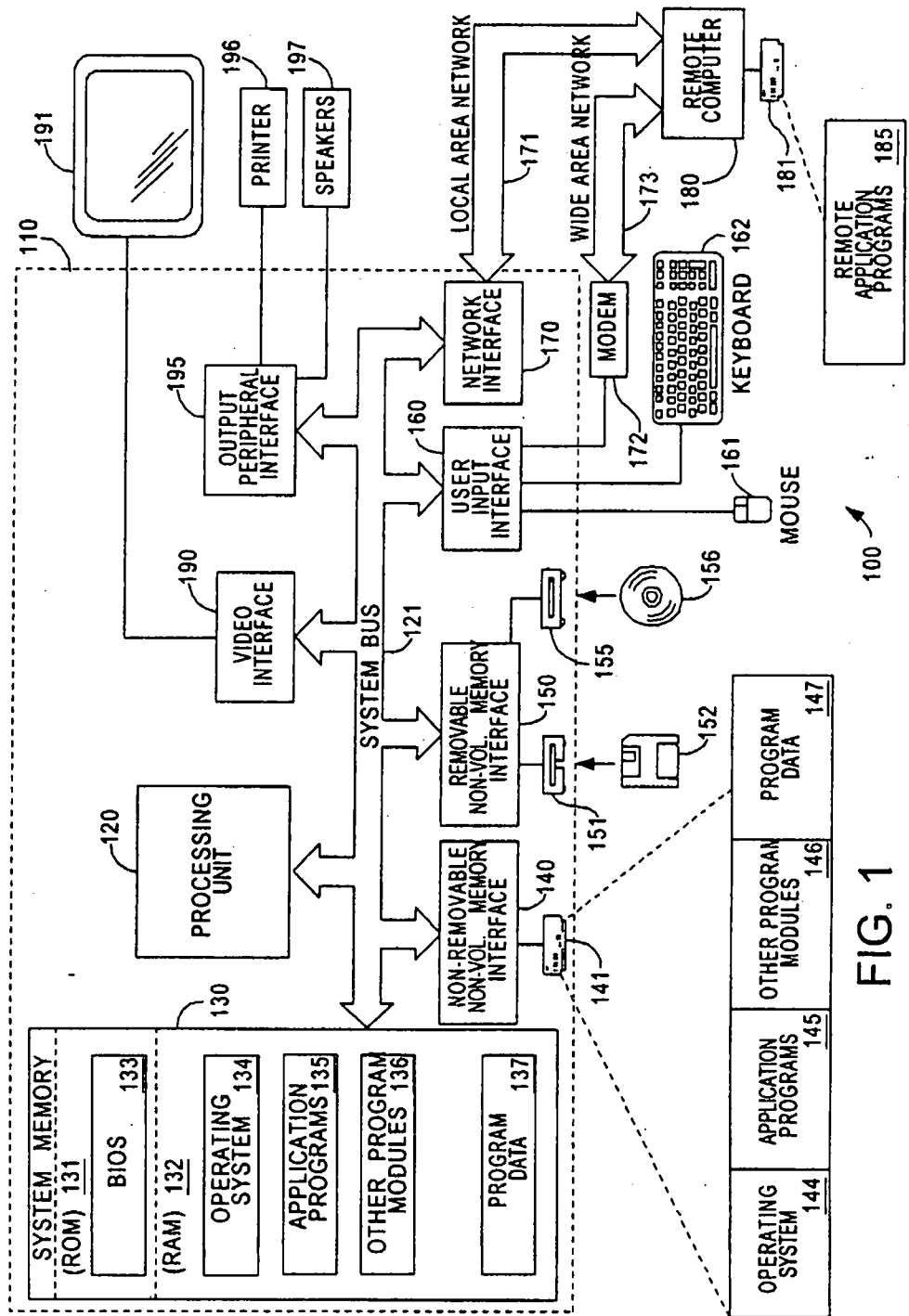


FIG. 1

244

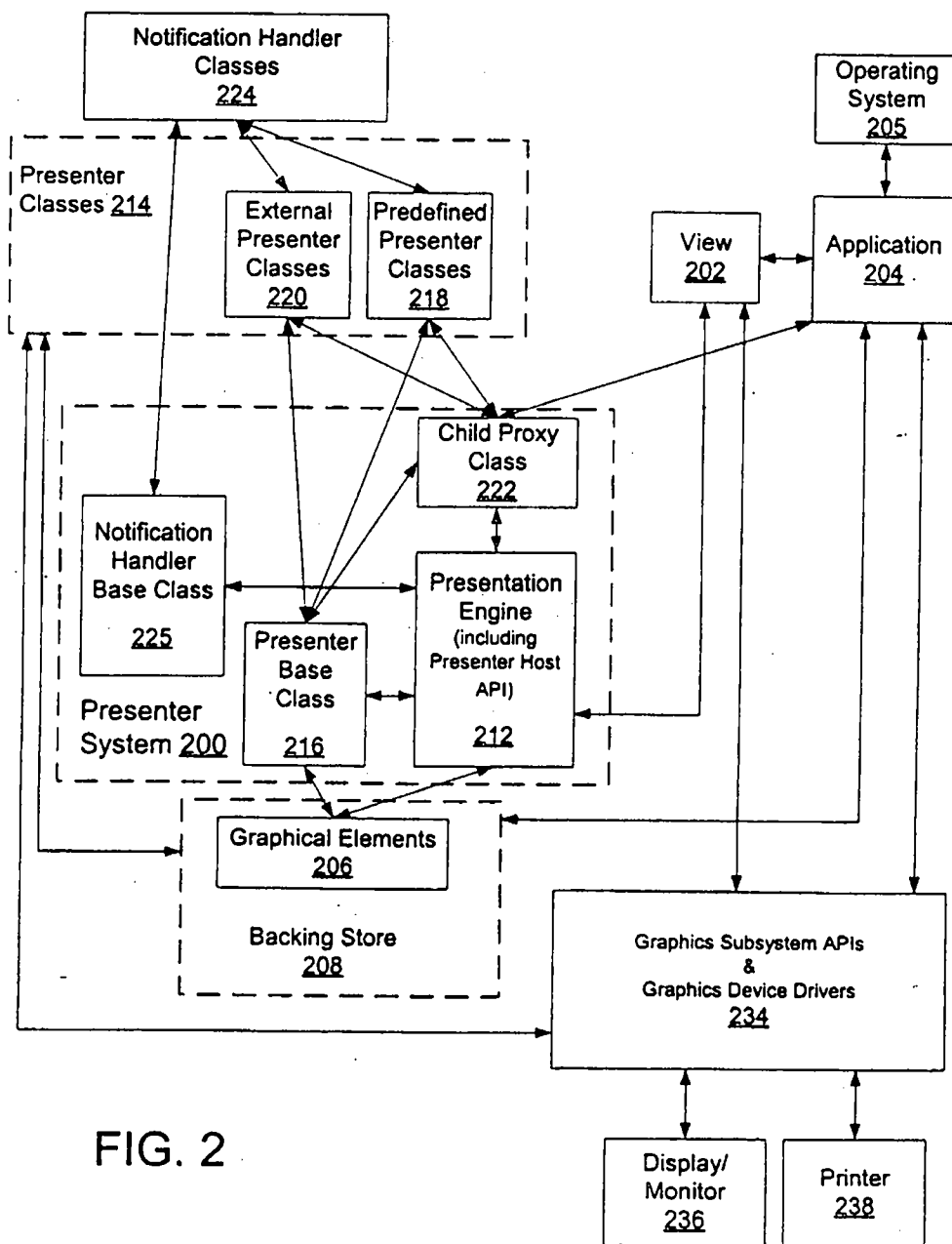


FIG. 2

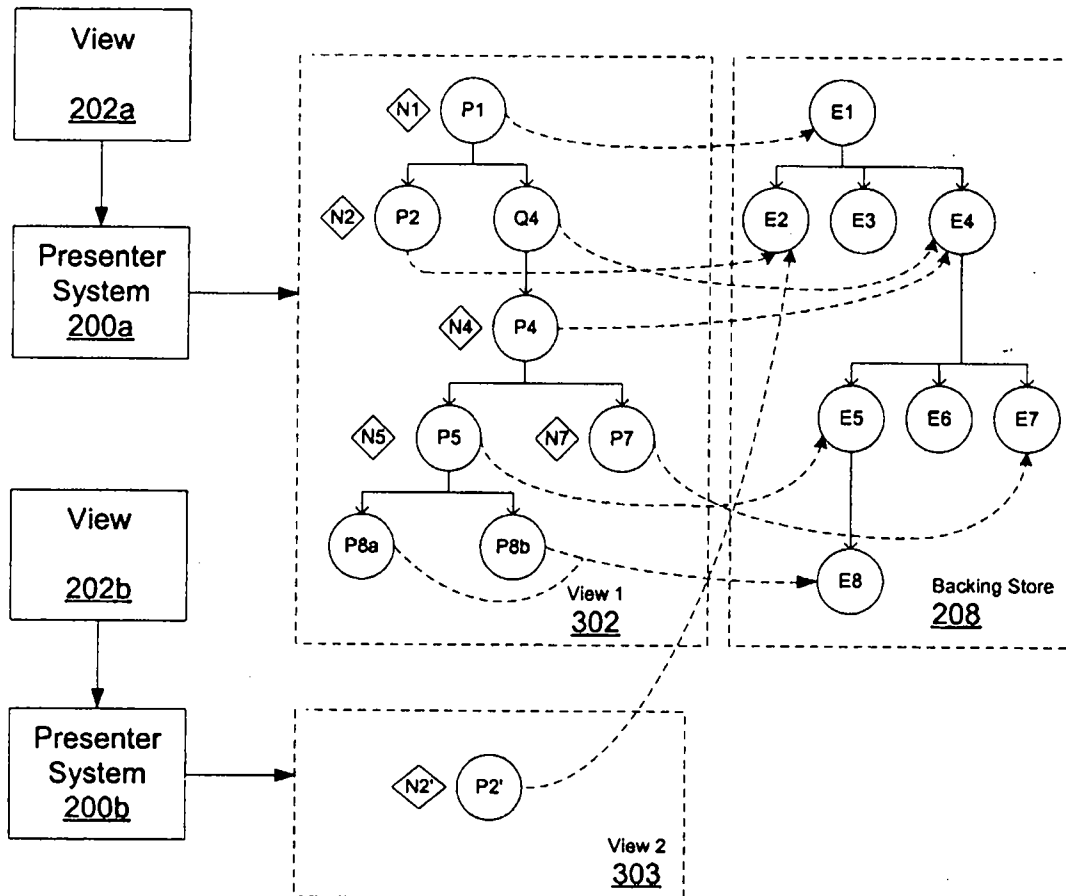


FIG. 3

Presenter Base Class Properties/Fields	
400	Background
402	Background Opacity
404	Border
406	Border Color
408	Border Style
410	Bounding Box
412	Children
414	Clip
416	Padding
418	Content Height
420	Content Width
422	Height
424	Width
426	Default Height
428	Default Width
430	Max Width
432	Min Width
434	Margin
436	Visibility
438	Presenter
440	IsMainPresenter
442	Element Owner
444	Layout Context
446	Notification Handler
448	Notification Handler Type
450	Dirtiness
460	Layout Size
462	Affects Layout
464	Affects Parent Layout
466	Affects Render

FIG. 4a

5/10

Presenter Base Class Methods

470	OnUpdate
471	Register Atom
472	OnUpdateBoundingBox
474	OnRender
476	OnHitTestDetail
478	OnCreateViewResult
480	HitTest
482	OnGetBypassList
484	CalculateBounds
486	GetChildProxyForElement
488	OnBeforeBypass
490	OnDisconnectChildren
492	OnMinWidth
494	Presenter
496	QueueLayoutTask
498	OnQueryValue

FIG. 4b

Child Proxy Class	
500	Element Owner
502	Transform
504	Bounding Box
506	Dirtiness
510	QueryDefaultSizeInfo
520	Update
530	Attach
540	Query Value
550	Query Layout Size
560	Compute MinMax

FIG. 5

Notification Handler Interface	
600	OnNotify
602	OnLayoutTask

FIG. 6

Notification Handler Site Class	
700	Element Owner
702	Register Atom
704	Create Notification
706	Notify Descendants
708	NotifyAncestors
710	NotifyAll
712	NotifySelf

FIG. 7

View Class	
800	View
802	Root Element
804	DoLayout
806	View Size
808	GetViewResult
810	HitTest
812	Create Page

FIG. 8

BoxSizeInfo	
900	Proposed Size
902	Parent Size
904	FixedWidth
906	FixedHeight

FIG. 9

Page Descriptor	
1000	PageSize
1002	Break Record
1004	AvailableSize

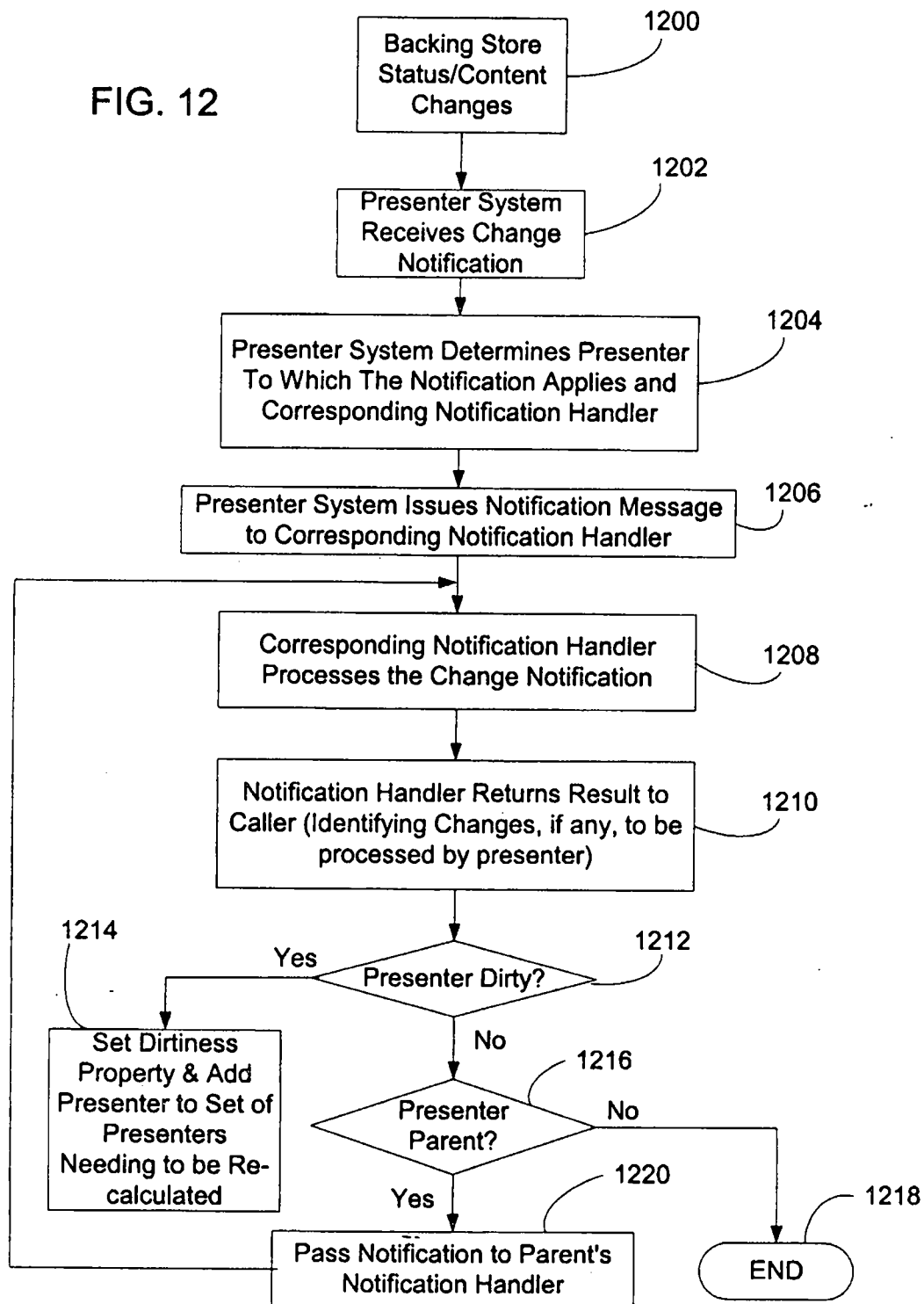
FIG. 10

Presentation Engine Class	
1100	Constructor
1102	DoLayout
1104	UpdateLayout
1106	CreatePage

FIG. 11

8/10

FIG. 12



9/10

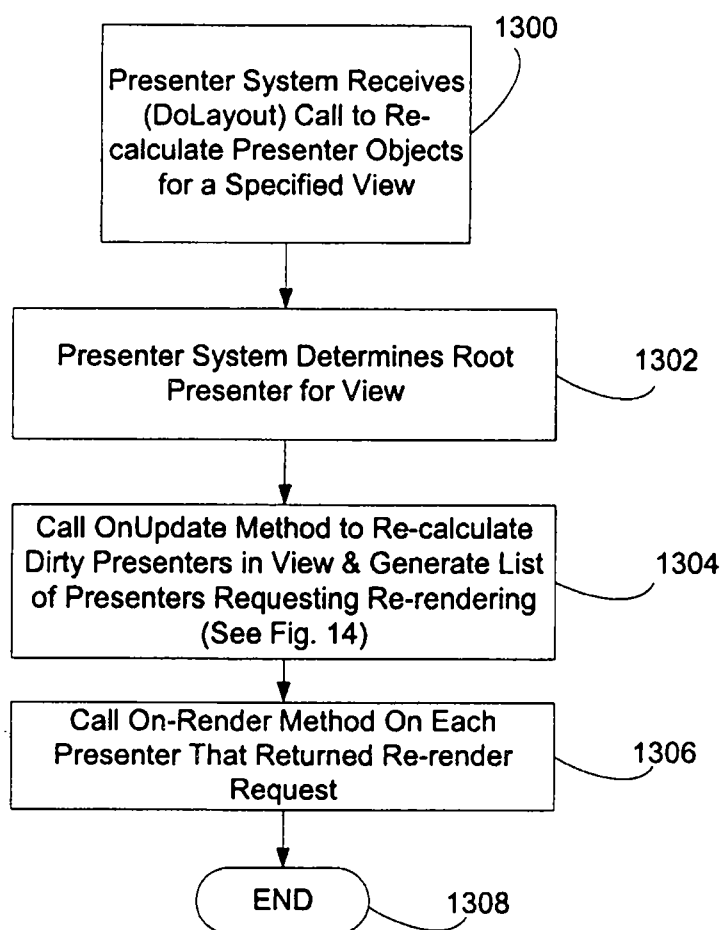


FIG. 13

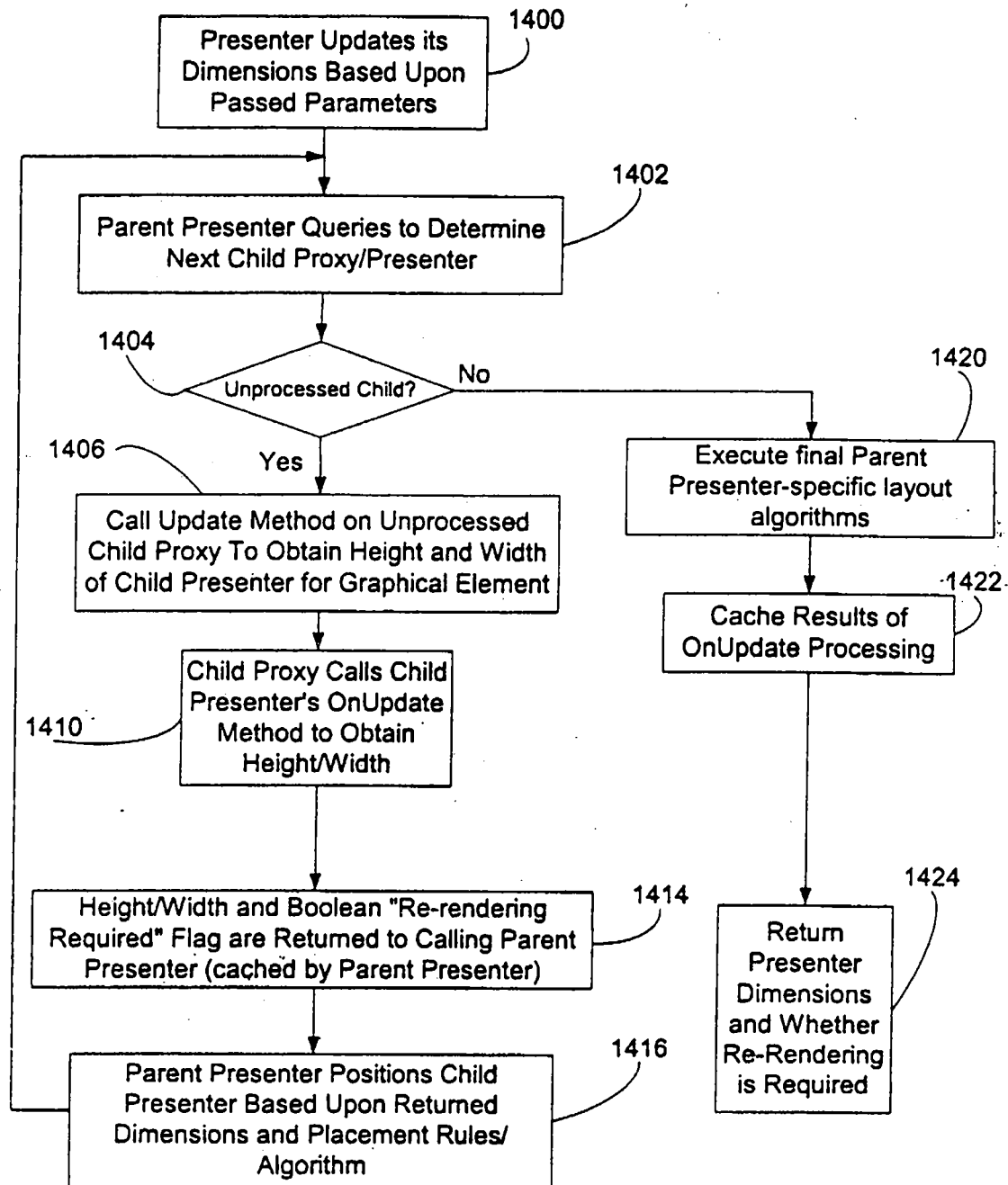


FIG. 14

SISTEMA PARA HOSPEDAR OBJETOS GRÁFICOS DE

DISEÑO/PRESENTACIÓN

CAMPO DE LA INVENCION

La presente invención se relaciona en general a dispositivos de cómputo. Más particularmente, la presente invención se relaciona a componentes de cómputo para disponer elementos gráficos visualizados a través de una interfaz gráfica de documento/usuario.

ANTECEDENTES DE LA INVENCION

Visualizar y/o convertir una salida gráfica para una aplicación que se ejecuta en un sistema de cómputo incluye muchas tareas. Una de tales tareas es la administración de diseño/presentación. La administración de diseño/presentación se relaciona con arreglar y adornar un conjunto de elementos de visualización dentro de sus espacios de visualización asignados (por ejemplo, rectángulos). Aunque "diseño" y "presentación" pueden ser distinguibles en otros contextos, presentación y diseño, como se usa aquí, deben ser tratados como términos intercambiables y equivalentes. Los elementos procesados

para visualización en sistemas de diseño/presentación suministran datos de salida gráficos para convertir componentes de un sistema de salida de gráficos y controladores de visualización gráfica. Tales componentes de conversión controlan el soporte físico de visualización de gráficos tales como monitores e impresoras.

Las operaciones de diseño/presentación ejecutadas por sistemas de cómputo comprenden una amplia variedad de funciones llevadas a cabo en elementos de visualización gráfica. Un ejemplo de tal operación es la disposición de texto dentro de dimensiones asignadas de una interfaz de usuario de visualización gráfica de una aplicación editora. La operación del diseño determina el contenido y la colocación de las líneas del texto dentro de un espacio rectangular designado. Otro ejemplo es encuadrar un elemento (por ejemplo, una caja de diálogo, una barra de herramientas, una barra de control, etc.) con un borde estandarizado. Las operaciones también se refieren aquí como "rasgos". Los componentes de conversión de un sistema de cómputo producen datos de salida gráficos basados en el estado de los elementos de visualización

que son modificados potencialmente por diseños ejecutados/operaciones de presentación/rasgos previos.

Varios motores de presentación son conocidos. Por ejemplo, MICROSOFT WINDOWS' tiene "el modulo de usuario" que lleva acabo operaciones de diseño con respecto a ventanas de alto nivel en una interfaz de usuario gráfico. El administrador de cajas de diálogo de MICROSOFT WINDOWS' coloca objetos en unas posiciones predefinidas y permite la agrupación de objetos. El posicionamiento de los objetos ocurre de una manera lógica y es independiente de un dispositivo físico. JAVA SWING permite operaciones de colocación básica con respecto a objetos dentro de una vista con base en propiedades especificadas sobre el objeto.

Las operaciones de diseño/presentación han sido llevadas a cabo de un número diferente de maneras. Una aplicación misma puede completamente llevar a cabo su operación de diseño de elementos de visualización. La operación de diseño (o "presentación") es encapsulada dentro de la aplicación. La aplicación adorna/dispone los elementos de visualización para una vista o vistas asociadas con un estado actual de la aplicación. La

aplicación dispone los elementos dentro de la región de visualización asignada a la aplicación. La aplicación de ahí en adelante convierte la salida de visualización, correspondiente a los elementos de visualización dispuestos, a los sistemas/controladores gráficos.

Otra disposición de procesamiento de diseño, incorporada en el sistema operativo MICROSOFT WINDOWS XP, suministra soporte para un conjunto predefinido de rasgos de diseño que pueden ser llamados por las aplicaciones. El conjunto de rasgos de diseño comprende operaciones de diseño/presentación especializadas para los elementos de visualización gráficos de la interfaz de usuario de las aplicaciones. El conjunto de rasgos de diseño/presentación predefinidos, tales como por ejemplo, un generador de bordes para suministrar rectángulos, reduce la carga de programación para los desarrolladores de aplicaciones con respecto a aspectos específicos de organización y presentación de elementos de una visualización gráfica. El conjunto de rasgos predefinidos también facilita la consistencia con respecto a ejecutar ciertos rasgos de diseño/funciones básicas tales como los bordes anteriormente señalados. Las aplicaciones suplementan las funcionalidades de diseño predefinido con

operaciones de diseño adicionales embebidas dentro de las aplicaciones mismas.

La arquitectura de procesamiento de diseño conocida, incluye rasgos de diseño predefinidos que pueden ser llamados por una aplicación, simplifica la programación de tareas de diseño encontradas por las aplicaciones durante el curso de la disposición de una vista antes de convertir los datos gráficos y los comandos para convertir componentes de un sistema de cómputo. Sin embargo, la arquitectura de procesamiento de diseño conocida no soporta fácilmente mejoras o modificaciones con respecto al conjunto de rasgos de diseño predefinidos que son llamables por las aplicaciones. Así, por ejemplo, nuevos rasgos de diseño son colocados directamente en el código de la aplicación en vez de colocar los rasgos en los rasgos de diseño predefinidos suministrados por la arquitectura de presentación/diseño. De ahí en adelante, durante la ejecución, la aplicación calcula el diseño utilizando una combinación de funcionalidades de diseño predefinidas y ejecuta internamente las operaciones de diseño/presentación.

RESUMEN DE LA INVENCION

La presente invención comprende un sistema presentador para incorporación dentro de un sistema de administración de diseños de salida gráfica. Tal administración de diseños está dirigida a la manera en la cual los objetos son dispuestos, dimensionados, y colocados en un espacio de visualización. Tales sistemas procesan un diseño, con base en un programa que suministra elementos gráficos que contienen datos que representan contenidos visualizables del programa (por ejemplo, un bloque de texto que requiere estar dispuesto en una página visualizada en pantalla o impresa). Los presentadores llevan a cabo un rol complementario a los elementos gráficos (que especifican datos) por medio de definir estados de visualización para presentar el contenido de los elementos gráficos. Como tal, los presentadores mantienen una descripción de visualización para un elemento gráfico correspondiente.

De acuerdo con la presente invención, el sistema presentador hospeda y dispone los presentadores asociados con elementos gráficos dentro de una vista. El sistema presentador, en su rol de soportar la derivación de los

presentadores de muchos tipos, incluye una clase base de presentador desde la cual las clases presentadoras se derivan. De ahí en adelante, los presentadores son generados desde las clases presentadoras derivadas.

El sistema presentador también incluye una interfaz huésped presentadora. La interfaz huésped comprende un método para diseñar elementos gráficos en una vista de acuerdo con los presentadores asociados con los elementos gráficos. Como tal, el diseño de los elementos gráficos dentro de una vista es especificado por los presentadores asociados.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

Aunque las reivindicaciones adjuntas establecen las características de la presenta invención con particularidad, la invención y sus ventajas se entienden mejor luego de la siguiente descripción detallada tomada en conjunto con los dibujos adjuntos, en los cuales:

La Figura 1 es un diagrama de bloques que ilustra un sistema de cómputo ejemplar para llevar a cabo una modalidad de la invención;

La Figura 2 es un diagrama esquemático de alto nivel que ilustra los componentes principales de una arquitectura de administración de diseño que incluye un sistema presentador para llevar a cabo el mantenimiento de diseños gráficos en un sistema de cómputo que involucra la presente invención;

La Figura 3 es un dibujo esquemático que ilustra las relaciones entre los elementos gráficos y los presentadores definiendo una interfaz gráfica de documento/usuario en un sistema de la presente invención;

Las Figuras 4a y 4b resumen la estructura de una clase base presentadora desde la cual clases de objetos presentadores adaptados son derivados para llevar a cabo tareas de diseño particular/de conversión en un sistema de la presente invención;

La Figura 5 resume la estructura de una clase encaminadora niño;

La Figura 6 resume la estructura de una interfaz de clase manejadora de notificación;

La Figura 7 resume la estructura de una clase de sitio manejador de notificación;

La Figura 8 resume la estructura de una clase de vista;

La Figura 9 resume las partes de una estructura `BoxSizeInfo`;

La Figura 10 resume las partes de una estructura de descriptor de página;

La Figura 11 resume la estructura de una clase de motor de presentación;

La Figura 12 es una tabla de flujo que resume etapas ejemplares para procesamiento con base en manejador de notificación de un cambio a un elemento gráfico;

La Figura 13 es una tabla de flujo que resume las etapas ejemplares para recalcular y convertir una vista

que comprende un conjunto de presentadores de acuerdo con una modalidad de la presente invención; y

La Figura 14 es una tabla de flujo que describe ilustrativamente las etapas para ejecutar una actualización sobre un presentador (y sus hijos asociados afectados) en respuesta a un cambio a un elemento gráfico relacionado.

DESCRIPCIÓN DETALLADA DE LOS DIBUJOS

Se describe aquí una arquitectura de interfaz de usuario y de diseño/presentación de documento. La arquitectura de diseño/presentación descrita incluye componentes gráficos de proceso de diseño/conversión (presentadores) que llevan a cabo aspectos de la vista de datos subyacentes asociados (elementos gráficos) para una interfaz de documento/usuario. Un sistema presentador hospeda los presentadores. En su rol de anfitrión, el sistema presentador orquesta y coordina la actualización de una vista, comprende un conjunto de presentadores, en respuesta a cambios a elementos gráficos correspondientes. Así, el sistema presentador suministra un mecanismo que enlaza el estado (por ejemplo, estado de

los datos) de elementos gráficos a los estados de vista actualizados, almacenados, y convertidos por objetos presentadores correspondientes.

La arquitectura con base en sistema presentador de diseño/presentación de la presente invención es su alto grado de extensibilidad con respecto a los tipos de presentadores que ésta hospeda. Una clase base de presentador suministrada por el sistema presentador facilita la extensión de un conjunto de tipos de presentadores (clases de objetos) disponible para diseñar y convertir información siempre suministrada por un elemento gráfico correspondiente. Los tipos de presentador diversos derivados de la clase base presentadora incorpora métodos particulares de diseño/conversión a medida que se sobrepone a métodos por omisión especificados por la clase presentadora base. Los objetos presentadores derivados del conjunto de clases de objeto presentador implementan rasgos de diseño/presentación de acuerdo con el estado de los datos de un elemento gráfico correspondiente.

El sistema presentador hospeda los objetos presentadores derivados del conjunto de clases de objeto

presentador. El sistema presentador lleva a cabo la administración de por vida (es decir, la creación, eliminación, etc.) de objetos presentadores. El sistema presentador maneja un rastreo tramposo para actualizar de manera selectiva solamente los presentadores cambiados. El sistema presentador soporta presentadores encadenados juntos para suministrar adornos a los presentadores principales. El sistema presentador también coordina la actualización y la conversión de presentadores.

La notificación de los cambios a elementos gráficos que efectúa potencialmente los estados de las vistas se facilita, en una modalidad de la invención, por los manejadores de notificación. Los manejadores de notificación que el manejo de la notificación de los cambios esté separado del procesamiento de los cambios llevados a cabo por los presentadores. Cada tipo de elemento presentador está asociado con un tipo de manejador de notificación. Cada manejador de notificación procesa los cambios a un elemento gráfico correspondiente. Si se requiere actualización, entonces el manejador de notificación inicia el establecimiento de una propiedad tramposa sobre un objeto presentador asociado. El conjunto de propiedades tramposas causa el

recálculo de un objeto presentador en respuesta a los cambios a un elemento gráfico correspondiente.

La arquitectura de interfaz de usuario diseño/presentación, descrita como vía de ejemplo aquí, se incorpora en un sistema operativo que hospeda las aplicaciones que tienen interfaces gráficas de usuario/documento. Las aplicaciones, presentadores, y componentes de conversión específicos de dispositivos de los sistemas operativos y de los controladores específicos de dispositivos, convierten los comandos/datos del control del dispositivo de la salida gráfica de interfaz/documento que visualicen o muestren el soporte lógico del dispositivo (por ejemplo, monitores, impresoras, etc.). Las características del diseño de la salida convertida están basadas en por lo menos en parte sobre el procesamiento y conversión del diseño/presentación llevado a cabo por los presentadores hospedados por el sistema presentador.

La arquitectura de presentación permite desasociar un estado de datos de elementos visualizados de sus estados de vista. Tal disociación suministra una plataforma de adaptación extensible para crear nuevas y

ricas personalidades de visualización para un elemento gráfico subyacente. En una modalidad de la invención, un elemento de visualización está representado por un objeto de elemento gráfico (que representa el estado de datos de los elementos) y por lo menos un objeto presentador asociado (que representa un estado de vista) para diseñar el elemento en una vista particular. El presentador lleva a cabo una actualización adaptada de un diseño para un elemento relacionado. El presentador también convierte la vista del elemento después de actualizar el diseño.

Utilizando objetos presentadores que disocian el estado de los datos de un elemento visualizado y su visualización en una vista. Como consecuencia, cada elemento gráfico (fuente de datos visualizados) es capaz potencialmente de tener presentadores múltiples que suministren vistas distintas del elemento gráfico (por ejemplo, una vista a escala completa y una vista miniatura de un elemento de fotografía). La habilidad para asociar presentadores múltiples con un elemento gráfico único también permite que un elemento único sea asociado con localizaciones múltiples dentro de un campo (por ejemplo, dividir un elemento gráfico único entre columnas múltiples). También, el sistema presentador

soporta presentadores distintos de múltiple encadenamiento de tipos diferentes a un elemento único soportando así la sujeción de una variedad de adornos adaptados a una vista básica para un elemento. Estos, lo mismo que otros aspectos de una arquitectura de visualización de diseño/conversión ejemplar se describen aquí en seguida con referencia a las figuras.

La Figura 1 ilustra de manera ejemplar un ejemplo de un ambiente operativo adecuado 100 para llevar a cabo la arquitectura de visualización de diseño/conversión con base en un sistema presentador de la presente invención. El ambiente operativo 100 es solamente un ejemplo de un ambiente operativo adecuado y no tiene el propósito de sugerir ninguna limitación en cuanto al alcance del uso o funcionalidad de la invención. Otros sistemas de cómputo bien conocidos, ambientes, y/o configuraciones que pueden ser adecuados para el uso con la invención incluyen, pero no se limitan a, computadores personales, computadores servidores, dispositivos de cómputo manuales o portátiles, dispositivos de cómputo manuales, sistemas multiprocesadores, sistemas con base en microprocesador, PCs de red, minicomputadores, computadores medios, ambientes distribuidos de cómputo que incluyen

cualesquiera de los anteriores sistemas o dispositivos, y similares. La arquitectura de administración de diseños descrita que incluye un sistema presentador sirve a un número de intereses diversos incluyendo extensibilidad, integración, y uniformidad con respecto a la funcionalidad de visualización de un sistema de cómputo, tal como el ilustrado en la Figura 1.

La invención está descrita en el contexto general de un conjunto de etapas y procesos que se llevan a cabo por instrucciones ejecutables por computador, tal como módulos de programa, que son ejecutados por un computador. Generalmente, los módulos de programa incluyen rutinas, objetos de programa, componentes, estructuras de datos, etc. que llevan a cabo tareas particulares o implementan tipos de datos abstractos particulares. Aunque la modalidad ilustrativa está descrita con referencia a procesos ejecutados localmente en un sistema de cómputo único, la invención es incorporada potencialmente dentro de nodos de redes que operan en ambientes de cómputo distribuidos en donde las tareas se llevan a cabo por dispositivos de procesamiento remoto que están enlazados por medio de una red de comunicaciones. En un ambiente de cómputo distribuido,

los módulos de programa son localizados generalmente tanto en medios de almacenamiento de computadores locales y remotos incluyendo dispositivos de almacenamiento de memoria.

Con referencia continuada a la Figura 1, un sistema ilustrativo para implementar la invención incluye un dispositivo de cómputo de propósito general en la forma de un computador 110. Los componentes del computador 110 pueden incluir, pero no limitarse a, una unidad de procesamiento 120, una memoria de sistema 130, y un bus del sistema 121 que acopla los diversos componentes del sistema incluyendo la memoria del sistema a la unidad de procesamiento 120. El bus del sistema 121 puede ser de cualquiera de los tipos diversos de estructuras de bus incluyendo un bus de memoria o un controlador de memoria, un bus de periféricos, y un bus local que usa cualquiera de una variedad de arquitecturas de bus. Por vía de ejemplo, y no limitativo, tales arquitecturas incluyen la Industry Standard Architecture (ISA) como bus, el bus Micro Channel Architecture (MCA), el bus Enhanced ISA (EISA), el bus local Video Electronics Standards Association (VESA), y el bus

Peripheral Component Interconnect (PCI) también conocido como bus Mezanine.

El computador 110 típicamente incluye una variedad de medios legibles por computador. Los medios legibles por computador pueden ser cualquier medio disponible que pueda ser accesado por el computador 110 e incluye tanto medios volátiles como no volátiles, medios removibles y no removibles. Como vía de ejemplo, y no por limitación, los medios legibles por computadora pueden comprender medios de almacenamiento de computadora y medios de comunicación. Los medios de almacenamiento de computadora incluyen tanto medios volátiles y no volátiles, removibles y no removibles implementados en cualquier método o tecnología para almacenar la información tal como instrucciones legibles por computador, estructuras de datos, módulos de programa y otros datos. Los medios de almacenamiento de computadora incluyen, pero no se limitan a, RAM, ROM, EEPROM, memoria flash y otras tecnologías de memoria, CD-ROM, discos digitales versátiles (DVD) u otros almacenamientos de disco óptico, cassettes magnéticos, cinta magnética, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, o cualquier otro medio que

pueda ser utilizado para almacenar la información deseada y que pueda ser accesado por la computadora 110. Los medios de comunicación típicamente incluyen instrucciones legibles por computadora, estructuras de datos, módulos de programa y otros datos en una señal de datos modulada tal como una onda portadora u otro mecanismo de transporte e incluye cualquier medio de suministro de información. El término "señal de datos modulada" significa una señal que tiene una o más de sus características establecidas o cambiadas de tal manera como para codificar información en la señal. Por vía de ejemplo y no limitativamente, los medios de comunicación incluyen medios alambrados tales como redes alambradas o redes de conexión directa, y medios inalámbricos tales como medios acústicos, RF, infrarrojos y otros medios inalámbricos. Combinaciones de cualquiera de las anteriores también están incluidas dentro del alcance de los medios legibles por computador.

La memoria del sistema 130 incluye medios de almacenamiento en computador en la forma de memoria volátil y/o no volátil tales como la memoria de sólo lectura (ROM) 131 y la memoria de acceso aleatorio (RAM) 132. Un sistema básico de entrada/salida 133 (BIOS), que

contiene las rutinas básicas que ayudan a transferir la información entre elementos dentro del computador 110, tal como durante el arranque, son almacenados algunas veces en ROM 131. La RAM 132 típicamente contiene los datos y/o módulos de programa que son accesibles inmediatamente a y/o están presentes para ser operados por la unidad de procesamiento 120. Como vía de ejemplo, y sin limitación, la Figura 1 ilustra el sistema operativo 134, los programas de aplicación 135, y otros módulos de programas 136, y los datos de programa 137.

El computador 110 también puede incluir otros medios de almacenamiento por computador removibles/no removibles, volátiles/no volátiles. Como vía de ejemplo solamente, la Figura 1 ilustra un aparato de disco duro 140 que lee de o escribe a un medio magnético no removible, no volátil, un aparato de disco magnético 151 que lee de o escribe en un disco magnético removible, no volátil 152, y un aparato de disco óptico 155 que lee de o escribe en un disco óptico removible, no volátil 156 tal como un CD-ROM u otro medio óptico. Otros medios de almacenamiento en computador removible/no removible, volátil/no volátil que pueden ser usados en el ambiente operativo ilustrativo incluyen, pero no se limitan a,

cassettes de cinta magnética, tarjetas de memoria flash, discos versátiles digitales, cintas de video digital, RAM de estado sólido, ROM de estado sólido, y similares. El aparato de disco duro 141 está típicamente conectado al bus del sistema 121 por medio de una interfaz de memoria no removible tal como la interfaz 140, y el aparato de disco magnético 151 y el aparato de disco óptico 155 están típicamente conectados al bus del sistema 121 por una interfaz de memoria removible, tal como la interfaz 150.

Los aparatos y sus medios de almacenamiento por computadora asociados discutidos anteriormente e ilustrados en la Figura 1, suministran el almacenamiento de las instrucciones, estructuras de datos, módulos de programa u otros datos legibles por computadora para el computador 110. En la Figura 1, por ejemplo, el aparato de disco duro 141 está ilustrado en que almacena el sistema operativo 144, los programas de aplicación 145, otros módulos de programa 146, y los datos de programa 147. Note que estos componentes pueden ser los mismos o diferentes al sistema operativo 134, los programas de aplicación 135, u otros módulos de programa 136, y los datos de programa 137. El sistema operativo 144, los

programas de aplicación 145, otros módulos de programa 146 y los datos de programa 147 se les dan diferentes números aquí para ilustrar que, como un mínimo, ellos son copias diferentes. Un usuario puede entrar comandos e información al computador 20 por medio de dispositivos de entrada tal como teclado 162 o un dispositivo de apuntamiento 161, comúnmente conocido como un ratón, una bola de rastreo o una almohadilla de toque. Otros dispositivos de entrada (no mostrados) pueden incluir un micrófono, un joystick, una almohadilla de juego, un plato satélite, un escáner, o similares. Estos y otros dispositivos de entrada están conectados frecuentemente a la unidad de procesamiento 120 por medio de una interfaz de entrada de usuario 160 que está acoplada al bus del sistema, pero que puede estar conectada por otra interfaz y estructuras de bus, tal como puerto paralelo, puerto de juegos o un puerto serial universal (USB). Un monitor 191 u otro tipo de dispositivo de visualización también puede estar conectado al bus del sistema 121 por medio de una interfaz, tal como una interfaz de video 190. En adición al monitor, los computadores también pueden incluir otros dispositivos de salida periféricos tales como parlantes 197 e impresoras 196, que pueden estar conectados por medio de una interfaz periférica de salida 195.

El computador 110 opera potencialmente en un ambiente de red usando conexiones lógicas a uno o más computadores remotos, tal como un computador remoto 180. El computador remoto 180 puede ser un computador personal, un servidor, un enrutador, un PC de red, un dispositivo paralelo u otro nodo de red común y típicamente incluye muchos o todos de los elementos descritos anteriormente con relación al computador 110, aunque solamente un dispositivo de almacenamiento en memoria 181 ha sido ilustrado en la Figura 1. Las conexiones lógicas ilustradas en la Figura 1 incluyen una red de área local (LAN) 171 y una red de área amplia (WAN) 173, pero puede incluir también otras redes. Tales ambientes de red son muy comunes en oficinas, en redes de computadores de empresas grandes, intranets y la Internet.

Cuando es usado en un ambiente de red LAN, el computador 110 está conectado a la LAN 171 por medio de una interfaz de red o adaptador 170. Cuando es usado en un ambiente de red WAN, el computador 110 típicamente incluye un módem 172 u otro medio para establecer comunicaciones en la red WAN 173, tal como la Internet.

El módem 172, que puede ser interno o externo, puede estar conectado al bus del sistema 121 por medio de una interfaz de entrada de usuario 160, u otro mecanismo apropiado. En un ambiente de red, los módulos de programa ilustrados con relación al computador 110, o porciones de ellos pueden estar almacenados en un dispositivo de almacenamiento de memoria remoto. Por vía de ejemplo, y no como limitación, la Figura 1 ilustra los programas de aplicación remotos 185 como residentes en el dispositivo de almacenamiento de memoria 181. Se podrá apreciar que las conexiones de red mostradas en la ilustración y otros medios de establecer un enlace de comunicaciones entre los computadores pueden ser usados.

La Figura 2 es un diagrama esquemático de alto nivel que identifica los módulos de soporte lógico y de programas lo mismo que lo objetos derivados y los componentes de datos de una arquitectura de sistema de administración de diseño/presentación de la presente invención. La arquitectura del sistema de administración de diseño/presentación ilustrativo representado en la Figura 2 incluye un sistema presentador 200 que soporta el diseño/presentación de objetos despleables con base en un conjunto de objetos presentadores dispuestos

jerárquicamente asociados con una vista 202 creada, desde una clase de objeto de vista (ver, Figura 8 descrita aquí más adelante) en la solicitud de una aplicación 204 que se ejecuta sobre un sistema operativo 205.

En una modalidad de la invención, cada vista (por ejemplo, la vista 202) tiene un sistema presentador correspondiente (por ejemplo, el sistema presentador 200) que es responsable de mantener los objetos desplegados dentro de la vista. En una modalidad de la invención la vista 202 es una raíz visual para todas las visuales que son creadas por el sistema presentador 200. El contenido desplegado dentro de la vista 202 está basado en elementos gráficos 206 dentro de la estructura de respaldo 208. La aplicación 204 es una fuente de elementos gráficos 206 dentro del almacén de respaldo 208. Un ejemplo ilustrativo que ilustra las relaciones entre las vistas, los sistemas presentadores, los árboles presentadores para las vistas, y los árboles de elementos gráficos asociados con una aplicación se ilustran en la Figura 3 descrita de aquí en adelante.

La arquitectura del sistema de administración de diseño/presentación ilustrado en la Figura 2 está

dividido en un conjunto de componentes funcionales. Las líneas entre los componentes ilustrados representan los trayectos de interacción entre los componentes. La presente invención no está limitada a una disposición particular de los componentes y sus interacciones ilustrativas indicadas en la modalidad ilustrada en la Figura 2. En vez de eso, la funcionalidad de los componentes descritos aquí abajo están agrupados en diferentes maneras de acuerdo con modalidades alternativas de la invención.

En una modalidad de la invención, la aplicación 204 llama a un método constructor sobre una interfaz de programa de aplicación de clase de objeto de vista (ver estructura de clase de vista en la Figura 8 descrita aquí más abajo) para crear un objeto raíz para la vista 202 en la derivación de objeto. El método de llamada para crear la vista 202 pasa una referencia a un elemento gráfico de raíz de los elementos gráficos 206 contenidos en el almacén de respaldo 208. Después de crear la vista 202, una llamada al método DoLayout en la vista 202 crea un árbol presentador para la vista 202 con un objeto presentador de raíz correspondiente al elemento gráfico de raíz pasado inicialmente al método constructor que

creó la vista 202. El árbol presentador (no mostrado en la Figura 2) para la vista 202 comprende un conjunto de presentadores que son responsables de organizar y convertir la salida visual para la vista 202.

En una modalidad de la invención cada vista (por ejemplo, la vista 202) posee un sistema presentador (por ejemplo, el sistema presentador 200). Así, en respuesta al método DoLayout, la vista 202 inicia creando un nuevo árbol presentador y lo regresa al llamador (por ejemplo, la aplicación 204). Inicialmente, la vista 202 llama a un método constructor sobre una clase de objeto de motor de presentación para crear un motor de presentación 212 derivado que incluye una interfaz de programa de aplicación anfitrión presentador (el anfitrión presentador API). El motor de presentación 212 incluye el código ejecutable para coordinar y orquestar la ejecución de las operaciones de diseño/presentación en un sistema de cómputo que soporta una salida gráfica. Las funciones llevadas a cabo/coordinadas por el motor de presentación 212 incluyen: crear objetos presentadores (tanto versiones raíz como versiones hijo), coordinar el recálculo de los estados de visualización de los presentadores, convertir, administrar de por vida los

presentadores, el rastreo tramposo, los presentadores de cadena múltiple a un elemento único, y el diseño incremental de comportamiento. Ejemplos de los métodos llamables en el anfitrión API del motor de presentación 212 como se describió aquí en seguida con referencia a la Figura 12. Estos métodos son requerimientos de alto nivel que crean y mantienen el árbol de objetos presentadores para la vista 202. Las tareas identificadas llevadas a cabo por el motor de presentación 212 son ilustrativas y aquellos versados en la técnica podrán apreciar que tareas adicionales/alternativas pueden llevarse a cabo por el motor de presentación 212 en modalidades alternativas de la invención.

El motor de presentación 212 construye el sistema presentador 200 para manejar los objetos presentadores asociados con la vista 202. En una modalidad de la invención, el sistema presentador 200 incluye una clase base presentadora 216. La clase base presentadora 216 es una clase de base caracterizada completamente por crear subclases presentadoras y objetos presentadores capaces de explotar la funcionalidad de administración de diseño del sistema descrito aquí. La clase base presentadora 216 comprende una plantilla, e incluye un código de programa

virtual (sobreponible) y estructuras de datos desde las cuales un conjunto de clases presentadoras 214 se derivan. Las clases presentadoras 214 comprenden un conjunto de tipos de presentador especializados que corresponden a vistas, de elementos gráficos, que tienen propiedades/comportamientos de visualización particulares. Las clases presentadores adaptadas 214 derivadas de la clase base presentadora 216, por vía de ejemplo, contienen diseño, hijos dispuestos, visuales creadas para elementos de sujeción, contenidos de paginado para documentos tipográficos, y visuales de actualización incrementales que incluye elementos gráficos 206. La funcionalidad especializada de cada clase presentadora dentro del conjunto de clases presentadoras 214 se establece por sobreponer y suplementar los métodos virtuales definidos por la clase base presentadora 216. Los objetos presentadores derivados desde el conjunto de clases presentadoras 214 representan y manejan aspectos de visualización de los elementos gráficos 206.

En una modalidad de la invención el conjunto de clases de objeto presentador 214, incluye tanto clases presentadoras 218 predefinidas como clases presentadoras

externas 220 es extensible. El conjunto de clases presentadoras predefinidas 218 están provistas con un sistema presentador 200. Las clases de objeto presentadores externos 220 son instaladas generalmente de manera independiente del sistema presentador 200. Sin embargo, una integración apropiada de las clases objeto presentadoras externas 210 con el sistema presentador 200 se facilita por una especificación de interfaz publicada para la clase de base presentadora 216 (descrita aquí en seguida con referencia a las Figuras 4a y 4b). Un desarrollador de clases presentadoras externas 220 se sobrepone/modifica los códigos de programa y las estructuras de datos asociados con los métodos adaptables de diseño/conversión de la clase presentadora 216 para lograr el dimensionado adaptable, el posicionamiento del niño, las funciones de conversión. De ahí en adelante, la nueva clase presentadora es instalada en el conjunto de clases de objetos presentadores externos 220, como por ejemplo, instalar un archivo que contiene una nueva clase de objetos en un directorio designado dentro de un sistema de archivos de computador.

En una modalidad de la invención, las clases presentadoras 214 incluyen clases presentadoras para

llevar a cabo características de diseño que incluyen, como vía de ejemplo: diseño de contenidos, disposición de objetos hijo, creaciones visuales, paginación del contenido del documento, y actualizaciones incrementales de las visuales (por ejemplo, páginas HTML que se expanden en pantallas de visualización múltiples). Ejemplos de tipos de presentadores específicos se incluyen: un presentador de texto que mide y despliega el texto formateado de varias maneras (por ejemplo, en negrilla, en itálica, etc.), un presentador de imagen que mide y despliega una imagen especificada, y un presentador de documento que hace una partición de un espacio entre múltiples presentadores hijo.

Después de crear el sistema presentador 200 y las clases presentadoras 214, un conjunto de objetos presentadores asociados con la vista 202 y el sistema presentador 200 son derivados para la vista, iniciando con el objeto presentador raíz. Se puede notar que el método constructor llama a la clase de objeto de motor de presentación para crear el motor de presentación 212 que identifica el elemento gráfico de raíz para la vista 202. Después de la derivación, el motor de presentación 212 llama a los elementos gráficos 206 para determinar un

tipo de objeto presentador asociado con el elemento gráfico raíz para la vista 202. Después de determinar el tipo de presentador del elemento gráfico, el motor de presentación 212 llama un método, sobre una de las clases de objeto presentadores 214 que corresponde al tipo de objeto presentador identificado por los elementos gráficos 206, para derivar el objeto presentador raíz para la vista 202.

Como se indicó por el paso entre las clases presentadoras 214 y el almacén de respaldo, los objetos presentadores, que incluyen el presentador de raíz, accedan a los elementos gráficos 206 del almacén de respaldo 208 para determinar las propiedades de los elementos gráficos particulares. Como vía de ejemplo, un objeto presentador para un elemento gráfico particular es capaz de identificar cualquier hijo de elemento gráfico del elemento gráfico particular y el tipo de presentador para cada elemento gráfico hijo. De ahí en adelante, el objeto de presentador llama, para cada elemento gráfico hijo identificado, un método sobre sí mismo (por ejemplo, `GetChildProxyForElement` 486) para derivar un objeto encaminador hijo desde una clase encaminadora hijo 222 del sistema presentador 200. Los objetos encaminadores

hijo son objetos envueltos en objetos presentadores (derivados de las clases de objeto presentador 214) están anidados dentro del espacio visualizado asignado a un objeto presentador padre. La clase encaminadota hijo 222 cumple con las necesidades de encapsulación y seguridad (por ejemplo, un objeto presentador hijo limita el acceso por un objeto presentador padre, o la aplicación 204, a los contenidos de un presentador contenido dentro del presentador hijo). Alternativamente, la capa de encapsulación suministrada por los objetos encaminadores hijo son sobrepasados y los presentadores padre accedan directamente sus presentadores hijo.

El método sobre el objeto presentador para crear un objeto encaminador hijo para un elemento, en una modalidad de la invención, llama un método constructor sobre la clase encaminadota hijo 222 y pasa la identidad del elemento gráfico hijo para derivar un nuevo objeto encaminador hijo y objeto presentador para el elemento gráfico hijo. Una vez derivado, el nuevo objeto encaminador hijo crea un objeto presentador para el elemento gráfico hijo por medio de un constructor presentador correspondiente al tipo de presentador especificado en el elemento gráfico hijo identificado. El

objeto encaminador hijo, a su vez, regresa a su propia referencia al objeto presentador padre. El objeto presentador padre mantiene la referencia del objeto encaminador hijo regresada en un arreglo de referencias para objetos encaminadores hijo. Como tal las entradas del arreglo de objetos encaminadores hijo para un objeto presentador correspondiente al elemento gráfico hijo para un elemento gráfico padre con el cual el presentador padre está asociado.

Otro aspecto de la arquitectura de elementos gráficos de disposición/presentación ilustrada en la Figura 2 es la notificación de cambio. En una modalidad de la presente invención, un conjunto de clases manejadoras de notificación 224 se suministran para los objetos presentadores derivados de las clases presentadoras 214. Las clases manejadoras de notificación 224 están definidas de acuerdo con una clase base manejadora de notificación 225. La clase base manejadora de notificación 225 es una clase abstracta que define una interfaz para las clases manejadoras de notificación 224. La clase base manejadora de notificación 225 suministra la definición de interfaz de dos métodos virtuales (ver Figura 6), la implementación de los cuales se suministra

en las clases manejadoras de notificación 224 de acuerdo con sus clases presentadoras correspondientes 214. Las clases manejadoras de notificación 224 ayudan a actualizar el diseño incremental, recalcular solamente los objetos presentes efectuados por cambios a un conjunto de elementos gráficos correspondientes. En una modalidad de la invención, un manejador de notificación es especificado para un elemento gráfico y combinaciones de vista. En casos en donde presentadores múltiples son creados para una combinación única de elemento gráfico/vista, el manejador de notificación único opera como una fuente única para rastrear los cambios al elemento con propósitos de actualizar los presentadores múltiples para el elemento en una vista. Cada tipo de presentador especifica un tipo de manejador de notificación particular correspondiente. Cuando un objeto presentador para un elemento es derivado y que requiere un manejador de notificación, el sistema presentador crea un manejador de notificación correspondiente del tipo especificado y enlaza el manejador de notificación con el objeto presentador.

Las derivaciones de las clases de manejador de notificación 224 determina que porciones de los objetos

presentadores están dañados y por lo tanto requieren actualización. Inicialmente, una relación de notificación se establece entre un elemento gráfico y un manejador de notificación creado para un presentador en una vista (por ejemplo, la vista 202 para la aplicación 204). El manejador de notificación recibe notificaciones de cambio asociadas con el elemento gráfico en el almacén de respaldo 208 por medio del motor de presentación 202 del sistema presentador 200. El manejador de notificación determina una porción, si existe, del presentador que ha sido afectada por el cambio al elemento. El manejador de notificación regresa la información de porción afectada al sistema presentador 200. El presentador está marcado como "alterado" por el motor de presentación 212. El sistema presentador 200 acumula la información de presentador "alterado" para todos los presentadores en una estructura de datos mantenida para una vista (por ejemplo, vista 202). En un tiempo después, en respuesta a una llamada para recalcular el diseño para una vista, el motor de presentación 212 del sistema presentador 200 invoca un método/operación de actualización solamente sobre los presentadores alterados. Así, las clases manejadoras de notificación 224 permiten limitar las actualizaciones de diseño a solamente los objetos

presentadores afectados creados por las clases presentadoras 214.

La arquitectura de sistema de administración de diseño de la presente invención opera independientemente de cualquier dispositivo de soporte físico de salida particular sobre el cual los elementos diseñados son visualizados finalmente de acuerdo con las vistas definidas. La arquitectura del sistema de administración de diseño/presentación ilustrado en la Figura 2 lleva a cabo una etapa de procesamiento con respecto a la salida de visualización gráfica. En una modalidad de la invención, la salida dispuesta asociada con la arquitectura del sistema de administración de diseño/presentación es independiente del dispositivo. De ahí en adelante, durante la etapa de conversión los objetos presentadores convierten la vista recalculada a componentes de salida gráficos de un sistema de cómputo. Sin embargo, en otras modalidades de la invención, la salida del sistema de diseño/presentación es regresada sea por los objetos presentadores 214 o el sistema presentador 200 a la aplicación que llama 204, y la aplicación 204 lleva a cabo la tarea de conversión. En una modalidad de la invención, los comandos de

conversión, sea que fueron emitidos por los objetos presentadores derivados de las clases presentadoras 214, la vista 202, o la aplicación 204, son pasados a un subsistema de gráficos y controladores de dispositivos de gráficos 234 sobre el computador del usuario. El subsistema de gráficos y los controladores del dispositivo de gráficos 234 transforma las instrucciones de conversión y los datos en un dispositivo de salida seleccionado particular tal como un monitor 236 o una impresora 238.

En resumen de la arquitectura de sistema de administración de diseño/presentación descrita anteriormente, un sistema presentador 200 suministra una plataforma de procesamiento de diseño independiente del dispositivo para aplicaciones tales como la aplicación 204. El procesamiento del diseño es llevado a cabo por objetos presentadores derivados de un conjunto de clases presentadoras 214 hospedadas por el sistema presentador 200. El sistema presentador 200 es parte de la vista 202. Los objetos presentadores corresponden a estados/conversiones de vista particulares de los elementos gráficos 206. Los estados de vista separados (objetos presentadores) de los estados de elementos

(elementos gráficos 206) permiten múltiples estados/presentadores de vista que sean independientes designados para un estado de elemento único.

El diseño es separado en vistas, tal como la vista 202, en donde cada vista corresponde a una región de visualización tal como un rectángulo particular sobre una pantalla de salida asignada a la aplicación 204 o a una página de documento en una salida de impresora gráfica. La separación de los estados de datos y vistas permite que un elemento único maneje múltiples vistas de ese elemento gráfico y por lo tanto permita que un elemento gráfico único sea visualizado en una variedad de vistas/maneras.

Otro aspecto del sistema de administración de diseño descrito es su alto grado de extensibilidad. Extendiendo el conjunto de clases presentadoras 214 se facilita por la clase base presentadora 216 provista por el sistema presentador 200 y el API presentador anfitrión del motor de presentación 212.

La arquitectura del sistema de administración de diseño anteriormente descrita soporta una variedad de

rasgos de procesamiento de diseño avanzados para salidas gráficas de conversión de aplicaciones para interfaces de usuario gráfico e impresoras. La habilidad para separar un elemento único (elemento gráfico) en múltiples estados de vista (presentadores) ayuda a la paginación y a los elementos divisores en columnas múltiples (una forma de paginación) en aplicaciones cuando la salida de impresora convertida o la vista de salida de impresora de un documento sobre una pantalla de visualización. Tales aplicaciones incluyen motores o buscadores web y programas de procesamiento de palabras.

Otro beneficio de los presentadores múltiples es la habilidad de las aplicaciones para llevar a cabo intentos múltiples de ajustar su contenido dentro de un espacio de visualización dado. Las aplicaciones pueden derivar ensayos múltiples sobre objetos presentadores diferentes, para un elemento único, con parámetros de entrada diferentes y luego permiten al usuario seleccionar la mejor de las vistas mostradas del elemento gráfico.

Todavía otro beneficio surge de la arquitectura de administración de diseño anteriormente descrita es la

habilidad para llevar a cabo actualizaciones incrementales sobre una vista que comprende un conjunto de presentadores para un conjunto de elementos correspondientes. El cambio es implementado sobre una base de elemento por elemento en el almacén de respaldo (ver por ejemplo, almacén de respaldo 308 en la Figura 3). Los manejadores de notificación y el sistema presentador 200 suministran una infraestructura para limitar las actualizaciones de una vista correspondiente a los presentadores que han sido afectados por cambios para elementos correspondientes en el almacén de respaldo.

Todavía otro beneficio de la arquitectura de administración de diseño ilustrativa es la habilidad para adornar los estados de vista de los elementos involucrados en los presentadores creados previamente. Esto se lleva a cabo por presentadores adicionales de encadenamiento en un presentador de un tipo especificado para un tipo de elemento particular. El encadenamiento permite que los desarrolladores mejoren las interfaces de usuario y la salida del documento agregando bordes, fondos, etc., a un estado de vista especificado por un tipo de presentador particular para un elemento.

Habiendo descrito la arquitectura general del sistema de administración de diseño de la presente invención, dirigimos la atención a la Figura 3 la cual ilustra las relaciones entre las vistas, motores de presentación, objetos presentadores dentro de las vistas, elementos gráficos, y manejadores de notificación para una vista ilustrativa particular.

Conceptualmente una vista define una manera particular de diseñar un conjunto de elementos gráficos representados en un almacén respaldo 208 de acuerdo con presentadores especificados. En una modalidad de la invención una vista esta definida por un rectángulo en un campo de salida (por ejemplo una pantalla de visualización). En adición aun rectángulo y su posición, una vista también especifica elementos gráficos y presentadores asociados contenidos en la vista. Adicionalmente, se nota que en una modalidad de la invención cada vista esta asociado con su propio derivado de sistema presentador (por ejemplo, el sistema presentador 200a para la 202a y el sistema presentador 200b para la vista 202b).

Con referencia continuada la figura 3, el almacén de respaldo 208 comprende una colección organizadas de elementos gráficos (Ex) asociados con las vistas 202a y 2002b. los elementos gráficos (Ex) son entidades de contenido de usuario organizado. Ejemplos de los elementos incluyen: un botón de interface de usuario gráfico (GUI) o panel de texto, una ventana de editor de texto, un mapa de bit, etc. en una modalidad de la invención, los elementos gráficos (Ex) en el almacén de respaldo 208 están dispuestos en la forma de un árbol. Cada elemento (Ex) en el conjunto de elementos gráficos 206 mantenidos dentro del almacén de respaldo 208 esta asociados un conjunto de mantenedores de posición/propiedades (que pueden ser o explícitos o implicados en otros mantenedores de posición/propiedades explícitas). Una o muchas de tales mantenedores de posición/propiedades sobre los elementos gráficos especifican un tipo de presentador que es usado para diseñar y convertir el contenido del elemento gráfico en cooperación con otros componentes del sistema del computador. En modalidades de la invención, los presentadores encadenados están definidos explícitamente, o alternativamente, están implicados de otras propiedades especificadas en un elemento, (por ejemplo, una

propiedad, "de bordes" de dos pixeles indica un presentador encadenado de borde). Los tipos de propiedad especificados para los elementos gráficos dependen de el elemento gráfico particular e incluyen: color, nombre de fuente, altura, ancho, etc.

El almacén de respaldo 208 esta asociado con vistas potencialmente múltiples (por ejemplo las vistas 202a y 202b). Cada una de las vistas potencialmente múltiples tienen exclusivamente un conjunto de derivados de objeto presentador correspondiente aun conjunto de elementos gráficos dentro del almacén de respaldo 208. Así, en el caso de un elemento gráfico que esta siendo presentado en vistas múltiples, un objeto presentador es creado para el elemento dentro de cada vista (por ejemplo, el presentador P2' para el elemento E2). El elemento gráfico con el cual el presentador esta asociado es especificado en un campo propietario de elemento y corresponde a las líneas punteadas de los presentadores para sus elementos gráficos correspondientes. Esto es solamente un ejemplo de la arquitectura de presentación representada en las figuras 2 y 3 que soportan

presentadores múltiples que operan sobre la información suministrada por un elemento gráfico único.

Como una consecuencia de la arquitectura de presentación que contiene la habilidad anteriormente descrita para ligar vistas múltiples (y presentadores) a una misma colección de elementos gráficos en el almacén de respaldo 208, una interface de usuario gráfica es capaz de mostrar vistas múltiples de un mismo conjunto de información suministradas por los elementos gráficos. Por ejemplo, en la aplicación tal como MICROSOFT POWERPOINT, dos vistas simultaneas de una diapositiva están soportadas, la vista principal y la previsualización (o miniatura), para un mismo elemento gráfico de diapositiva.

La habilidad para crear derivados de objetos presentadores múltiples para un elemento gráfico único también facilita la paginación de un elemento gráfico único (por ejemplo un texto de documento) a través de páginas/columnas múltiples (es decir, en donde un elemento gráfico es dividido en dos páginas, dos objetos presentadores son creados para el elemento, uno por cada página). Con referencia a la figura 3, un objeto

presentador padre (por ejemplo P5) recibe una propiedad pasada desde un elemento gráfico (por ejemplo, E5) indicando que el contenido dentro del elemento (por ejemplo, el elemento de texto E8) es presentable en dos columnas. Dos objetos presentadores hijo P8a y P8b correspondiente a dos rectángulos de columna/páginas separados y distintos sobre el documento, son derivados bajo el presentador P5 para visualizar el texto suministrado por el elemento gráfico E8.

Todavía otro caso de derivación de presentadores múltiples para un elemento gráfico único surge en el caso de los adornos (por ejemplo, bordes, fondos, marcos, etc.) asignados a un elemento gráfico. Un ejemplo de un presentador encadenado se suministra por el presentador encadenado Q4 para el elemento E4 que suministra un borde alrededor de un rectángulo definido por el presentador P4 para mantener el texto suministrado por un campo de datos dentro del elemento E4. Ambos el presentado Q4 y el presentador P4 están muy enlazados a/asociados con un elemento gráfico E4 por medios de especificar el elemento gráfico E4 en sus campos propietarios de elemento. En una modalidad de la invención un presentador padre detecta de un elemento correspondiente a uno de sus hijos que un

presentador encadenado se requiere (por ejemplo, un borde de dos pixeles sobre el elemento). En el caso del presentador Q4 encadenado, cuando el presentador P1 llama un método sobre si mismo para crear un presentador hijo (encaminador) para un elemento hijo E4, este reconoce de las propiedades E4 que un presentador encadenado (Q4) se requiere. Primero el encaminador hijo/representado Q4 es creado y es identificado como un presentador hijo del presentador P1. Q4 a su vez, crea el encaminador hijo/presentador P4. P4 es así un presentador hijo del presentador Q4. La información es almacenada dentro del encaminador hijo para que presentador encadenado Q4 identifique el siguiente presentador (P4) en la cadena.

Habiendo descrito casos ilustrativos particulares que involucran presentadores múltiples que son ligados a un mismo elemento gráfico y que crean múltiples vistas (sistemas de presentador) para un conjunto de elementos gráficos en una almacena de respaldo 208, las relaciones ente las entidades mostradas en la figura 3 se describen ahora. Como se menciono anteriormente, las vistas 202a y 202b contienen los sistemas presentadores 200a y 200b, respectivamente. El sistema presentador 200a crea y soporta un conjunto de presentadores 302 para la vista

202a por medio de un código y una clase de base que incluyen un API en la cual las clases presentadoras están inscritas. El sistema presentador 200b crea y soporta un conjunto de presentadores 303 para la vista 202b.

Como se indico por las líneas entre los presentadores (Px) y los elementos gráficos (Ex) en el almacén de respaldo 208, los presentadores (Px) están asociados con elementos gráficos correspondientes (Ex). Los presentadores (Px) representan estados de vista para elementos gráficos correspondientes (Ex). Los elementos gráficos (Ex) en el almacén de respaldo 304 son mantenidos, como vía de ejemplo, en una estructura de árbol construida por una aplicación en una de muchas maneras posible. Ejemplos de tales maneras incluyen correr un archivo de marcación de texto por medio de un analizador, usando código para derivar los elementos gráficos, etc. el conjunto de presentadores 302 en la estructura de árbol es construida por el sistema presentador 200a en respuesta a una llamada aun método particular en una interface (por ejemplo, DoLayout) suministrada por su motor de presentación. Como se explico previamente aquí anteriormente con referencia a la figura 2, los nodos del árbol de objeto presentador

están determinados por los nodos del árbol de elemento gráfico asociado con una vista particular. Los elementos gráficos dentro de una vista están determinados por el elemento gráfico raíz especificada para la vista, sus hijos, y sus nietos, etc.

El conjunto de presentadores 302 en la vista unos están dispuestos jerárquicamente. Un presentador de raíz P1 tiene dos presentadores hijos P2 y P4 (Q4 es un presentador de adorno en el presentador P4). El presentador P2 no tiene hijos. El presentador P4 sin embargo tiene dos presentadores hijos P5 y P7. El presentador P5, a su vez, tiene los presentadores hijos P8a y P8b que están asociados con un elemento gráfico único E8. Tal compartir/dividir del presentador gráfico único entre dos objetos presentadores surge, como vía de ejemplo, cuando un elemento gráfico de texto E8 es dividido en dos columnas de un elemento gráfico texto pariente o padre E5, el cual tiene un presentador (P5) del tipo TextPresenter. El presentador P5 inicia un proceso de división de columna cuando el presentador P5 nota una propiedad sobre E5 que solicita el contenido de E5' (E8) para ser dividido en dos columnas. Los

presentadores P8a y P8b corresponden a la existencia gráfica de partes del elemento gráfico E8 en las dos columnas.

Como se ha notado previamente que un elemento gráfico único puede tener objetos presentadores asociados múltiple. Tres casos ilustrativos han sido identificados. En el primer escenario, se ha mostrado por las vistas múltiples 202a y 202b y los sistemas presentadores correspondientes 200a y 200b en la figura 3, un elemento gráfico único (por ejemplo, E2) tiene presentadores correspondientes P2 y P2' y asociados con las vistas 202a y 202b, respectivamente. Tales vistas corresponden por vía de ejemplo, a una vista miniatura y una vista de tamaño completo de una imagen fotográfica (E2) visualizada o mostrada en regiones diferentes de una interface de usuario generada por una aplicación de procesamiento de imagen fotográfica. En este caso, los presentadores P2 y P2' están creados para el elemento E2 en cada vista 202a y 202b que contiene el elemento. En un segundo escenario que involucren presentadores múltiples para un elemento único (por ejemplo, el elemento de texto E8), los elementos presentados P8a y P8b corresponden a

las dos columnas dentro de un rectángulo definido por un elemento presentador único P5. Alternativamente, tales casos de presentadores múltiples en una vista que referencia un mismo elemento gráfico surge por ejemplo, cuando un padre lleva a cabo intentos múltiples de diseñar una vista y guarda los arboles. En un tercer escenario, un presentador (por ejemplo, Q4) que es dependiente de otro presentador (por ejemplo, P4), en oposición al estado de datos de un elemento gráfico (por ejemplo, E4) es encadenado a otro presentador por los tanto enriqueciendo los rasgos de visualización suministrados por el presentador original. Un ejemplo del encadenamiento del presentador es un presentador de borde que suministra un adorno de borde aun presentador principal.

Como se ha mencionado anteriormente a la figura 2, esos objetos encaminadores hijos derivados de la clase encaminadora hijo 222 son envolturas para cada objeto representador hijo de un objeto presentador papá. En la ilustración gráfica de la estructura de árbol presentador para la vista 202a, los objetos encaminador hijo están representados por las cabezas de flecha en las líneas que conecta un presentador padre a sus objetos presentador

hijo. Los objetos encaminador hijo limitan el acceso para un objeto presentador padre a sus objetos presentador hijo.

Los manejadores de notificación (Nx) son responsables de notificar a los presentadores correspondientes (Px) de cambios en estados de datos de sus elementos gráficos (Ex). Los manejadores de notificación Nx no son esenciales a la operación del sistema de administración presentación/diseño. En lugar de eso, ellos facilitan las actualizaciones incrementales a menos de todos los presentadores cuando un cambio ocurre en un elemento dentro de una vista.

Con referencia a las figuras 4a y 4b, la clase base presentadora incluye un conjunto de métodos (propiedades y campos). En una modalidad de la invención, las propiedades son accesada en un objeto presentador referenciado particular por medio de invocar las operaciones get/set en el objeto. Una propiedad de fondo 400 especifica un valor de pintura para el fondo del presentador. Una propiedad de opacidad de fondo 402 almacena un valor que especifica una opacidad para el fondo. Una propiedad de borde 404 almacena un valor que

especifica el espesor de un borde dibujado alrededor del borde del espacio de diseño del presentador. La propiedad de borde 404 almacena por ejemplo, cuatro valores que indican los anchos del borde para los lados superior, inferior, derecho e izquierdo de un rectángulo asignado a un presentador. Una propiedad de color de borde 406 especifica un valor que designa el color del borde alrededor de un presentador. Una propiedad de estilo de borde 408 almacena un valor que indica un conjunto de estilos de visualización del borde. Ejemplos de estilos de bordes incluyen: sólido, arenoso, con bordes, curvado adentro y curvado hacia fuera.

Una propiedad de caja de enlace 410 especifica el área (por ejemplo, rectángulo) en el cual los gráficos presentadores (o hijos) son dibujados. El tamaño por defecto de la propiedad de caja de enlace 410 es el mismo que un tamaño especificado para un campo de tamaño de diseño 460. Sin embargo, el tamaño de la caja de enlace se puede exceder el tamaño de diseño permitiendo así que el presentador dibuje por fuera del tamaño del diseño.

Por otro lado, si una propiedad de clip 414 se establece como verdadera, entonces el presentador (y sus

presentadores hijos) no pueden ser dibujados por fuera de un rectángulo especificado por el campo de tamaño de diseño 460. Si la propiedad clip 414 se establece como verdadera, entonces la propiedad de caja de enlace 410 no es más grande que el tamaño del diseño. Esto asegura que el presentador dibujado no convierte los gráficos por fuera de su espacio de diseño asignada.

Una propiedad hijo 412 es un arreglo, o alternativamente cualquier otra estructura de datos de multielementos adecuada, que almacena referencias (por ejemplo, manejadores, apuntadores, directo/indirecto etc.) ha presentadores hijos anidados dentro de un presentador padre y sus localizaciones transformadas dentro del espacio de diseño del presentado papá. La propiedad hijo 412 es poblada en un primer uso en el contexto de una llamada del método OnUpdate (discutido aquí más abajo) sobre el presentador. "El primer uso" se refiere a cuando el presentador padre secuencialmente solicita la colección hijo y en la colección hijo es poblado por presentadores que corresponden a elementos gráficos hijo, del elemento gráfico correspondiente al presentador papá, desde el árbol de elementos gráficos en el almacén de respaldo (por ejemplo, el almacén de

respaldo 208). La propiedad hijo 412 facilita establecer/mantener una jerarquía que refleje las relaciones de anidamiento con respecto a los presentadores en la vista. En una modalidad de la invención, las entradas del arreglo se referencia a los objetos encaminadores hijo. Los objetos encaminador hijo son objetos envoltura para los objetos presentador hijo que están anidados dentro del espacio de visualización asignado a un objeto presentador papá. La envoltura de objeto encaminador hijo cumple la necesidad de encapsulación y seguridad (por ejemplo, un presentador hijo limita el acceso a un presentador padre a su contenido) la propiedad clip 414, como se explico anteriormente, controla si los contenidos de un presentador estarán confinados, por recorte, al tamaño de diseño de un presentador según se especifico en el campo de tamaño de diseño 460.

Un conjunto de propiedades designa el espacio asignado al contenido dentro del espacio de un presentador. En adición a la propiedad de borde 404, una propiedad de almohadilla 416 especifica una cantidad de espacio al interior del borde que es dejada sin ocupar por el contenido. Una propiedad de altura de contenido

418 especifica un valor que indica la altura del contenido dentro de un presentador, la altura de la porción del presentador que permanece después de tener en cuenta los valores especificadores en la propiedad de borde 404 y la propiedad de almohadilla 416. Una propiedad de ancho de contenido 420 especifica un valor que indica el ancho asignado al contenido dentro de un presentador después de tener en cuenta la propiedad de borde 404 y la propiedad de almohadilla 416 en sus dimensiones. La propiedad de altura del contenido 418 y la propiedad de ancho de contenido 420 en combinación especifican el área de un presentador que es ocupada potencialmente en un presentador por el contenido.

Una propiedad de altura 420 y una propiedad de ancho 424 especifican la altura real y el ancho real de un presentador, y estos dos valores incluyen el contenido, las dimensiones de contenido, almohadilla y borde. Una propiedad de altura por defecto 422 y una propiedad de ancho por defecto 424 especifican los valores por defectos asignables al presentador cuando la propiedad de altura 422 y la propiedad de ancho 424 no se han suministrado para un objeto presentador específico. Una propiedad de altura de ancho máximo 430 y una

propiedad de ancho mínimo 432 especifican límites para el valor asignado a la propiedad de ancho 424.

Todavía otra propiedad dimensional para presentadores es una propiedad de margen 434. La propiedad de margen 434 especifica un espacio de respaldo al exterior de los bordes del presentador. La propiedad de margen 434 es especificada por un presentador hijo y es accesada por un presentador padre cuando la disposición de un diseño comprende una jerarquía de presentadores desanidados. Una propiedad de visibilidad 436 especifica el estado de visibilidad del presentador. En la propiedad de visibilidad 436 es referenciada o esta referenciada para determinar como (o si) una imagen gráfica correspondiente al presentador y sus hijos es convertida. En una modalidad de estados de visibilidad potencial de la invención incluyen: visible, colapsada, y escondida.

Un numero de propiedades presentadoras son utilizadas para enlazar los presentadores a otras entidades en el sistema de administración de diseño o representación. Una propiedad presentadora 438 especifica el tipo de presentador que es usado para diseñar y

convertir el elemento gráfico. El tipo de presentador es inicialmente establecido en un elemento gráfico sobre un elemento gráfico, y el sistema presentador 200 lee la propiedad presentadora 438 desde el elemento gráfico y deriva un objeto presentador correspondiente al tipo presente identificado por la propiedad presentadora 438. La propiedad presentadora 438 es pasada a un derivado que, a su vez crea una derivación del tipo del presentador identificado. Una propiedad IsMainPresenter 440 especifica si el presentador es el único presentador o el principal (por ejemplo, últimos) en un conjunto de presentadores encadenados asociados con un objeto de elemento gráfico. Sin referirnos a la figura 3, el presentador P4 es el presentador principal para la cadena que incluye B4 y P4. En una modalidad de la invención, la propiedad IsMainPresenter 440 es una propiedad tipo Booleano.

Una propiedad de propietario de elemento 442 identifica el elemento gráfico para el cual el presentador fue derivado. Haciendo referencia nuevamente a la figura 3, las líneas fantasma que conecta los presentadores (Px) a los elementos gráficos correspondientes (Ex) representa las asociaciones

designadas por el contenido de la propiedad de propietario de elemento 442. Una propiedad de contexto de diseño 444 especifica un contexto global que facilita la mensajería interpresentador durante el procesamiento de diseño. Los valores almacenados en la propiedad de contexto de diseño 444 permiten que los presentadores pasen los mensajes independientemente del sistema presentador 200. Una propiedad de manejador de notificación 446 especifica una referencia aun manejador de notificación para el presentador. Una propiedad de tipo de manejador de notificación 448 especifica un tipo de un manejador de notificación referenciado por la propiedad de manejador de notificación 446.

Una propiedad de alteración 450 especifica si un presentador particular ha sido cambiado y por lo tanto requiere un proceso de actualización. El sistema presentador 200 y los componentes asociados de la arquitectura de administración de diseño soportan un procesamiento incremental de un diseño. La propiedad de alteración 450 mantenida por cada presentador se indica sobre una base presentador por presentador de si un presentador requiere ser actualizado.

Con referencia continuada a la figura 4a, un conjunto ilustrativo de campos de la clase base presentadora se resumen. En una modalidad de la invención, los campos son campos de tiempo de corrimiento de lenguaje común (CLR) (similar a los miembros bien conocidos variables en las clases C++). En una modalidad de la invención, las propiedades descritas anteriormente son similares a los campos. Sin embargo, las propiedades sobre los elementos gráficos (por ejemplo elementos gráficos 206) pertenecen y son manejadas por el almacén de respaldo (por ejemplo, el almacén de respaldo 208). Cuando una propiedad cambia, el almacén de respaldo notifica al motor de presentación el cambio. El motor de presentación (por ejemplo, el motor de presentación 212 para el sistema presentador 200) a su vez notifica a los manejadores de notificación apropiados 224. Así, en el caso de una propiedad, una notificación de cambio es recibida por vía de uno de los manejadores de notificaciones anteriormente mencionados. Los campos, por otro lado, no son manejados por el almacén de respaldo 208 (y son en su lugar manejados por ejemplo por el CLR). Por lo tanto, a ningunas notificaciones de cambio son disparadas al motor de presentación cuando hay cambios de campo.

El campo de tamaño de diseño 460 mantiene las dimensiones de un espacio de diseño asignado al presentador. El campo de tamaño diseño 460 es utilizado para disponer los presentadores vecinos en un diseño de interface página/usuario. Un campo de diseño de afectación 462 es una colección de propiedades (por ejemplo, bordes, almohadillas, etc.) de un presentador que, cuando se cambia afecta el diseño del presentador. Un campo de diseño de afectación 464 es una colección de propiedades (por ejemplo, márgenes, visibilidad, etc.) de un presentador que, cuando es cambiado por un elemento (por ejemplo el elemento A), las alteraciones (es decir afectaciones potenciales) del diseño de un primer padre del elemento (elemento A) tiene un presentador especificado. Una campo de conversión de efectos 466 es un mecanismo de notificación que dispara la invalidación y re-conversión del presentador.

Haciendo referencia ahora a los métodos de la clase base presentadores 206 listados en la figura 4b, un método OnUpdate 470 lleva a cabo las tareas de medición en un presentador para un elemento. Los autores del presentador sobre escriben una versión de clase base en

el método OnUpdate 470 para crea/suministrar un diseño adaptado en tamaño/posicionamiento para una vista correspondiente de un elemento.

Una primera acción por defecto del método OnUpdate 470 es el dimensionamiento del presentador. El método OnUpdate 470 recibe las dimensiones del tamaño del presentador en un parámetro BoxSizeInfo pasado. El método OnUpdate 470 establece las dimensiones especificadas en el campo de tamaño de diseño 460 de acuerdo con el BoxSizeInfo especificado y la propiedad de ancho por defecto 428 y la propiedad alto por defecto 426. Fuentes adicionales de información de dimensionado/posicionamiento para un presentador se incorporan en la arquitectura del procesamiento de diseño/presentación con base en presentador descrita, y tales fuentes son utilizadas en modalidades alternativas de la invención.

Una segunda tarea por defecto del método OnUpdate 470 es posicionar los presentadores hijo dentro del espacio de dimensiones locales del presentador. La versión clase base del método OnUpdate 470 e interactivamente llama al método OnUpdate 470 sobre los

presentadores hijo identificados en la propiedad de hijo 412 para descubrir sus dimensiones e hijos. Luego de descubrir y medir los presentadores hijos, los hijos son posicionados dentro del presentador padre invocando un método de transformación (discutido más adelante) sobre los hijos individuales. Las llamadas interactivas al método OnUpdate 470 en los presentadores hijos finalizan cuando no hay más hijos especificados en un presentador. En adición al registrar su tamaño en el campo de tamaño de diseño 460 e interactivamente llamar/posicionar los presentadores hijos, el método OnUpdate 470 regresa un valor al llamador indicando si la reconversión es deseada en vista de la llamada al método OnUpdate 470.

Un método de registro de átomo 471 registra un nombre de hijo para el presentador para el contexto de diseño 244 lo mismo que cualquier otra listas apropiadas de presentadores incluyendo aquellas mantenidas por las extensiones de los presentadores. El método de átomo de registro 471 es llamado, Por ejemplo, para comunicarse con otro objeto presentador en una manera no estándar (por ejemplo, una manera no corrientemente soportada por la interface de programa de aplicación presentadora) las presentadoras registran un átomo y entonces establecen el

objeto con quienes ellos desean comunicase usando el átomo. Por ejemplo, un presentador hijo desea comunicar un texto descendiente de un presentador papá. El presentador hijo registra un átomo marcado/etiquetado "descendiente" y recibe una identificación para el átomo. El hijo computa o calcula el descendiente y establece el valor en el átomo usando la identificación. El presentador pariente entonces usa la identificación (que es también recibida por el registro "descendiente") para recuperar la información almacenada.

Un método `OnUpdateBoundingBox` 472 produce una propiedad de caja de ligado actualizada 410 para un presentador. El método `OnUpdateBoundingBox` 472 es llamado si el método `OnUpdate` 470 regresa a un valor (por ejemplo "verdadero") indicando que una reconversión es requerida. La implementación por defecto del método `OnUpdateBoundingBox` 472, suministra en la clase base presentadora, barridos de los presentadores hijos identificados en la propiedad hijos 412 para el presentador y combina (es decir, lleva a cabo una unión matemática de rectángulos para) las propiedades de la caja de ligado de los hijos transformados con las dimensiones del presentador almacenado en el campo de

tamaño de diseño 460. La implementación de la clase base del método `OnUpdateBoundingBox` 472 también accesa la propiedad `clip` 414 y si este es "verdadero" ignora las cajas de ligados de los hijos. En su lugar los presentadores hijos serán recortados al campo de tamaño de diseño 400 a las dimensiones del campo de tamaño diseño 460 de su presentador papá. Los presentadores que dibujan al exterior de sus dimensiones de campo de tamaño de diseño 460 sobre escriben este método. Tales clases presentadoras adaptadas llaman la clase presentadora base y agregan un "área de entintado" adicional en su código de sobre escritura. En una modalidad de la invención, los presentadores se les permiten estimar un "área de entintado" que sea más grande que la requerida realmente. Si el área de la propiedad de caja de ligado 410 es más pequeña que le área requerida para dibujar el presentador, entonces el dibujo es recortado con base en las dimensiones de la propiedad de caja de ligado 410 especificada para el presentador.

Un método `OnRender` 474 es invocado luego de que los presentadores que solicitan/requieren una reconversión. El método `OnRender` 474 es el llamado principal en un presentador para reconvertir el

presentador y cualquiera de sus presentadores hijos. El método OnRender 474 es invocado preferiblemente luego de que el método OnUpdate 470 es invocado en todos los presentadores "alterados" en un diseño (por ejemplo, una escena). En una modalidad de la invención, el metodo OnRender 474 es invocado en objetos presentadores con base en si el método OnUpdate 470 llama a un objeto presentador que regresa una indicación que el presentador requiere una reconversión. En una modalidad de la invención, el metodo OnRender de clase de base 474 lleva a cabo unas operaciones de disposición de presentación/diseño independientes del dispositivo en un presentador. Otros componentes, incluyendo el subsistema de gráficos y los excitadores de dispositivos gráficos 234, convierten la vista del presentador en un formato específico de dispositivo. Las derivaciones adaptadas del método OnRender 474 incluyen llamadas a los componentes de generación gráfica del sistema del computador para convertir la salida gráfica de la interface documento/usuario.

En una modalidad de la invención, una versión de clase de base del metodo OnRender 474 convierte los contenidos de un presentador incluyendo cualquier hijo en

tres etapas de procesamiento de diseño. Durante una etapa `OnRenderBeforeChildren`, el objeto presentador invoca las operaciones designadas para procesar antes de procesar cualquiera de sus presentadores hijos. En una modalidad de la invención, la implementación de clase base del método `OnRender` 474 suministra un enganche para adaptar y no especifica ningunas operaciones de procesamiento durante la etapa `OnRenderBeforeChildren`. Sin embargo, un remplazo adaptado ilustrativo para este comportamiento por defecto es establecido/dibujado en un fondo adaptado para el área de despliegue del presentador.

La versión clase de base del método `OnRender` 474, durante una etapa de conversión de hijo, accesa los presentadores hijo del presentador identificado en la propiedad hijo 412. El método `OnRender` 474 invocado a su vez sobre cada una de los presentadores hijo (al grado requerido por el hijo cuando el método `OnRender` 470 fue llevado a cabo). Así, una llamada de alto nivel al método `OnRender` 474 deriva en cascada presentadores hijos, y sus hijos, hasta que el final del árbol ha sido alcanzado (es decir, el presentador ya no tenga hijo). Cuando la llamada de alto nivel esta completa, todos los

presentadores hijos en un árbol que han requerido reconversión han sido reconvertidos.

Durante una etapa `OnRenderBeforeChildren`, el objeto presentador invoca las operaciones que deben ser ejecutadas después de llamar el método `OnRender` 474 sobre presentadores hijos identificados en la propiedad `hijo` 412. A la versión clase de base en el método `OnRender` 474 suministra un enganche para adaptar y no especifica ningunas operaciones de post-procesamiento durante la etapa de `OnRenderBeforeChildren`. Sin embargo, los ejemplo de adaptaciones incluyen anotaciones adornos etc.

Si un presentador no reconvierte nada diferente a sus hijos, no hay necesidad de sobrescribir la versión de clase de base del método `OnRender` 474. sin embargo, si cualquier cosa ha sido dibujada por el presentador mismo, entonces la clase de base para el método `OnRender` 474 es sobrescrita para convertir datos gráficos dentro del espacio asignado al presentador. El código de reemplazo incluye, como vía de ejemplo, llamadas a las aplicaciones 230 y/o el subsistema de gráficos y los excitadores de dispositivos gráfico. Un presentador encadenado está en el árbol de presentadores para una vista particular. Si

el presentador encadenado requiere reconversión, entonces su método `OnRender` será invocado por el sistema presentador de una manera similar al `OnRender` para el presentador principal.

En el contexto de esta aplicación un "ensayo de golpe" se refiere a un proceso para identificar que presentador cae bajo un punto de rejilla particular en un espacio de rejilla de interface de usuario gráfico, por ejemplo, un ensayo de golpe es usado para localizar en donde está localizado el apuntador del ratón (por ejemplo), sobre el cual el presentador está sentado) cuando un usuario selecciona uno de los botones del ratón. Un método `OnHitTestDetail` 476 regresa los datos almacenados dentro de la estructura `HitTestDetail` para un presentador. Un clase presentadora adaptada sobrescribe la implementación de la clase de base del método `OnHitTestDetail` 476 para llena la estructura `HitTestDetail` para permitir que el presentador regrese los datos diferentes del elemento y el presentador que fue golpeado, por lo tanto suministrar, por ejemplo, una información más detallada con respecto al presentador de modo que una parte del presentador que fue realmente

golpeada (por ejemplo una posición de un carácter para un texto).

En método `OnCreateViewResult` 478 se expone los parámetros que describen el estado calculado de un presentador en un objeto resultado de una vista. Los usuarios solicitan el objeto resultado de la vista para cualquier elemento en el contexto de una vista designada. La versión base de clase del método `OnCreateViewResult` 478 suministra, por medio de un objeto de resultado de vista, la altura, ancho, la posición de la esquina superior izquierda de un presentador. Si información adicional /alternativa con respecto al estado del presentador se desea, tal como muchas líneas de texto contenidas en el presentado, entonces la versión base de clase en el método `OnCreateViewResult` 478 es sobrescrita para suministrar la información deseada por medio del objeto de resultados de vista.

Un método de ensayo de golpe 480 inicia un ensayo de golpe sobre el subárbol de raíz en un presentador especificado. Las coordenadas de un punto (en las coordenadas locales del presentador) son pasadas al método ensayo de golpe de 480 luego de lo cual el ensayo

de golpe se lleva a cabo. El método de ensayo de golpe 480 regresa los datos del ensayo de golpe que contiene los resultados del ensayo de golpe para el punto designado y el presentador. Los resultados contienen todos los golpes visuales por el punto. Una visual es un objeto base (por ejemplo, una línea) a partir de la cual un presentador deriva una imagen de visualización. Un visual es un objeto gráfico que tiene la capacidad de dibujar realmente una imagen sobre la salida de visualización gráfica, en una llamada del presentador OnRender, varias llamadas son emitidas a una visual del API para convertir, por ejemplo, texto, imágenes, video, etc.

Un método OnGetBypassList 482 suministra una lista de presentadores hijo previamente calculada para el presentador identificado. La lista de presentadores hijo facilita el sobrepasar a los presentadores hijos durante el cálculo del diseño (por ejemplo comparar un nuevo hijo de diseño con un hijo de diseño previamente calculado de los presentadores y reutilizar uno calculado previamente si existe un case de uno en la lista). Si un presentador descubre su hijo por referencia a la colección de propiedades hijo 412, este no tiene sobrefectuar este

método. Sin embargo si el presentador usa un método de `GetChildProxyForElement` 486 (descrito más abajo), el método `OnGetBypassList` 482 es generalmente pasado por alto.

En una aplicación de un sistema de diseño de la presente invención, en el caso en donde el método `GetChildProxyForElement` 486 es llamado, el método `OnGetBypassList` 482 es llamado antes del método `OnUpdate` 470, y un objeto presentador crea y pobla una lista de arreglos con objetos encaminadores hijo (descrito aquí más abajo con referencia a la Figura 5) que son capturados de un cálculo de diseño previo. Si un objeto presentador no tiene una captura para almacenar los objetos encaminadores hijo recalculados, todos los presentadores hijo del objeto presentador son recalculados en cada llamada del método `OnUpdate` 470 en el presentador.

Un método de ligado de calculo 484 calcula y regresa los límites de un subgráfico de raíz de una visual identificada.

Un objeto encaminador hijo, como se explicó anteriormente es un objeto de envoltura que actúa como un intermediario entre un presentador padre y un presentador hijo para un elemento gráfico particular. Los objetos encaminadores hijo para los presentadores hijo de un objeto presentador particular se referencian como una colección dentro de la propiedad hijo 412 del objeto presentador. El método `GetChildProxyForElement` 486 es un método en objeto presentador padre que crea un objeto encaminador hijo y objeto presentador para un elemento especificado. El método, regresa "nulo" si el elemento especificado no tiene un presentador.

Un método `OnBeforeBypass` 488 es llamado cuando un sistema presentador 200 quiere "evitar" un presentador particular (es decir, escapar un llamada al método `OnUpdate` del presentador 470). Tal evitado ocurre, por ejemplo, cuando el presentador no está alterado ni el método `OnUpdate` 470 llama a los parámetros de dimensión de presentador de entrada (`BoxSizeInfo`) no han cambiado desde la última vez que el método `OnUpdate` 470 llamó al presentador. El método `OnBeforeBypass` 488 es un mecanismo promotor del comportamiento que habilita los presentadores, que tiene su propia colección adaptada de

objetos contaminadores hijo (que usan el método `GetChildProxyForElement` 486), para reutilizar los objetos encaminadores hijo de sus hijos cuando una actualización se lleva acabo en un diseño (por ejemplo, cuando solamente una porción pequeña de un diseño es afectada por un cambio).

Un método `OnDisconnectChildren` 490 remueve todos los objetos presentadores hijo de un conjunto de presentadores hijos referenciados en la propiedad hijo 412 para un presentador especificado. Si la propiedad hijo por defecto 412 es pasada por alto para una clase presentadora derivada de la clase base presentadora 216, entonces el método `OnDisconnectChildren` 490 es también pasado por alto para asegurar la disposición adecuada de los presentadores hijo.

Un paso Min/Max es un paso de dimensionamiento especial usado por presentadores de partición (por ejemplo, `DockPresenter`) cuando el dimensionamiento del hijo al contenido. Conociendo los anchos mínimo y máximo del hijo le permite al padre distribuir de manera inteligente el espacio entre múltiples hijos que comparten una dimensión de ancha particular. Sin la

información Min/Max suministrada por la función MinMaxPass, un presentador de partición requeriría múltiples paso para dimensionar de manera inteligente al contenido. Un método OnMinWidth 492 es llamado después de un paso Min/Max. El método OnMinWidth 492 recolecta un valor mínimo de ancho que es un producto intermedio de un cálculo de ancho máximo. La versión de clase de base del método OnMinWidth 492 emite un mensaje "no importa" que invoca un paso de cálculo con ProposedSize=(0, infinito) y ambas direcciones "no fijadas".

Un método de presentador 494 es un método constructor para un derivado de presentador. El método presentador 494 no tiene parámetros. El método presentador 494 es invocado cuando el presentador es derivado por el sistema presentador 200. El método presentador 494 establece el derivado de presentador hasta permitirle invocar otros métodos (por ejemplo, OnUpdate) en un presentador.

Un método QueueLayoutTask 496 agrega un tarea de diseño especificada, que va ser llevada a cabo con respecto al objeto presentador, a una fila de tareas de diseño que son ejecutadas por el sistema presentador 200.

Cuando la tarea de diseño especificada alcance el frente de la fila, el sistema presentador 200 llama un método OnLayoutTask 604 (descrito aquí enseguida) en el manejador de notificación para el presentador.

Un método OnQueryValue 498 regresa un valor correspondiente aun valor especificado calculado para el presentador.

Habiendo descrito la clase base de presentador 216, se dirige ahora la atención a la Figura 5 que suministra un resumen de las propiedades y métodos de una modalidad ilustrativa de la clase encaminadora hijo 222. Como se explicó previamente, en una modalidad de la invención, los objetos encaminadores hijo son envolturas para los objetos presentadores que son hijos de los objetos presentador papá, y soportan acceso limitado por el padre a los recursos de los objetos presentador hijo. Todas las comunicaciones (requisiciones/respuestas) entre un objeto presentador padre y un objetos presentador hijo son enrutados por medio del encaminador para el presentador hijo. El objeto encaminador hijo mantiene una referencia al presentador hijo y enruta las

llamadas/requisiciones del presentador padre al presentador hijo.

En una modalidad del sistema presentador 200, la clase encaminadora hijo 222 incluye una propiedad de pertenencia de elemento 500 que especifica en los elementos gráficos 206 con los cuales el presentador referenciado del encaminador hijo está asociado.

Una propiedad de transformación 502 posiciona el objeto presentador hijo (y sus visuales asociadas) al interior de su objeto presentador papá. Una transformada es una matriz de transformación de gráficos estándar 3x3 en una clase de base de objeto visual. Cuando la matriz de transformación se establece, un objeto visual, asociado con el elemento gráfico del presentador, coloca su hijo en un desplazamiento especificado por la matriz de transformación.

Una propiedad de caja de límites 504 especifica los límites de conversión de un presentador hijo. Los límites de conversión de un presentador hijo pueden por lo tanto digerir del tamaño del diseño del presentador hijo.

Una propiedad de alteración 506 especifica el estado de alteración del presentador del encaminador hijo. En una modalidad del sistema presentador 200, el estado de alteración es: limpio, alterado parcialmente o alterado. Un estado limpio indica que no hay cambios que hayan ocurrido en el presentador hijo ya que desde la última actualización de la vista. Un estado alterado indica que un cambio ha ocurrido en el presentador hijo desde la última actualización de la vista. Y un estado de alteración parcial indica que por lo menos uno de los hijos pertenecientes al presentador de hijo está alterado, pero el presentador hijo en si mismo no está alterado. Tales estados permiten que los presentadores padre optimicen las operaciones de actualización.

Un método QueryDefaultSizeInfo 510 inicializa las dimensiones de un presentador hijo a partir de valores por defecto suministrados por el presentador padre del presentador hijo.

Un método Update 520 solicita a un encaminador hijo invocar el método OnUpdate 470 sobre el objeto presentador hijo. El presentador padre llama el método

Update 520 en el encaminador hijo cuando el presentador padre busca calcular el diseño del presentador hijo. La llamada de método Update 520 incluye dos parámetros pasados. La información de tamaño de caja especifica las dimensiones de la caja dentro de la cual el hijo es calculado en el diseño, y, si la paginación está activada el presentador padre pasa la información de la página en un parámetro de descriptor de página (ver Fig. 10 descrita aquí más abajo). El método de actualización 520 suministra así una capa entre el que llama y un objeto presentador hijo que permite un tamizado/filtro de tales solicitudes a un objeto presentador hijo. El método de Update 520 suministra una manera de pasar por alto las llamadas normalmente enrutadas al método OnUpdate en el presentador hijo. Por ejemplo, la llamada del método Update 520 es invocada en casos en donde el presentador hijo no ha cambiado y los parámetros de entrada (información de tamaño de caja y descripción de página) no han cambiado.

Un método de sujeción 530, emitido por un presentador padre a un encaminador hijo, invoca un comando OnRender en un presentador hijo asociado con el encaminador hijo.

Un método de valor de pregunta 540 suministra un valor de presentador correspondiente a un tipo de valor calculado (propiedad) identificado en el método de llamada. La llamada del método de valor de pregunta 540 es delegada al método OnQueryValue 498 correspondiente en el objeto presentador hijo.

Un método del tamaño del diseño de solicitud 550 suministra parámetros de tamaño del diseño calculado. La llamada del método de tamaño de diseño de solicitud 540 es delegado al método OnQueryValue 496 correspondiente en el objeto presentador hijo. Un método de computo MInMax 56 suministra un ancho mínimo y máximo de un presentador hijo.

Haciendo referencia a la Figura 6 el conjunto de clases de bases asociados con el sistema presentador 200 incluyen una clase de base manejadora de notificación. Como se explicó anteriormente aquí, los objetos manejables de notificación reciben todas las notificaciones con respecto a cambios que se efectúan potencialmente a un presentador asociado con una vista particular, información alterada acumulada y determinar

la facilidad de si un objeto presentador asociado requiere actualización. Ambos métodos en la clase de base de manejador de notificación reciben como un parámetro, un objeto que incluye un método que suministre información relevante en los elementos gráficos) por ejemplo, los elementos gráficos 206) con los cuales el manejador de notificación está asociado.

La clase base de manejador de notificación incluye un método OnNotify 600 que suministra una notificación de la alteración de un objeto presentador asociado particular. El método OnNotify 600 potencialmente incluye tareas de notificación derivativa adaptada adicional (pasando por alto la acción por defecto de suministrar un valor de propiedad de alteración) que son ejecutadas en respuesta a la recepción de una notificación. El método OnNotify 600 recibe como información de parámetro de entrada alrededor de un cambio que ha resultado en una notificación. El método OnNotify regresa un valor Booleano que indica si o no el manejador de notificación puede procesar el cambio por si mismo (o si es necesario notificar al manejador de notificación para que un presentador ancestro en el árbol presentador para una vista). El método también puede

regresar un valor de alteración indicando si el presentador está "limpio", "alterado", o un "alterado parcialmente".

Un método OnLayoutTask 602 ejecuta cualesquier tareas previamente en fila por un presentador asociado por medio del QueueLayoutTask 496.

Haciendo referencia a la Figura 7, el conjunto de clases de bases asociados con el sistema presentador 200 incluye adicionalmente una clase de sitio de manejador de notificación. La clase de sitio de manejador de notificaciones es un objeto de sitio pasado a un objeto manejador de notificación que permite que el manejador de notificación invoque los servicios de notificación soportados por el sistema presentador 200.

La clase de sitio de manejador de notificación incluyen una propiedad de propietario del elemento 700 que especifica el objeto elemento gráfico con el cual el objeto de sitio de manejador de notificación está asociado.

Un método de átomo de registro 702 registra un nombre y lo suministrado dentro del contesto de diseño 444 para un manejador de notificación. La funcionalidad del método de átomo de registro 702 es similar al método de átomo de registro 471 descrito aquí anteriormente con referencia a la clase base presentadora. El método de átomo de registro 702 es por lo tanto un medio para comunicar solicitudes o consultas por medio de un manejador de notificación a otros objetos (por ejemplo, objetos presentadores).

Un método de notificación de crear 704 crea un nuevo objeto de notificación presentador e inicializa sus campos con base en parámetros pasados. Los parámetros pasados incluyen una identificación de un tipo de átomo (indicando el tipo de objeto de notificación de presentador y los datos de notificación.

Un método de notificar descendientes 706 invoca el método OnNotify 600 en los manejadores de notificación asociados con los descendientes de un presentador, con los cuales el sitio de manejador de notificación está asociado, para pasar una notificación de presentador especificada. El método de notificar descendiente 706

lleva a cabo la función de notificar los presentadores descendientes de cualesquier cambios a un presentador padre (por ejemplo redimensionamiento). Similarmente, un método de notificar ancestros 708 invoca el método OnNotify 600 en los manejadores de notificación asociados con los ancestros de un presentador, con los cuales el sitio de manejador de notificación está asociado, para pasar una notificación de presentador especificada. Los ancestros están determinados por medio de atravesar el árbol de elementos gráficos (ver Figura 3) en el almacén de respaldo 208. Los elementos gráficos representados en el almacén de respaldo incluyen una propiedad que especifica el padre (ancestro inmediato) del elemento gráfico. Un método de notifique todo 710 invoca el método OnNotify 600 en los objetos de manejador de notificación para todos los presentadores en un árbol presentador propiedad de una vista para pasar una notificación especificada. Finalmente, un método de auto notificación 712 suministra un medio para que un presentador invoque el OnNotify 600 para marcarse el mismo como alterado.

Volviendo a la Figura 8, una clase de objeto de vista ilustrativo está mostrado a partir del cual la vista 202 es derivada. Los objetos de vista poseen una

derivación del sistema presentador (por ejemplo el sistema presentador 200) y son la raíz visual de todos los visuales creados dentro del derivado del sistema presentador creado. El objeto de vista inicia todos los cálculos sobre los presentadores dentro de la vista. Un método de vista 800 es un método constructor que es usado para crear un nuevo objeto de vista. La vista recibe como un parámetro pasado una referencia a un elemento que es el elemento gráfico de raíz de la vista. La propiedad RootElement 802 regresa una referencia a un elemento gráfico de raíz para la vista (que fue establecida por medio del método de vista descrito anteriormente 800).

Un método DoLayout 804 crea un nuevo árbol presentador para la vista y regresa una referencia al nuevo árbol. El método DoLayout 804 recibe como parámetros de entrada: una altura y ancho propuestos para el rectángulo de vista los mismo que dos valores Booleanos (fixedwidth y fixedheight) que establecen si las dimensiones propuestas pueden ser alteradas durante la ejecución del método DoLayout 804.

Una propiedad ViewSize 806 establece las dimensiones de la vista. La propiedad ViewSize 806, una

propiedad de solo escritura es utilizada cuando el objeto vista es hospedado dentro de una ventana, y la propiedad ViewSize 806 representa el tamaño inicial desde el cual una operación de diseño es llevada a acabo.

Un método GetViewResult 808 recibe una referencia de elemento gráfico como una entrada, y el método GetViewResult 808 regresa en la vista resultante para el elemento gráfico. Un valor nulo es regresado en el evento de que el elemento gráfico no sea localizado dentro del conjunto de elementos gráficos para la vista.

Un método HitTest 810 lleva a acabo un ensayo de golpe a un punto pasado dentro de un sistema de coordenadas en la vista. El método HitTest 810 regresa un primer objetivo de ensayo de golpe no transparente. Si ningún elemento gráfico es golpeado, entonces el método regresa "falso"

Un método CreatePage 812 recibe como parámetros de entrada BoxSizeInfo (ver Figura 9 descrita aquí más adelante) y PageDescriptor (ver Figura 10 descrita más adelante). El método CreatePage 812 crea una nueva página, con base en los parámetros de entrada, y regresa

el objeto encaminador hijo correspondiente al elemento gráfico de raíz para la página.

Volviendo a la Figura 9, un a estructura `BoxSizeInfo` ilustrativa se demuestra. La estructura `BoxSizeInfo` especifica un conjunto de dimensiones para presentadores y sus presentadores hijo dentro de una vista, y como estas dimensiones deberían ser tratadas por el receptor. Una propiedad de tamaño propuesto 900 establece el tamaño propuesto para el presentador hijo dentro de la vista. Inicialmente, el valor de la propiedad de tamaño propuesto 900 viene de las propiedades para un elemento gráfico hijo en un almacén de respaldo. Esto puede modificarse por un presentador padre antes de pasar a un hijo. La propiedad tamaño de padre 902 es un tamaño propuesto para un presentador papá. La propiedad tamaño de padre 902 es usada, como vía de ejemplo, para calcular dimensiones, a partir de valores en porcentaje del papá, para presentadores hijo. La estructura `BoxSizeInfo` también incluye dos propiedades de valor Booleano, el `FixedWidth` 904 y el `FixeHeiht` 906 que instruye al presentador receptor si la dimensión suministrada como un valor impuesto (fijo), o si el valor de dimensión es meramente una altura o ancho sugeridos.

Regresando a la Figura 10, partes de una estructura de descriptor de página ilustrativas se muestran. Un descriptor de página incluye una propiedad `PageSize 1000`. La propiedad `PageSize 100` es una propiedad de solo lectura que especifica el tamaño de una página. Esta vez inicializada cuando la estructura descriptor de página es creada. Una propiedad parada de registro 1002 es una estructura que incluye: un inicio de posición de carácter que identifica el primer carácter del presentador actual que está sobre la siguiente página (con respecto al elemento gráfico que es la raíz de una vista que está siendo paginada), un número de caracteres (contando hacia atrás desde el inicio de posición de carácter) que puede invalidar una página anterior; y un valor Booleano `IsDirty` que indica si el registro de parada (que cubre un rango de carácter particular dentro de un elemento gráfico paginado) está invalidado por un cambio en el almacén de respaldo.

La estructura de descriptor de página también incluye una propiedad `AvailableSize 1004`. La página completa puede no estar disponible a un presentador luego de comenzar un método `OnUpdate` sobre el presentador. La

propiedad AvailableSize 1004 indica el espacio restante de la página para un presentador. La propiedad AvailableSize 1004 es inicializada a un valor para un tamaño de página.

Volviendo a la Figura 11, un conjunto de métodos se han identificado que se llevan a acabo por el motor de presentación 212 del sistema presentador 200 creado por la vista 202 (ver Figura 2). Un método constructor 1100 sobre el motor de presentación 212 accesa la vista 202 que contiene una referencia al elemento gráfico de raíz para la vista (suministrada por el árbol de elementos gráficos 206. un método DoLayout 1102 recibe un ancho y un alto como parámetro pasados que representan las dimensiones de la vista 202 y, en respuesta, comienza a ejecutar un proceso de diseño con respecto a los contenidos de los elementos gráficos 206. un método UpdateLayout 1104, ejecutado después de ejecutar el DoLayout por lo menos un vez, comienza aplicar cambios incrementales a un diseño previamente calculado para la vista 202. un método CreatePage 1106 recibe, como un parámetro de entrada, la información de tamaño de caja (BoxSizeInfo) que contiene las dimensiones de una página que va a ser diseñada y un descriptor de página contiene

información con respecto a como se inicia la página particular. En vez de diseñar todos los contenidos bajo una raíz de los elementos gráficos 206 para la vista 202, el método CreatePage 1106 diseña solamente una página única de interés.

Una arquitectura de procesamiento de diseño ilustrativa ha sido descrita con referencia a las figuras 1 a 11. la arquitectura ilustrativa incluye estados de vista separados (objetos presentadores) enlazados con estados de datos de elementos (objetos de elementos gráficos). La arquitectura ilustrativa incluye un sistema presentador 200 que facilita la extensión de las capacidades de diseño/presentación de la vista por medio de una clase base de presentador 216 desde la cual nuevas clases objeto presentadoras, que incluyen nuevas funcionalidades de diseño/presentación se han derivado. El sistema y la arquitectura descritos aquí también incluyen un mecanismo de notificación para enlazar elementos a sus correspondientes objetos presentadores para invocar de manera selectiva porciones de recalculo de una vista en respuesta a cambios. Las clases de base objeto ilustrativas de muchas maneras potenciales alternativa de llevar a cabo la arquitectura extensible

que involucra la presente invención y que no debe ser vista como limitativa del alcance de la invención.

Habiendo descrito un conjunto de componentes que hacen una arquitectura de procesamiento de presentación/diseño ejemplarizante de la presente invención, se dirige ahora la atención a la Figura 12 que ilustra la progresión general de los cambios de procesamiento a un elemento mostrado gráficamente. Inicialmente, en 1200 el almacén de respaldo 208 cambia. Por ejemplo, la aplicación 204 ejecuta una operación que resulta en un estado y/o cambio de contenido en uno o más de los componentes mostrados en una interface de usuaria gráfica para la aplicación. En respuesta, en la etapa 1202 una notificación de cambio, que surge de un cambio a un elemento gráfico en el almacén de respaldo, es recibido por el motor de presentación 212 del sistema presentador 200.

Como vía de antecedentes, con respecto a la etapa 1204, el sistema presentador 200 mantiene asociaciones entre elementos y presentadores en una información elemento a presentador (EPI). Siempre que un presentador es creado para un elemento gráfico, una referencia a ese

presentador es creada sobre el elemento y es almacenada en el EPI. El EPI es mantenido por el sistema presentador. Si un elemento gráfico tiene más de un presentador, habrá múltiples referencias (una por presentador) en el EPI de los elementos gráficos. Cuando una propiedad cambia en un elemento gráfico, las siguientes etapas ocurren:

(a) El almacén de respaldo 208 notifica el cambio en el elemento gráfico,

(b) El almacén de respaldo 208 notifica al motor de presentación 212 del cambio, con referencia al elemento gráfico que ha experimentado un cambio en una de sus propiedades,

(c) El motor de presentación 212 camina el EPI del elemento gráfico que cambió y determina los presentadores que han sido creados para el elemento gráfico, y

(d) Una vez que el presentador este terminado el motor de presentación 212 obtiene una referencia al manejador de notificación correspondiente del presentador

(por medio de la propiedad del manejador de notificación 446 sobre el presentador).

Así, en vista de lo anterior, en la etapa 1204, el motor de presentación 212 del sistema presentador 200 determina la identidad de un presentador al cual la notificación de cambio aplica. Como se explicó anteriormente, el sistema presentador mantiene una lista de presentadores que fueron derivados para el elemento en su EPI. Después de determinar un presentador al cual la notificación de cambio aplica (asumiendo para este ejemplo que solamente un presentador único existe para elemento), el motor de presentación 212 del sistema presentador obtiene una referencia a un manejador de notificación repara el presentador por medio de la propiedad de manejador de notificación 446 sobre el presentador identificado.

De hay en adelante, durante la etapa 1206 el motor de presentación 212 del sistema presentador 200 adelanta la notificación de cambio al manejador de notificación usando la referencia obtenida durante la etapa 1204. en una modalidad de la invención, la etapa 1206 es llevada acabo por medio de invocar el método

OnNotify 602 sobre un objeto manejador de notificación particular.

Después de recibir la notificación de cambio desde el sistema presentador, durante la etapa 1208 (por ejemplo, durante la ejecución del método OnNotify 602) el manejador de notificación determina las acciones, si hay alguna, requeridas para el presentador en respuesta a los cambios, durante la etapa 1200, al elemento gráfico en el almacén de respaldo 208. En una modalidad de la invención, el manejador de notificación determina que porción, si hay alguna, de un presentador correspondiente requiere actualización en vista de la notificación de cambios recibida. Tales porciones incluyen, como vía de ejemplo, el hijo del presentador y presentados mismo.

Después, durante la etapa 1210 el manejador de notificación (por ejemplo, el método OnNotify 602) regresa un resultado que permite al presentador correspondiente determinar y ejecutar las acciones envista de la notificación de cambio recibida por el sistema presentador 200 durante la etapa 1202. en una modalidad de la invención, la forma del resultado retornado es manejador de notificación/presentador

específico. En algunos casos, el resultado retornado simplemente marca o etiqueta al presentador como "alterado", invocando así un recalcule completo del presentador correspondiente. En otros casos la información es suministrada para invocar selectivamente solamente porciones de la vista del presentador en sus capacidades de recalcule.

En una modalidad de la invención, el motor de presentación 212 del sistema presentador 200 recibe el resultado retornado desde el manejador de notificación para un presentador durante la etapa 1210. De hay en adelante, en la etapa 1212 si los cambios en el almacén de respaldo no se requieren una necesidad para recalcularse el presentador, entonces el control pasa al bloque de decisión 1216. Sin embargo si la etapa 1212 el resultado retornado indica que el presentador necesita ser recalculado, entonces el control pasa de la etapa 1214 en donde el sistema presentador 200 agrega al presentador al conjunto de presentadores "alterados" que requieren recalculación. En una modalidad ejemplar de la invención el sistema presentador establece la propiedad de alteración 450 en el presentador correspondiente (indicando una necesidad de recalcularse el presentador) y

agrega el presentador a un conjunto de presentadores que requieren ser recalculados. Los presentadores también especifican la información describiendo el grado de cambios que llevan al establecimiento de la propiedad de alteración 450. El control entonces pasa a la etapa 1216.

En la etapa 1216, si un presentador padre no existe para el presentador asociado con el manejador de notificación del interés presente, entonces el control pasa al final 1218. si un presentador padre si existe, entonces el control pasa desde la etapa 1216 a la etapa 1220 en donde una notificación es creada (pasar a) el manejador de notificación de presentador padre (resultando en el manejador de notificación para el padre procesando la notificación) como se indicó por la línea de regreso punteada a la etapa 1208.

Así, el conjunto de etapas ilustradas en la Figura 12 permite que el sistema presentador 200 acumule una lista de presentadores que han sido "alterados" como u resultado de cambio al almacén de respaldo 220 ya que el último calculo de un diseño para una interface documento/usuario.

Volviendo ahora a la Figura 13, un diagrama de flujo describe un proceso ejemplar para recalcular/reconvertir una interfaz documento/usuario de acuerdo con los cambios a los presentadores en una vista particular (ver Figura 12 descrita aquí anteriormente). En una modalidad de la invención, la aplicación 204 controla las actualizaciones de su vista 202 e invoca el procedimiento de actualización de vista, tal como el resumido en la Figura 13, en respuesta a los eventos tales, como: un periodo que expira, una solicitud de impresión de documento, o un cambio en el estado de una interface de usuario gráfico.

Inicialmente, durante la etapa 1300 el sistema presentador 200 y el motor de presentación particular 212 reciben una llamada al método DoLayout 1102 (descrito aquí anteriormente con referencia al motor de presentación 212 en la descripción del API). En respuesta a recibir el método DoLayout 1102 llamado por la aplicación 204 el sistema presentador 200 recalcula los objetos presentadores contenidos dentro de la vista 204 para acomodar los cambios a elementos en el almacén de respaldo 208 debido a la última llamada del método DoLayout 1102.

En una modalidad de la presente invención en donde los presentadores para una vista están dispuesto jerárquicamente (ver Figura 3), el sistema presentador 200 inicialmente determina el presentador raíz para la vista en la etapa 1302. En una modalidad de la invención, el presentador raíz está identificado por la propia presentadora en el elemento raíz que se somete a la vista (por ejemplo vista 204a) y por lo tanto el sistema presentador (por ejemplo el sistema presentador 200a). Después determinar el presentador raíz, el control pasa la etapa 1304 en donde el sistema presentador 200 comienza atravesar el árbol de presentadores para la vista y a recalcular cada presentador alterado.

Durante la etapa 1304 el sistema presentador 200 invoca el método OnUpdate 470 en cada presentador alterado a la vista. El método OnUpdate 470 llamado es pasado a una altura y a un ancho de un rectángulo dentro del cual el presentador debe recalcular su diseño. Después de recalcular su diseño, el presentador captura su resultado. Así, luego de completar la etapa 1304 en los presentadores anteriormente alterados han sido recalculados y sus resultados capturados para facilidad

de acceso durante la reconversión subsiguiente. El bit alterado es clarificado en cada uno de los presentadores llamados. Un conjunto de etapas llevó a cabo durante una llamada del método OnUpdate 470 ilustrativo se describen aquí en adelante en mayor detalle con referencia a la Figura 14.

Hay un número de maneras en la cual limpiar los presentadores alterados durante la etapa 1304. Sin embargo en una modalidad de la invención el sistema presentador 200, y en particular el motor de presentación 212 el método de UpdateLayout 1104, nivela las relaciones jerárquicas entre presentadores en una vista para simplificar su rol en completar la etapa 1304. En particular, durante la ejecución del método OnUpdate 470 cada presentador iterativamente llama (o alternativamente requiere al sistema presentador 200 llamar) en el método OnUpdate 470 en los hijos del presentador llamado. Así, el sistema presentador 200 inicia en su travesía del árbol presentador en el método OnUpdate 470 del presentador raíz (por ejemplo, el presentador P1 de la Figura 3) y soporta llamadas subsecuentes a presentadores hijo a medida que la llamada del método OnUpdate 470 atraviesa la ramas del árbol presentador jerárquico para

recalcular los presentadores alterados. Las llamadas iterativas al método OnUpdate 470 sobre los presentadores hijos aseguran que todos los presentadores en la vista son atravesados, y si los presentadores están alterados ellos son recalculados.

Luego de completar, cada método OnUpdate 470 llamado durante la etapa 1304 reposiciona su propiedad de alteración 450 indicando que ha sido recalculado o "limpiado". Cada método OnUpdate 470 también regresa un valor al llamador indicando si el presentador requiere reconversión (es decir, redibujo). En una modalidad de la invención el valor retornado es un valor Booleano. Si es "verdadero" entonces la reconversión es requerida. Si es "falso" lo regresado entonces la reconversión no es necesaria. Después de procesar cada uno de los presentadores alterados para un vista durante la etapa 1304, el control pasa a una etapa de conversión 1306 en donde los presentadores identificados en la lista de presentadores requiere que la reconversión sea procesada.

Durante la etapa 1306 el sistema presentador 200 invoca el método OnRender 474 sobre cada presentador identificado en la lista de presentadores que requieren

reconversión. La operación de una modalidad ilustrativa del método OnRender 474 es descrita con referencia a la Figura 14. En la implementación de la clase base del método OnRender 474, el presentador no dibuja nada, y en su lugar iterativamente llama al método OnRender 474 en sus hijos y/o presentadores encadenados para llenar sus espacios asignados (por ejemplo, rectángulos). Las versiones adaptadas del método OnRender 474 convierten el mapa de bits a través de llamadas a la aplicación 204 y/o los API del subsistema de gráficos y los excitadores de dispositivos gráficos 234. Después de procesar todos los presentadores que requieren reconversión durante la etapa 1306, el control pasa al fin 1308, y el control regresa al llamador de la llamada del método DoLayout 1102 en el motor de presentación para la vista particular.

Volviendo a la Figura 14, un diagrama de flujo resume las etapas del método OnUpdate 470 llevado a cabo para recalcular un presentador de una vista. El método OnUpdate 470 calcula un nuevo diseño para un presentador, incluyendo cualesquiera presentadores hijos, y captura los resultados para acceder durante la etapa de reconversión. En el ejemplo provisto aquí enseguida, la llamada del método OnUpdate 470 a cada presentador pasa

un conjunto de dimensiones que especifica los límites del espacio de la vista (por ejemplo rectángulo) que puede ser ocupado por el diseño del presentador. El conjunto de dimensiones especifica, por ejemplo, la altura y el ancho y si esos valores pueden ser modificados durante la ejecución de la llamada del método OnUpdate 470. Sin embargo, la invención contempla recalcular presentadores con base en cualquiera uno o más de una variedad de parámetros pasados, incluyendo dimensiones de espacio de la vista, que afectan un diseño.

Durante la etapa 1400, el presentador lleva acabo cualquier operación de dimensionamiento requerida, surgiendo desde los parámetros pasados antes de llamar cualquier presentador hijo (vía los objetos encaminadores hijo) para llevar a cabo sus actualizaciones. El contenido de tal trabajo es dependiente del diseño de un tipo presentador particular. De ahí en adelante, el presentador llama sobre el sistema presentador 200 para localizar y actualizar cualesquiera presentadores hijo del presentador actual. Durante la etapa de preprocesamiento, el presentador detecta potencialmente la necesidad de crear una nueva página/columna y crear un

nuevo encaminador hijo/presentador para manejar la nueva página /columna para la vista.

Luego, durante la etapa 1402, el método OnUpdate 470 presentador padre actual determina un presentador hijo siguiente para llamar la actualización de su diseño. Se debe notar que esta etapa, por lo menos en una primera instancia requiere del presentador padre determine la existencia de un elemento hijo y crea un encaminador hijo/presentador correspondiente. En iteraciones posteriores, el presentador padre puede usar los encaminadores hijo capturados identificados en su campo hijo 412 para identificar un encaminador hijo (no procesado)/presentador remanente siguiente, es decir, un presentador hijo que no ha sido todavía llamado durante la interacción actual en el método OnUpdate en el presentador padre actual.

El presentador padre recibe una respuesta del almacén de respaldo o determina en si mismo de sus propiedades hijo 412 si un presentador hijo permanece para ser procesado, y en la etapa 1404, si no hay hijos procesados remanentes que sean actualizados, entonces el control pasa a la etapa 1420 (descrita aquí enseguida).

De otra manera si el presentador tiene u presentador hijo no procesado dentro de su propiedad hijo 412, que no ha sido actualizado, entonces el control pasa a la etapa 1406.

En la etapa 1406, el presentador llama al método Update 520 en el encaminador hijo retornado, que está interpuesto entre el presentador padre y un presentador hijo actual de interés, para obtener una altura y un ancho de un presentador hijo con el cual el encaminador hijo está asociado. El presentador hijo está asociado con un elemento gráfico particular (especificado en su campo de propiedad de elementos 442).

En la etapa 1410 el encaminador hijo llama al método OnUpdate 470 sobre el presentador hijo con el cual el encaminador hijo está asociado. El método OnUpdate 470 calcula una altura y ancho actualizados del presentador hijo (y si la reconversión es requerida). Se debe notar que durante la etapa 1410 el método OnUpdate 470 es invocado recursivamente por los presentadores hijo hasta que un presentador hijo llamado determina, en la etapa 1402 que este no tiene hijos y regresa a su presentador padre que llama (por medio de su encaminador hijo).

Después de invocar el método OnUpdate 470 sobre el presentador hijo el control pasa a la etapa 1414.

Así, en la etapa 1414 el presentador padre que llama (de la etapa 1406) recibe una respuesta a la llamada del método Update 520 al encaminador hijo identificado. La respuesta incluye la altura y ancho del presentador hijo como parámetros retornados. La respuesta también indica si el presentador hijo que se llamó requiere reconversión. Luego, en la etapa 1416 el presentador padre posiciona el presentador hijo que se llamó dentro del espacio de diseño asignado al presentador padre con base en la colocación de algoritmo/política. En una modalidad de la invención, durante la etapa 1416 el presentador padre coloca al presentador hijo dentro de su diseño con base en los parámetros de altura y ancho del presentador hijo recibidos por el presentador durante la etapa 1414.

El control luego regresa a la etapa 1402 en donde el método OnUpdate 470 del presentador padre continua actualizando cualquier presentador hijo remanente (a través de sus objetos de envoltura encaminador hijo).

Si en la etapa 1404, ningún hijo (elemento o encaminador/presentadores) permanece para ser procesado/actualizado entonces el control pasa de la etapa 1420. En la etapa 1420 en método OnUpdate 470 del presentador llamado lleva a cabo las funciones de diseño adaptado incluyendo potencialmente llamadas adicionales al método OnUpdate 470 sobre presentadores hijo para sintonizar el diseño para el presentador.

En la etapa 1422, o potencialmente cualquier punto en donde nueva información es derivada por el método OnUpdate 470, los resultados del procesamiento de diseño son capturados para un uso posterior durante una etapa de reconversión del procesamiento de vista de gráficos.

Después de el procesamiento de la actualización ha sido completado por el presentador, incluyendo todos los hijos del presentador llamado, en la etapa 1424 el presentador regresa sus dimensiones al llamador (sistema presentador 200) lo mismo que un valor que indica si el presentador requiere ser reconvertido.

Podrá apreciarse por aquellos versados en la técnica que una nueva plataforma es ejemplar y métodos ilustrativos han sido descritos para a la administración de diseño/presentación del procesamiento de una salida de interface gráfica de documento/usuario dentro de un ambiente de computo que incluye dispositivos de salida gráfica tales como pantallas de interface de usuario gráficas o una impresora. En vista de los muchos ambientes posibles en los cuales los principios de esta invención pueden ser aplicados y la flexibilidad del diseño y de llevar a cabo las utilidades de soporte lógico y herramientas, debe reconocerse que las modalidades descritas aquí solo significan que son ilustrativas y no deberían tomarse como limitativas del alcance de la invención. Aquellos versados en la técnica los cuales la presente invención aplica podrán apreciar que las modalidades ilustradas pueden ser modificadas en disposiciones y detalles sin salirse del espíritu de la invención. Por lo tanto, la invención como se describió aquí contempla todas aquellas modalidades que pueden venir dentro del alcance de las siguientes reivindicaciones y equivalentes de las mismas.

REIVINDICACIONES

1. Un sistema de administración de diseño de salida gráfica para disponer y mantener salida de programas visualizadas gráficamente que se ejecuta sobre un sistema de computo, el sistema de administración de diseño de salida gráfica comprende:

Elementos gráficos que contienen datos que representan los contenidos desplegados de un programa;

Presentadores que definen estados de visualización para los elementos gráficos, en donde un presentador de un tipo particular mantiene una descripción de diseño para un elemento gráfico correspondiente; y

Un sistema presentador, que incluye una interface anfitriona presentadora que comprende un método para preparar un diseño para una vista, de acuerdo con un conjunto de presentadores asociados con elementos gráficos contenidos dentro de la vista.

2.El sistema de administración de diseño de salida gráfica de la reivindicación 1 en donde el sistema presentador comprende adicionalmente una clase base de presentado, y en donde el conjunto de presentadores son derivados de las clases presentadoras derivadas de la clase base presentadora.

3.El sistema de administración de diseño de salida gráfica de la reivindicación 2 en donde los presentadores incluyen un método de actualización para calcular las propiedades de diseño para el presentador.

4.El sistema de administración de diseño de salida gráfica de la reivindicación 3 en donde los presentadores incluyen un método de conversión para generar instrucciones emitidas a un subsistema de salida de gráficos del sistema de computo.

5.El sistema de administración de diseño de salida gráfica de la reivindicación 2 en donde las clases presentadoras incluyen clases presentadoras predefinidas y unas clases presentadoras externas.

6.El sistema de administración de diseño de salida gráfica de la reivindicación 1, en donde cada uno de los presentadores calcula un estado de diseño para un elemento gráfico correspondiente con base en valores de parámetro de diseño especificados en el presentador.

7.El sistema de administración de diseño de salida gráfica de la reivindicación 1 que comprende adicionalmente un conjunto de manejadores de notificación, cada manejador de notificación está asociado con un conjunto de presentadores particulares en la vista, y en donde un manejador de notificación procesa un cambio a un elemento gráfico para determinar si se actualiza un presentador correspondiente.

8.El sistema de administración de diseño de salida gráfica de la reivindicación 7 en donde cada presentador especifica un tipo de manejador de notificación que suministra notificaciones de cambio al presentador particular con respecto a un elemento gráfico asociado.

9.El sistema de administración de diseño de salida gráfica de la reivindicación 8 en donde un

manejador de notificación determina una parte de un presentador correspondiente que requiere actualización en vista de cambios a un elemento gráfico asociado.

10. El sistema de administración de diseño de salida gráfica de la reivindicación 7 en donde el sistema presentador registra, por medio de los manejadores de notificación, los presentadores que requieren actualización con base en los cambios a correspondientes elementos gráficos.

11. El sistema de administración de diseño de salida gráfica de la reivindicación 1 en donde un elemento gráfico derivado de los elementos gráficos está asociado con presentadores múltiples.

12. El sistema de administración de diseño de salida gráfica de la reivindicación 11 en donde los presentadores múltiples con los cuales el elemento gráfico derivado está asociado son de un mismo tipo.

13. El sistema de administración de diseño de salida gráfica de la reivindicación 12 en donde los

presentadores múltiples corresponden a columnas distintas dentro de una misma vista.

14. El sistema de administración de diseño de salida gráfica de la reivindicación 12 en donde los presentadores múltiples corresponden a distintas páginas en una vista de documento de un elemento gráfico único.

15. El sistema de administración de diseño de salida gráfica de la reivindicación 11 en donde un primer presentador y un segundo presentador, de los múltiples presentadores asociados con el elemento gráfico derivado, están asociados con primeros y segundas vistas distintas respectivas.

16. El sistema de administración de diseño de salida gráfica de la reivindicación 15 en donde el primero y segundas vistas distintas son hospedadas por primeros y segundos sistemas presentadores respectivos.

17. El sistema de administración de diseño de salida gráfica de la reivindicación 1 en donde la vista corresponde a una región rectangular dentro de un espacio de visualización gráfico.

18. El sistema de administración de diseño de salida gráfica de la reivindicación 11 en donde los presentadores múltiples asociados con los elementos gráficos derivados son de tipos diferentes.

19. El sistema de administración de diseño de salida gráfica de la reivindicación 18 en donde uno de los presentadores múltiples comprende un presentador de adorno a un presentador principal.

20. El sistema de administración de diseño de salida gráfica de la reivindicación 1 en donde los presentadores dentro de una vista están dispuestos jerárquicamente tal que los presentadores hijo están contenidos dentro de una región especificada por un presentador padre correspondiente.

21. Un sistema presentador para la incorporación dentro de un sistema de administración de diseño de salida gráfica para procesar un diseño, a nombre de un programa que suministra elementos gráficos que contienen datos que representan el contenido desplegable del programa, por medio de presentadores que

definen los estados de visualización para los elementos gráficos, en donde un presentador mantiene una descripción de diseño para un elemento gráfico correspondiente, y en donde el sistema presentador hospeda y dispone los presentadores asociados con los elementos gráficos dentro de una vista, el sistema presentador comprende:

Una clase base de presentadora desde la cual las clases presentadoras son derivadas, y en donde los presentadores son derivados de las clases presentadoras; y

Una interface huésped presentadora que comprende un conjunto de métodos que incluyen por lo menos un método para preparar un diseño envuelto en el conjunto de presentadores asociados con los elementos gráficos dentro de la vista.

22. El sistema presentador de la reivindicación 21 en donde la clase base presentadora especifica un método de actualización adaptable, pasando por alto por las clases presentadoras derivadas, para calcular las propiedades de diseño para un presentador.

23. El sistema presentador de la reivindicación 22 en donde la clase base presentadora especifica un método de conversión adaptable, pasando por alto por las clases presentadoras derivadas, para generar instrucciones otorgadas a un subsistema de la salida de gráficos del sistema computador.

24. El sistema presentador de la reivindicación 21 en donde el sistema presentador soporta un conjunto de clases presentadoras que incluyen clases presentadoras predefinidas y clases presentadoras externas.

25. El sistema presentador de la reivindicación 21 en donde la clase base presentadora especifica un método para calcular un estado de diseño para un elemento gráfico con base en los valores de parámetros de diseño especificados en un presentador asociado.

26. El sistema presentador de la reivindicación 21 que comprende adicionalmente una clase base manejadora de notificaciones desde la cual las

clases manejadoras de notificación se derivan, y en donde cada manejador de notificación derivado de una clase de manejador de notificación está asociado con un presentador particular en una vista, y en donde un manejador de notificación procesa un cambio a un elemento gráfico para determinar si se actualiza un presentador correspondiente.

27. El sistema presentador de la reivindicación 26 en donde la clase base presentadora incluye un mantenedor de posición para un presentador para especificar un tipo de manejador de notificación que suministre notificaciones de cambio al presentador particular con respecto a un elemento gráfico asociado.

28. El sistema presentador de la reivindicación 26 en donde los registros del sistema presentador, por medio de los manejadores de notificación, los presentadores que requieren actualización con base en cambios para elementos gráficos correspondientes.

29. El sistema presentador de la reivindicación 21 en donde el sistema presentador está

asociado con una vista caracterizado por un elemento gráfico de raíz y un área asignada dentro del cual se muestra el elemento gráfico de raíz y sus elementos gráficos hijo.

30. El sistema presentador de la reivindicación 29 en donde la vista corresponde a una región rectangular dentro de un espacio de visualización gráfico.

31. El sistema presentador de la reivindicación 21 en donde la interface huésped presentadora incluye un constructor para derivar el sistema presentador para diseñar una visualización para una vista que comprende un conjunto de elementos gráficos.

32. El sistema presentador de la reivindicación 31 en donde la vista está especificada por lo menos en parte por un elemento gráfico de raíz pasado al constructor.

33. El sistema presentador de la reivindicación 21 en donde el sistema presentador soporta

presentadores múltiples encadenados tal que un primer presentador suministra adornos para un segundo presentador anunciado con un elemento gráfico

34. El sistema presentador de la reivindicación 21 en donde la clase base presentadora incluye un mantenedor de posición para especificar presentadores hijo, soportando así la creación de una estructura de árbol jerárquico de presentadores dentro de una vista tal que los presentadores hijo están contenidos dentro de una región específica dada por un presentador padre correspondiente.

35. Un método para procesar un diseño, a nombre de un programa que suministra elementos gráficos que contiene datos que representan contenidos desplegables del programa, por medio de presentadores que definen estados de visualización para los elementos gráficos, y en donde un presentador mantiene una descripción de diseño para un elemento gráfico correspondiente, y en donde el sistema presentador hospeda y dispone a los presentadores asociados con los elementos gráficos dentro de una vista, el método comprende:

Suministrar un sistema presentador que comprende una clase base presentadora y una interface huésped presentadora que comprende un método para crear un diseño involucrado en un conjunto de presentadores asociados con elementos gráficos dentro de una vista;

Recibir una solicitud, por la interface de huésped presentadora, para generar un diseño para un conjunto de elementos gráficos dentro de una vista; y

Derivar, para el conjunto de elementos gráficos, presentadores correspondientes desde un conjunto de clases presentadoras derivadas de la clase base presentadora e invocar un método en cada presentador derivado para calcular un estado de diseño para un elemento gráfico correspondiente al presentador.

36. El método de la reivindicación 35 que comprende adicionalmente la etapa de:

Suministrar una clase base manejadora de notificación que especifica una interface para un conjunto de manejadores de notificación que facilitan la

actualización incremental a un diseño con base en cambios a correspondientes elementos gráficos.

37. El método de la reivindicación 35 que comprende adicionalmente organizar los presentadores para la vista en la forma de un árbol jerárquico.

38. El método de la reivindicación 35 en donde la derivación del sistema presentador es creada en respuesta a una llamada para crear un diseño para una vista, creando así un sistema presentador distinto, y presentadores correspondientes, para cada vista.

39. El método de la reivindicación 35 en donde la clase base presentadora especifica un método de actualización para calcular las propiedades de diseño para un presentador derivado asociado con un elemento gráfico particular.

40. El método de la reivindicación 35 en donde la clase base presentadora especifica un método de conversión para generar instrucciones emitidas a un subsistema de salida de gráficos.

41. El método de la reivindicación 35 en donde el sistema presentador soporta la derivación de clases presentadoras predefinidas y clases presentadoras externas.

42. El método de la reivindicación 35 en donde las clases presentadoras corresponde a diferentes tipos de presentadores que diseñan diferentes tipos de contenido de elementos gráficos.

43. El método de la reivindicación 35 en donde los presentadores dentro de una vista están dispuestos jerárquicamente tal que los presentadores hijos están contenidos dentro de una región especificada para un presentador padre correspondiente.

44. Un medio legible por computadora que incluye instrucciones ejecutables por computadora para facilitar el procesamiento de un diseño, a nombre de un programa que suministra elementos gráficos que contiene datos que representan el contenido desplegable del programa, por medio de presentadores que definen estados de visualización para los elementos gráficos, y en donde un presentador mantiene una descripción de diseño para un

elemento gráfico correspondiente, y en donde el sistema presentador hospeda y dispone a los presentadores asociados con los elementos gráficos dentro de una vista, las instrucciones ejecutables por computador facilitan el comportamiento o llevar a cabo un método que comprende:

Suministrar un sistema presentador que comprende una clase base presentadora y una interface huésped presentadora que comprende un método para crear un diseño involucrado en un conjunto de presentadores asociados con elementos gráficos dentro de una vista;

Recibir una solicitud, por una interface huésped presentadora, para generar un diseño para un conjunto de elementos gráficos dentro de una vista; y

Derivar, del conjunto de elementos gráficos, presentadores correspondientes de un conjunto de clases presentadoras derivadas de la clase base presentadora e invocar un método sobre cada presentador derivado para calcular un estado de diseño para un elemento gráfico correspondiente al presentador.

45. El medio legible por computadora de la reivindicación 44 que comprende adicionalmente instrucciones ejecutables por computadora para:

Organizar los presentadores para la vista en la forma de un árbol jerárquico.

46. El medio legible por computadora de la reivindicación 44 en donde las instrucciones ejecutables por computadora facilitan la creación de una derivación del sistema presentador en respuesta a una llamada para crear un diseño para una vista, creando así un sistema presentador distinto, y presentadores correspondientes, para cada vista.

47. El medio legible por computadora de la reivindicación 44 en donde la clase base presentadora especifica un método de actualización para calcular las propiedades de diseño para un presentador derivado asociado con un elemento gráfico particular.

48. El medio legible por computadora de la reivindicación 44 en donde la clase base presentadora especifica un método de conversión para generar

instrucciones emitidas a un subsistema de salida de gráficos.

49. El medio legible por computadora de la reivindicación 44 en donde el sistema presentador soporta la derivación de clases presentadoras predefinidas y clases presentadoras externas.

50. El medio legible por computadora de la reivindicación 44 en donde las clases presentadoras corresponde a diferentes tipos de presentadores que diseñan diferentes tipos de contenido de elementos gráficos.

51. El medio legible por computadora de la reivindicación 44 en donde las clases base presentadoras especifican un mantenedor de posición para presentadores hijo, que facilitan de esta manera trazar jerárquicamente dentro de una vista tal que los presentadores hijo están contenidos dentro de una región específica dada por un presentador padre correspondiente.

RESUMEN

Una estructura de sistema presentador y métodos que para llevar a cabo tal estructura se describen para hospedar un conjunto de presentadores que faciliten el mantenimiento de diseños para vistas de aplicaciones asignados a un conjunto de elemento gráficos. El sistema presentador suministra una clase presentadora base y un conjunto de métodos de interface, llevados a cabo por un motor de presentación, para crear e integrar un conjunto extensible de clases presentadoras para manejar varios tipos de datos de elementos gráficos durante una operación de diseño dentro de una vista especificada. El sistema presentador permite que los desarrolladores de interface de usuario de aplicación implementen operaciones de diseño de visualización complejas por medio de llamadas al motor de presentación. Ejemplo de tales operaciones de diseño de visualización complejas se incluyen: paginación, cálculo parcial, cálculo incremental, árboles múltiples, encadenamiento de características/operaciones de diseño.

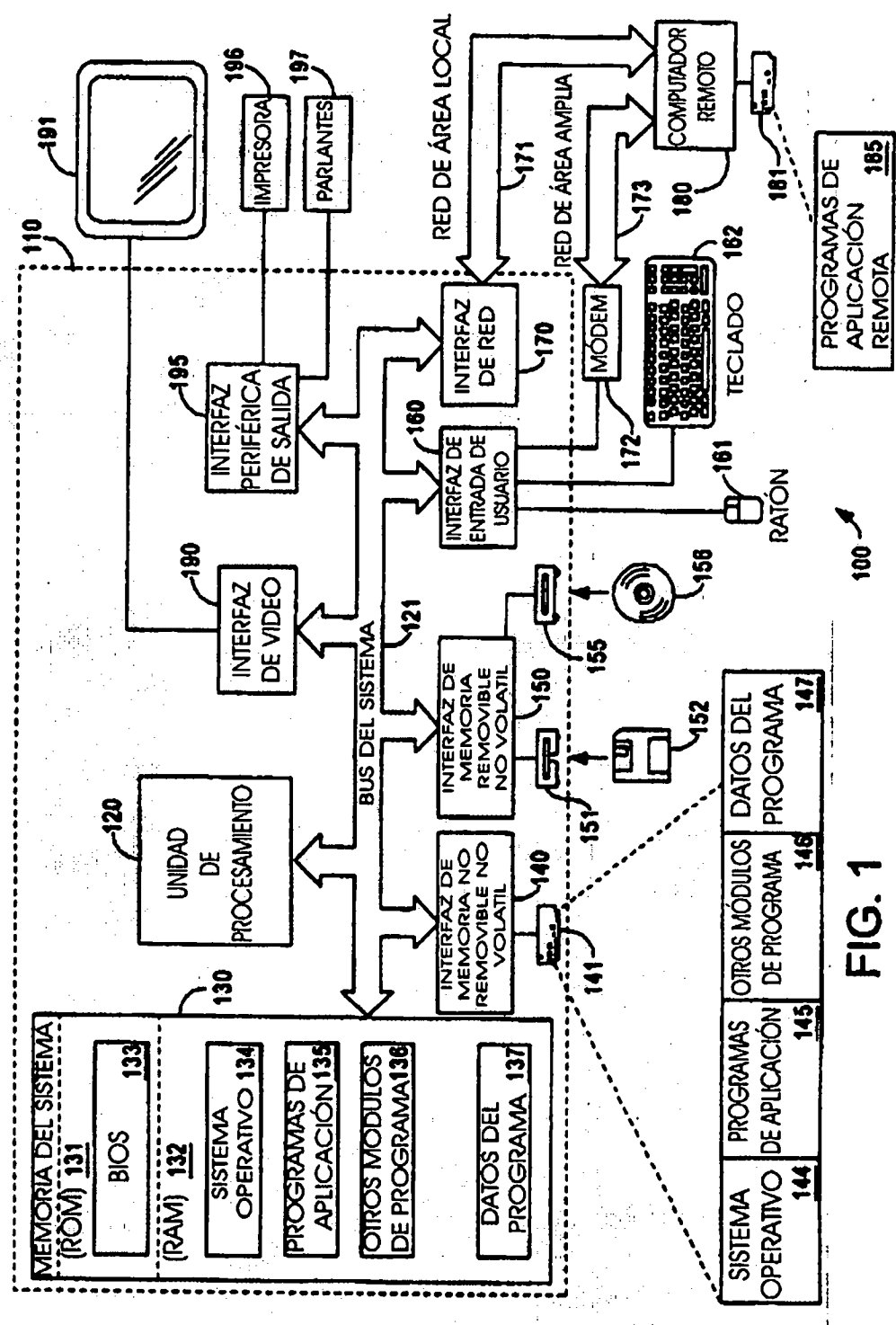


FIG. 1

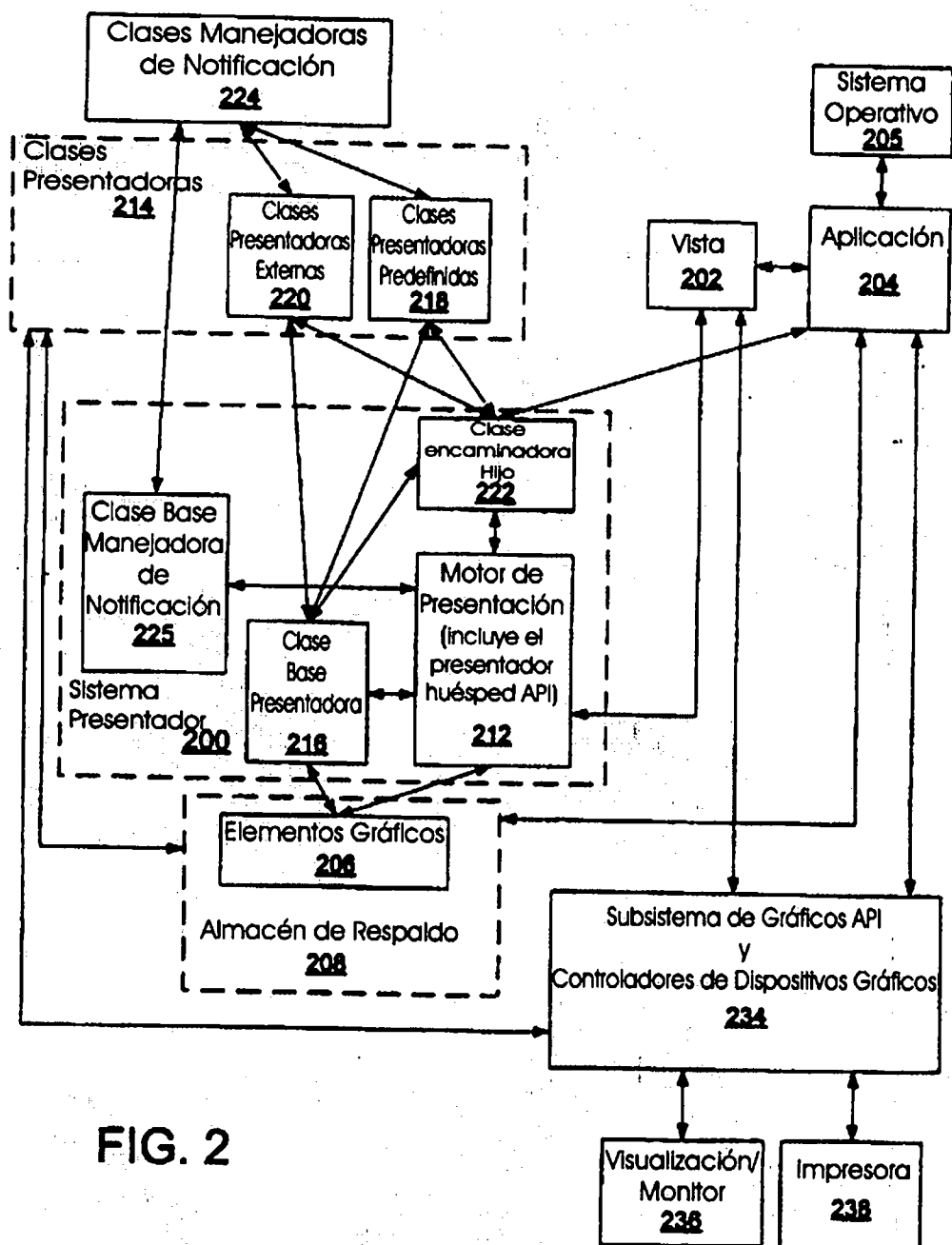


FIG. 2

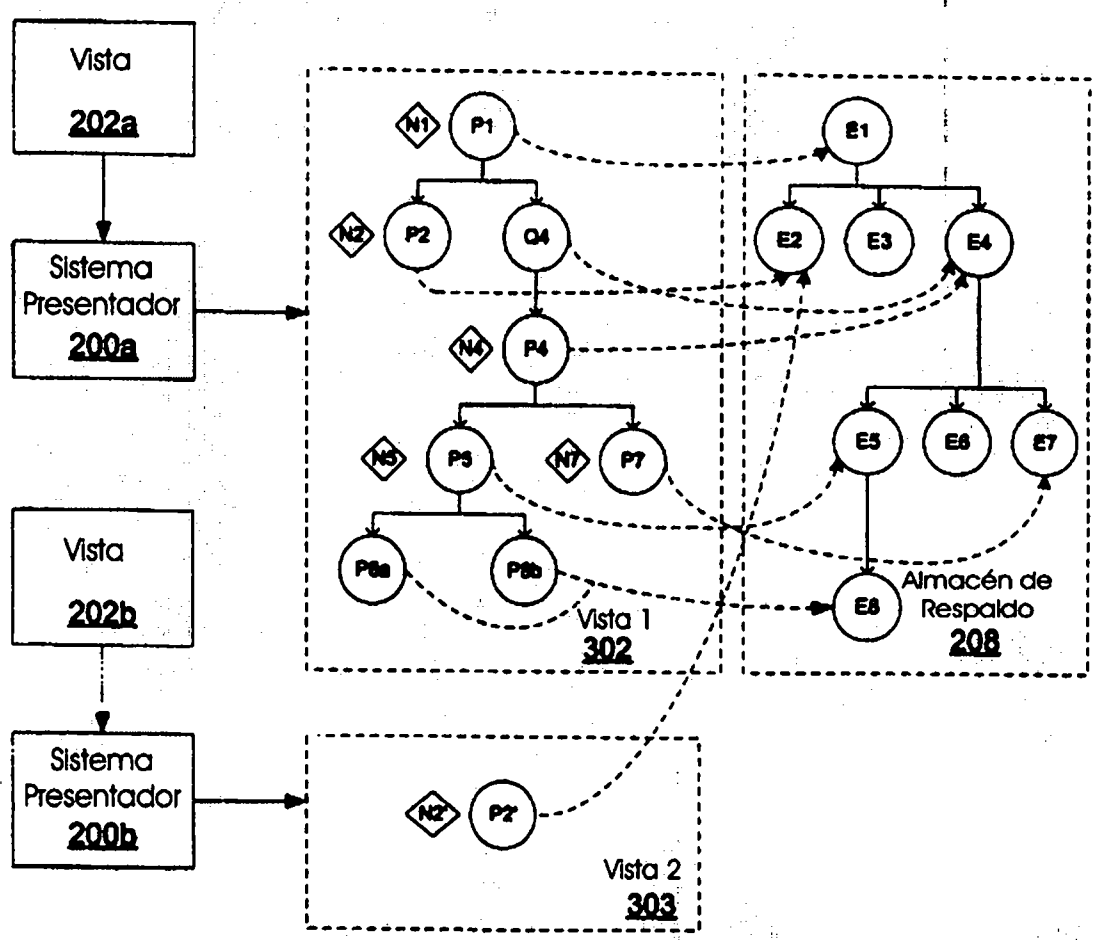


FIG. 3

**Propiedades/Campos de la Clase Base
Presentadora**

400	Fondo
402	Opacidad de Fondo
404	Borde
406	Color de Borde
408	Estilo de Borde
410	Caja de Límites
412	Hijos
414	Clip
416	Almohadillas
418	Contenido de Altura
420	Contenido de Ancho
422	Altura
424	Ancho
426	Altura por Defecto
428	Ancho por Defecto
430	Ancho Máximo
432	Ancho Mínimo
434	Margen
436	Visibilidad
438	Presentador
440	IsMainPresenter
442	Propietario de Elemento
444	Contexto de Diseño
446	Manejador de Notificación
448	Tipo de Manejador de Notificación
450	Alteración
460	Tamaño de Diseño
462	Afecta el Diseño
464	Afecta el Diseño Padre
466	Afecta la Conversión

FIG. 4a

Métodos de la Clase Base Presentadora

470	OnUpdate
471	Átomo de Registro
472	OnUpdateBoundingBox
474	OnRender
476	OnHitTestDetail
478	OnCreateViewResult
480	HitTest
482	OnGetBypassList
484	CalculateBounds
486	GetChildProxyForElement
488	OnBeforeBypass
490	OnDisconnectChildren
492	OnMinWidth
494	Presentador
496	QueueLayoutTask
498	OnQueryValue

FIG. 4b

Clase Encaminadora Hijo

500	Propietario de Elemento
502	Transformación
504	Caja de Límites
506	Alteración
510	QueryDefaultSizeInfo
520	Update
530	Sujeción
540	Valor de Consulta
550	Tamaño de Diseño de Consulta
560	Cómputo de MinMax

FIG. 5

Interfaz de Manejador de Notificación

600	OnNotify
602	OnLayoutTask

FIG. 6

Clase de Sitio de Manejador de Notificación

700	Propietario de Elemento
702	Átomo de registro
704	Crear Notificación
706	Notificar a los Descendientes
708	Notificar a los Ancestros
710	NotifyAll
712	NotifySelf

FIG. 7

384

Clase de Vista

800	Vista
802	Elemento Raíz
804	DoLayout
806	Tamaño de Vista
808	GetViewResult
810	HitTest
812	Crear Página

FIG. 8

BoxSizeInfo

900	Tamaño Propuesto
902	Tamaño Padre
904	Ancho Fijo
906	Altura Fija

FIG. 9

Descriptor de Página

1000	PageSize
1002	Registro de Rompimiento
1004	AvailableSize

FIG. 10

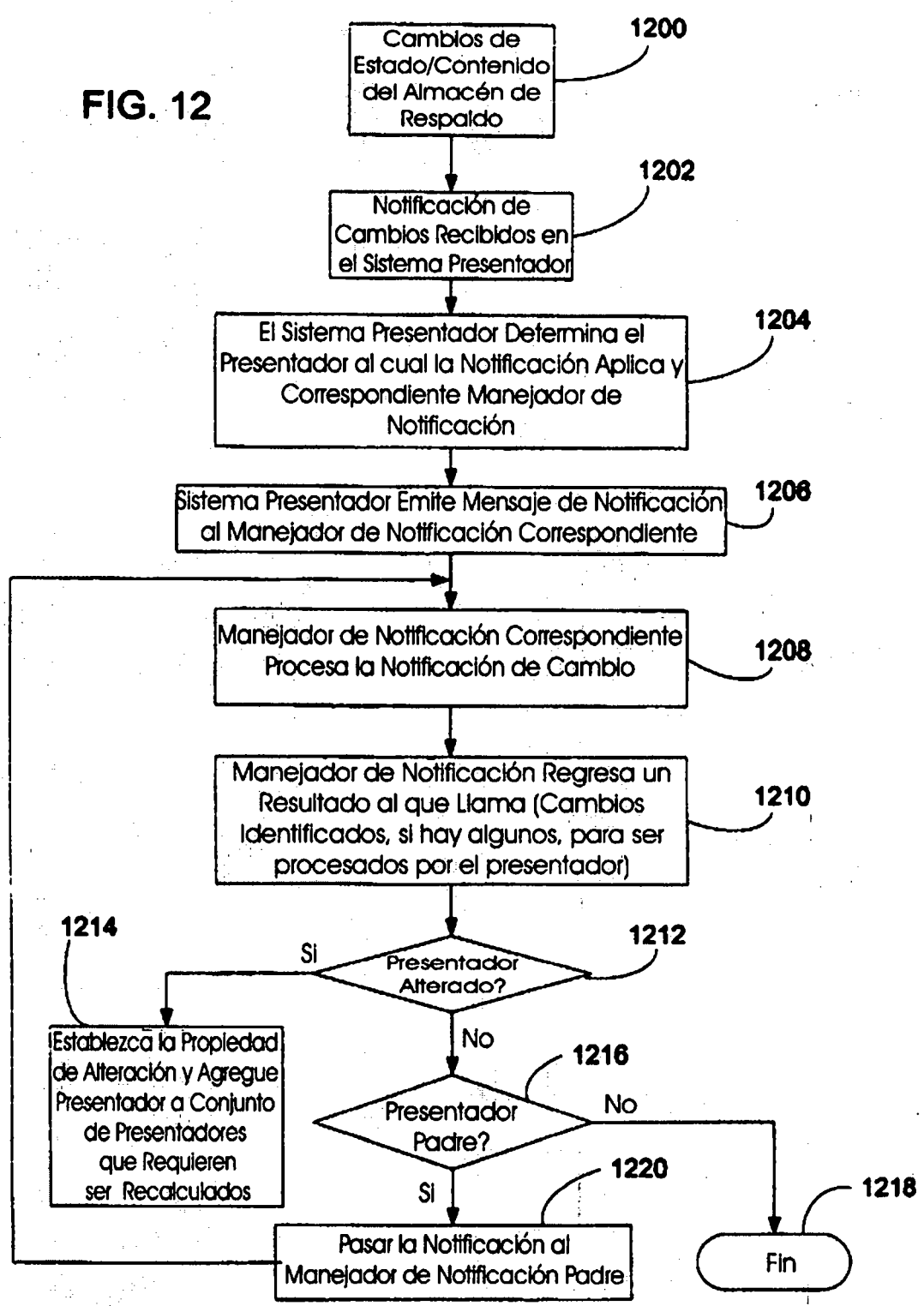
Clase de Motor de Presentación

1100	Constructor
1102	DoLayout
1104	UpdateLayout
1106	CreatePage

FIG. 11

228

FIG. 12



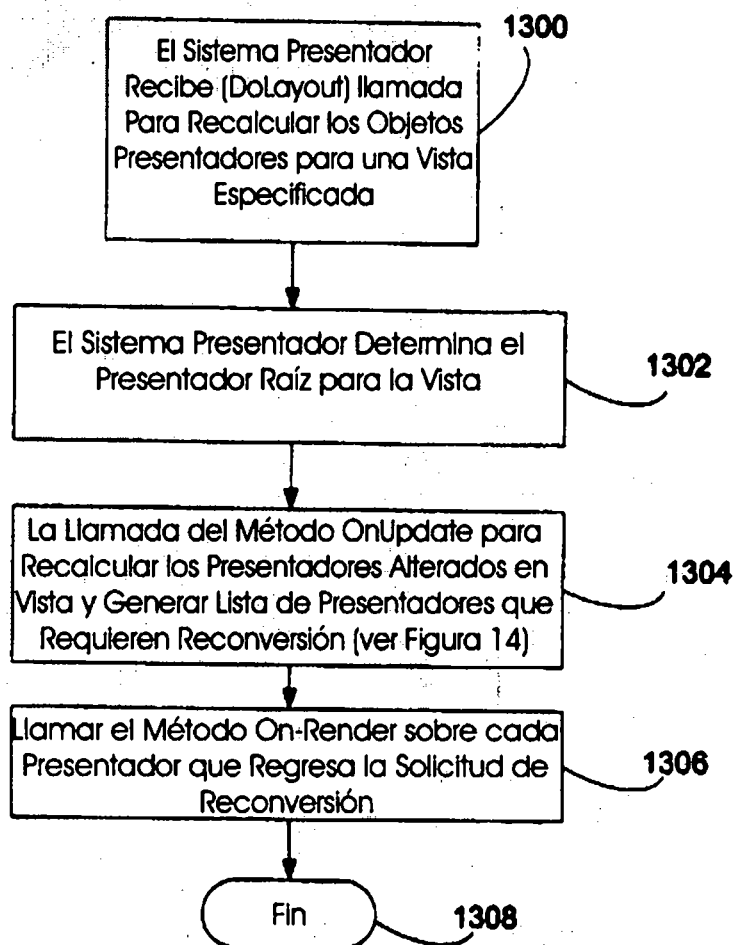


FIG. 13

388

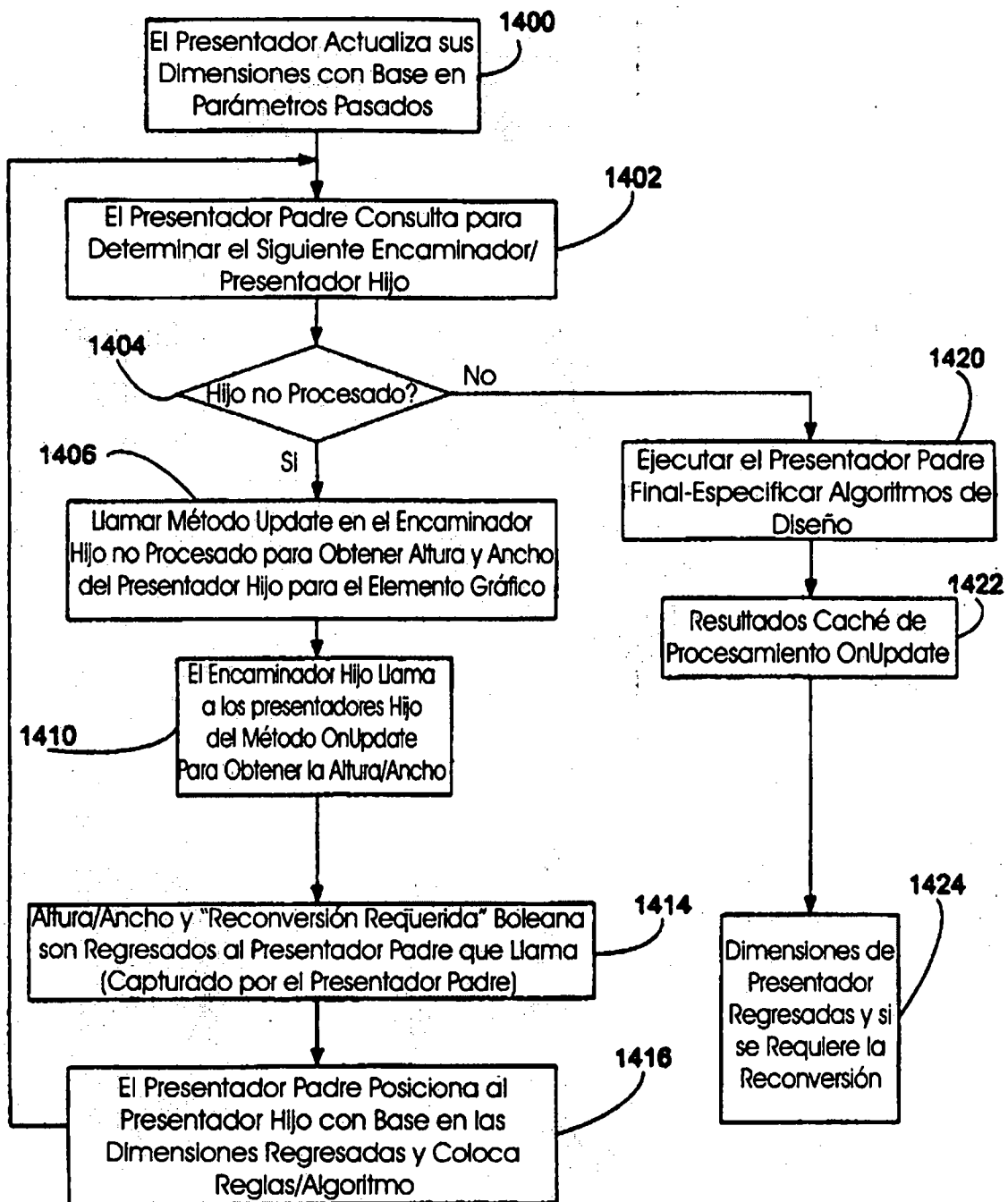


FIG. 14

231

2020
Bogotá D C

Doctor(a)
LUZ HELENA ADARVE
Apoderado(a)

REFERENCIA:	Radicación N°	04-39312
	Solicitud internacional	PCT/US03/15379
	Fecha limite de inicio en fase nacional	15/12/2005
	Trámite	011
	Evento	379
	Actuación	416
	Oficio N°	1406
	Folios	001

Teniendo en cuenta que la solicitud de privilegio de patente que se tramita bajo el expediente indicado en el asunto, reúne los requisitos de forma establecidos en los artículos 26 y 27 de la Decisión 486, de la Comisión de la Comunidad Andina, el Tratado de Cooperación en Materia de Patentes (PCT), el Reglamento y la Circular Única de la Superintendencia de Industria y comercio, se envía el extracto de la solicitud a la oficina de comunicaciones para efecto de su publicación en la Gaceta de Propiedad Industrial, conforme lo dispone el artículo 40 de la referida Decisión 486.

Cúmplase
Dado en Bogotá D C, 02 DIC. 2005


ALIX CARMENZA CESPEDES DE VERGEL
Jefe División de Nuevas Creaciones.

202028

DIVISION DE NUEVAS CREACIONES

RADICACIÓN EXPEDIENTE No. 04-039312

EL EXTRACTO DE LA PRESENTE SOLICITUD DE **PATENTE DE INVENCION**

PCT FUE PUBLICADO EN LA GACETA DE PROPIEDAD INDUSTRIAL No **563**

DE FECHA **28 DE ABRIL DEL 2006**, EL TERMINO HABIL SEÑALADO POR LA

LEY EXPIRA EL **31 DE JULIO DEL 2006**

NOTA: Dentro del plazo de los seis (6) meses siguientes a partir de la fecha de publicación en la gaceta de Propiedad Industrial, deberá solicitar y cancelar el examen de patentabilidad so pena de que la solicitud caiga en abandono, de acuerdo con lo establecido en el Artículo 44 de la Decisión 486/2000.

SE PRESENTARON OPOSICIONES POR PARTE DE TERCEROS.

SI ☐ ☐

NO ☒ ☐ ☐

Bogotá D.C., Marzo de 2007