Industria y Comercio

SUPERINTENDENCIA

Delegatura de propiedad Industrial

División de Nuevas Creaciones

SOLICITUD

PATENTE DE INVENCIÓN

21. **EXPEDIENTE No.** 05-51831

54. **TÍTULO** Sistema y método para las extensiones y herencias de unidades de información que se pueden Administrar con un sistema de interfase Hardware / Software.

51. **CLASIFICACIÓN INTERNACIONAL** G06F 17/30

71. **SOLICITANTE** Microsoft corporation

DOMICILIO Redmond, Washington, Estados Unidos de America

74. **APODERADO** Dario Cardenas Navas

22. **BOGOTÁ, D. C.,**

(FORMA P-10)

1

PETITORIO

Industria y Comercio
SUPERINTENDENCIA

FORMULARIO UNICO DE SOLICITUD DE PATENTE
PCT
ENTRADA EN FASE NACIONAL 24-05-06

① **SOLICITUD DE:**

[X] Patente de Invención [ ] Patente de Modelo de Utilidad

[ ] Capítulo I [X] Capítulo II

② **SOLICITANTE (71)**	Nombre:	**MICROSOFT CORPORATION**	**IDENTIFICACION**
	Dirección:	One Microsoft Way Redmond, Washington 98052, Estados Unidos de America	C.C. [ ] NIT [ ]
	Nacionalidad o Domicilio:	REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA.	C.E. [ ] Otro [X]
	Lugar de Constitución:	Estado de Washington, Estados Unidos de América	Cual _____
	Teléfono: 3123600 .	Fax: 3122410	Número _____
	E-mail:	general@cardenasycardenas.com	
③ **REPRESENTANTE O APODERADO**	Nombre:	**DARIO CARDENAS NAVAS**	**IDENTIFICACION**
	Dirección:	Carrera 7 No 71-52 Torre B Piso 9	C.C. [X] NIT [ ]
	Teléfono: 3123600	Fax: 3122410	C.E. [ ] Otro [ ]
	E-Mail:	general@cardenasycardenas.com	Cual No 17.066.629 BTA TP 1.250 C.S.J.

④ **INVENTOR(ES) (72)**

2. Nombre:	1. Bekim Demiroski	
	2. Robert T. Whitney	
	3. Patrick J. Thompson	
	4. Anil K. Nori	
	5. Sameet Agarwal	
	6. Pedro Celis	
	7. David G. Campbell	
	8. F. Soner Terek	
	9. Kim Cameron	
	10. Walter R. Smith	
	11. Darren A. Shakib	
	12. Nathaniel H. Ballou	
	13. Srinivasmurthy P. Acharya	
	14. Balan Sethu Raman	
	15. Peter M. Spiro	
2. Dirección:	1. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	2. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	3. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	4. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	5. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	6. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	7. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	8. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	9. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	10. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	11. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	12. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	13. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	14. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
	15. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América	
3.Nacionalidad o Domicilio:	1. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	2. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	3. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	4. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	5. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	6. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	7. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	8. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	9. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	10. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	11. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	12. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	13. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	14. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	
	15. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA	

1

⑤ **PCT (86)**

Solicitud Internacional No. PCT/US2004/024569 Fecha Julio 29, 2004

Publicación Internacional No. WO2005/024666 Fecha 17-03-2005

⑥ **Título(54):** SISTEMA Y MÉTODOS PARA LAS EXTENSIONES Y HERENCIAS DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR CON UN SISTEMA DE INTERFACE HARDWARE / SOFTWARE

⑦ **Clasificación Internacional (51):** G06F 17/30

⑧ **Prioridad** SI [X] NO [ ]

(33)País de Origen	(32)Fecha	(31)N° de Solicitud
U.S.A.	Octubre 24, 2003	10/693.574

Comprobante de Pago No.: 05-37.300 **Fecha** Mayo 19, 2005

ANEXOS

- [X] Comprobante de pago de la tasa de presentación de la solicitud.
- [X] Documento que acredita la existencia y representación legal cuando el solicitante sea persona jurídica. (Protocolo 16.069)
- [X] Poderes, si fuere el caso. (Protocolo 16.069)
- [ ] Documento de cesión del inventor al solicitante o a su causante.
- [ ] Declaraciones
- [X] Copia de la solicitud internacional. (Resumen, descripción, reivindicaciones, dibujos)
- [X] Traducción de la solicitud internacional al castellano. (Resumen, descripción, reivindicaciones, dibujos)
- [ ] Copia del examen preliminar internacional efectuado por la Autoridad Internacional de Examen Preliminar (IPEA).
- [ ] Traducción del examen preliminar internacional efectuado por la IPEA.
- [ ] De ser el caso, copia del contrato de acceso.
- [ ] De ser el caso, documento que acredite la licencia o autorización de uso de conocimientos tradicionales de las comunidades indígenas.
- [ ] De ser el caso, certificado de depósito del material biológico.
- [X] Arte final 12x12.
- [ ] De ser el caso, información sobre otras solicitudes de patente o títulos obtenidos en el extranjero por el mismo titular o su causante, relacionadas parcial o totalmente con la invención de esta solicitud.
- [X] De ser el caso, modificaciones efectuadas a la solicitud internacional.
- [ ] Otros.

11 FIGURA CARACTERISTICA

12 Solicito la Concesión de la patente,

NOMBRE: **DARIO CARDENAS NAVAS**

FIRMA:

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

3

SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 05 - 37,300
FECHA : MAYO 19 DE 2005

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

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGO	Vr. PAGO
CARDENAS Y CARDENAS P.I. CONSIGNACION		BANCO POPULAR	050-00110-6	38039997	19/05/2005	3,028,000.00

C O N C E P T O

CANT.	RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1	50005-01-01 SOLICITUDES	1 TRAMITES DE SOL. DE PATENTE DE IN	443,000.00
		TOTAL : $	443,000.00

SON: CUATROCIENTOS CUARENTA Y TRES MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. Sistema y método para las extensiones y herencias de unidades de información que se pueden administrar con un sistema de interface hardware/software.

CARDENAS & CARDENAS
ABOGADOS

B

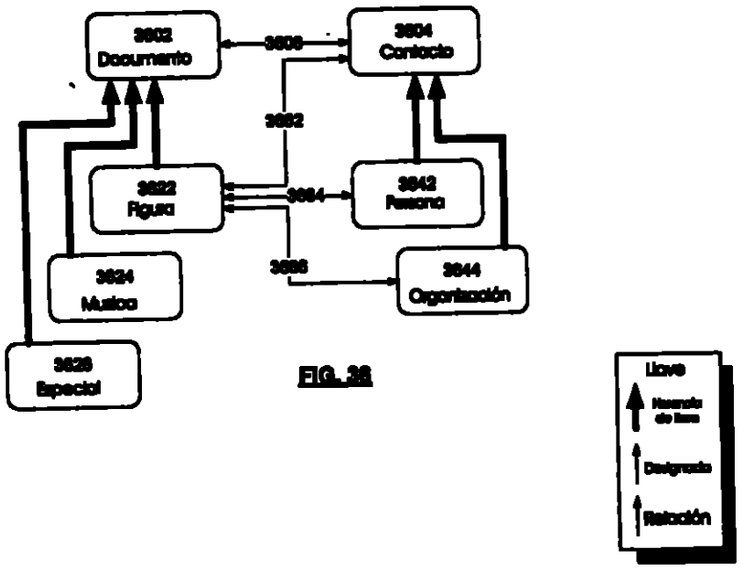

3602
Documento
3606
3604
Contacto
3622
3622
Figura
3624
3642
Persona
3624
Musica
3626
3644
Organización
3626
Especial
Llave
Designado
Relación

FIG. 36

5

SOLICITUD

PCT

Los abajo firmantes solicitan que la presente solicitud internacional sea procesada de acuerdo con el Tratado de Cooperación de Patentes.

PARA USO EXCLUSIVO DE LA OFICINA RECEPTORA
Solicitud Internacional No.
Fecha de Presentación Internacional
Nombre de la Oficina Receptora y "Solicitud Internacional PCT"
Archivo de Referencia del Agente o Solicitante (si se desea)(12 caracteres máximo) **MSFT – 4419**

Casilla No.1 Título del Invento	**SISTEMA Y MÉTODOS PARA LAS EXTENSIONES Y HERENCIAS DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR CON UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE**

Casilla No.2 Solicitante ☐ Esta persona también es inventor

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).* MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	Teléfono no. Fcsimil no. Teleprinter no. No. de registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU			
Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☒ Todos los estados designados excepto los E.U. de América	☐ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Casilla No.3 SOLICITANTE(S) Y/O INVENTOR(ES) ADICIONAL(ES)

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).* DEMIROSKI, Bekim c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	*Esta persona es:* ☐ *Solo solicitante* ☒ *Solicitante e inventor* ☐ *Solo inventor (si esta casilla de verificación es marcada, no llene adelante)* No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: MK	Estado (que es, el país) de residencia: EU			
Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

☒ **Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.**

Casilla No.4 AGENTE O REPRESENTANTE COMÚN; O DIRECCIÓN DE CORRESPONDENCIA

Por medio de la presente la persona identificada adelante ha sido señalada para actuar en nombre del (los) solicitante (s) ante las Autoridades Internacionales competentes como: ☒Agente ☐Representante común

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia). ROCCI, Steven J.; KNIGHT, Richard W. WOODCOCK WASHBURN LLP One Liberty Place, 46th Floor Philadelphia, Pennsylvania 19103 Estados Unidos de América	Teléfono: 215-568-3100 Facsimil: 215-568-3439 Teleprinter Registro del Solicitante con la Oficina No. 30,489;42.751

☒ **Dirección para correspondencia:** marque esta casilla de verificación cuando ningún agente o representante común es/haya sido señalado y el espacio anterior se utiliza en cambio para indicar una dirección espacial a la cual la correspondencia se deba enviar.

6

Continuación Casilla No. III SOLICITANTE(S) Y/O INVENTOR(ES) ADICIONAL(ES)

Si ninguna de las siguientes subcasillas se utiliza, esta hoja no se incluye en la solicitud

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriore Estado de residencia).	Esta persona es:
WHITNEY, Robert T. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriore Estado de residencia).	Esta persona es:
THOMPSON, J. Patrick c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: GB	Estado (que es, el país) de residencia: EU:

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriore Estado de residencia).	Esta persona es:
NORI, Anil K. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriore Estado de residencia).	*Esta persona es:*
AGARWAL, Sameet c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: IN	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Continuación Casilla No. III SOLICITANTE(S) Y/O INVENTOR(ES) ADICIONAL(ES)

Si ninguna de las siguientes subcasillas se utiliza, esta hoja no se incluye en la solicitud

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*	Esta persona es:
CELIS, Pedro c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐ Solo solicitante ☒ Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*	Esta persona es:
CAMPBELL, David G. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒ Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*	Esta persona es:
TEREK, F. Soner c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052Estados Unidos de America	☐ Solo solicitante ☒ Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: TR	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*	Esta persona es:
CAMERON, Kim c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de America	☐ Solo solicitante ☒ Solicitante e inventor ☐ Solo inventor (si esta casilla de vverificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: CA	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Continuación Casilla No. III SOLICITANTE(S) Y/O INVENTOR(ES) ADICIONAL(ES)

Si ninguna de las siguientes subcasillas se utilizo, esta hoja no se incluye en la solicitud

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).	Esta persona es: ☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante) No. de Registro del solicitante con la oficina
SMITH, Walter R. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).	Esta persona es: ☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante) No. de Registro del solicitante con la oficina
SHAKIB, Darren A. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU:

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).	Esta persona es: ☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante) No. de Registro del solicitante con la oficina
BALLOU, Nathaniel H. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).	*Esta persona es:* ☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante) No. de Registro del solicitante con la oficina
ACHARYA, Srinivasmurthy P. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Continuación Casilla No. III SOLICITANTE(S) Y/O INVENTOR(ES) ADICIONAL(ES)

Si ninguna de las siguientes subcasillas se utiliza, esta hoja no se incluye en la solicitud

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriores Estado de residencia).*	Esta persona es:
RAMAN, Balan Sethu c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriores Estado de residencia).*	Esta persona es:
SPIRO, Peter M. c/o MICROSOFT CORPORATION One Microsoft Way Redmond, Washington 98052 Estados Unidos de América	☐Solo solicitante ☒Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad: EU	Estado (que es, el país) de residencia: EU:

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☒ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriores Estado de residencia).*	Esta persona es:
	☐Solo solicitante ☐Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad:	Estado (que es, el país) de residencia:

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☐ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica ulteriores Estado de residencia).*	*Esta persona es:*
	☐Solo solicitante ☐Solicitante e inventor ☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)
	No. de Registro del solicitante con la oficina

Estado (que es, el país) de nacionalidad:	Estado (que es, el país) de residencia:

Esta persona es solicitante para los propósitos de	☐ Todos los estados designados	☐ Todos los estados designados excepto los E.U. de América	☐ Solamente los Estados Unidos de América	☐ Los estados indicados en la casilla complementaria

☐ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Casilla No. V DESIGNACIÓNES

El archivo de la presente solicitud constituye ante la Norma 4.9(a), de la designación de todos los Estados Contratantes que se rigen por el PCT en la fecha internacional de archivo, para que se otorge toda la protección disponible y, donde sea pertinente, para otorgar tanto la patente regional como la patente nacional.

Sin embargo,

☐ DE Alemania no esta designada para ningún tipo de protección

☐ KR *La republica de Corea* no esta designada para ningún tipo de protección

☐ RU La Federación Rusa no esta designada para ningún tipo de protección

(Se pueden utilizar las casillas de verificación anteriores para excluir (en forma irrevocable) las designaciones concernientes para evitar la cesación del efecto, bajo la ley nacional, de una solicitud anterior de donde se reivindica la prioridad. Ver las notas a la casilla no. V sobre las consecuencia de los requerimientos de la ley nacional en estos y otros estados.

Casilla No. VI REIVINDICACIONES DE PRIORIDAD

Por medio de la presente se reivindica la prioridad de la(s) siguiente(s) solicitud(es) temprana(s):

Fecha de presentación de la solicitud temprana (día/mes/año)	Número de la solicitud temprana	*Donde se encuentra la solicitud temprana:* Solicitud nacional: País o miembro del WTO	Solicitud regional:* Oficina regional	Solicitud internacional: Oficina receptora
item(1) 21 AGOSTO 2003 (21.08.03)	10/646.580	EU		
item(2) 21 AGOSTO 2003(21.08.03)	PCT/US03/26144			EU
item(3) 24Octubre 2003(24.10.03)	10/693.574	EU		

☐ Las reivindicaciones de prioridad se indican en la Casilla Complementaria.

Se solicita a la Oficina receptora preparar y transmitir a la Oficina internacional una copia certificada de la(s) solicitud(es) temprana(s) *(solo si la solicitud temprana fue presentada con la Oficina que para propósitos de esta solicitud internacional es la Oficina receptora)* identificada arriba como:

☒ Todos los items ☐ item (1) ☐ item (2) ☐ item 3 ☐ item (4) ☐ item (5) ☐ otros, ver casilla complementaria

* *En donde la solicitud temprana es una solicitud ARIPO, indique al menos un estado parte de la Convención de París para la Protección de Propiedad IndEUtrial o un Miembro de la Organización Mundial de Comercio para el cual tal solicitud temprana se presentó (Regla 4.10(b) (ii))*: .....

Casilla No. VII AUTORIDAD DE BÚSQUEDA INTERNACIONAL

Elección de la Autoridad de Búsqueda Internacional (ISA) (si dos o más Autoridades de Búsqueda Internacionales son competentes para llevar a cabo la búsqueda internacional, indique la autoridad escogida; se puede utilizar el código de dos letras):
ISA/EU.....

Solicitud para utilizar los resultados de la búsqueda temprana; Investigación de tal búsqueda *(si una búsqueda temprana se ha llevado a cabo por o solicitado de la Autoridad de Búsqueda Internacional):*

Fecha (día/mes/año)	Número	País (u Oficina Regional)
21 Agosto 2003 & 24 Octubre 2003	10/646.580 & 10/693.57	EU

Casilla No. VIII DECLARACIONES

Las siguientes declaraciones están contenidas en las Casillas Nos. VIII (i) a (v) (marque las casillas de verificación aplicables adelante e indique en la columna de la derecha el número de cada tipo de declaración):

☐ Casilla No. VIII(i) Declaración de la identidad del inventor:

☐ Casilla No. VIII(ii) Declaración del derecho del solicitante, como la fecha de presentación internacional, para solicitar y otorgarse una patente:

☐ Casilla No. VIII(iii) Declaración del derecho del solicitante, como la fecha de presentación internacional, para reivindicar la prioridad de una solicitud temprana

☐ Casilla No. VIII(iv) Declaración de inventor (solo para los propósitos de designación de los Estados Unidos de América)

Casilla No. IX LISTA DE VERIFICACIÓN; IDIOMA DE PRESENTACIÓN

Esta Solicitud Internacional contiene:

en forma de papel, el siguiente número de hojas:

Solicitud	7
Descripción (excluye listados de secuencias y/o tablas relacionadas con estas)	110
Reivindicaciones:	7
Resumen	1
Dibujos	35
Sub total de Hojas:	160

Parte de la lista de secuencias de las descripciones (el numero actual de hojas, si se archivaron en papel, así se hayan llenado en computador, ver (c) más abajo.
Total numero de hojas 160

(b) ☐ solo en forma legible por computador (Sección 801 (a)(i))

(i) ☐ listado de secuencias

(ii) ☐ tablas relacionadas con estas

(c) ☐ también en forma legible por computador (Sección 801 (a)(ii))

(i) ☐ listado de secuencias

(ii) ☐ tablas relacionadas con estas

Número y tipo de portadores (diskette, CD-ROM, CD-R u otro) en el cual está contenido el

☐ listado de secuencias.....

☐ las tablas relacionadas con esta:.....

(copias adicionales para ser indicadas de acuerdo con los ítems 9(ii) y/o 10(ii), en la columna derecha)

Esta solicitud internacional se acompaña por el(los) siguiente(s) ítem(s) (marque las casillas de verificación aplicables adelante e indique en la columna derecha el número de cada ítem):

Número de ítems:

1. ☒ Hoja de cálculo de tasas
2. ☐ Original del Poder de abogado por separado
3. ☐ Original del Poder general de abogado
4. ☒ copia del Poder general de Abogado; número de referencia si existe: .....
5. ☐ Declaración que explica falta de firma:
6. ☐ Documento(s) de prioridad identificado(s) en casilla No. VI como ítem(s):
7. ☐ Traducción de la solicitud internacional en (idioma):
8. ☐ Indicaciones separadas en relación con microorganismos u otro material biológico
9. ☐ Lista de secuencias en forma legible por computador

   (i) ☐ copia presentada para los propósitos de búsqueda internacional de acuerdo con la Regla 13ter solamente (y no como parte de la solicitud internacional):

   (ii) ☐ (solo cuando la casilla de verificación (b)(i) o (c)(i) se marca en la columna de la izquierda) copias adicionales incluyen, cuando aplica, la copia para los propósitos de búsqueda internacional de acuerdo con la Regla 13ter

   (iii) ☐ junto con las declaraciones importantes para la identidad de la copia, o copias con los listados de secuencias mencionados en la columna izquierda

10. ☐Tablas en forma legible por computador relacionadas con la lista de secuencias (indique número y tipo de portadores)

    (i) ☐ copia presentada para los propósitos de búsqueda internacional de acuerdo con la Sección 802(802b-cuarto) solamente (y no como parte de la solicitud internacional):

    (ii) ☐ (solo cuando la casilla de verificación (b)(ii) o (c)(ii) se marca en la columna de la izquierda) copias adicionales incluyen, cuando aplica, la copia para los propósitos de búsqueda internacional de acuerdo con la Sección 802(b-cuarto)

    (iii) ☐ junto con las declaraciones importantes para la identidad de la copia, o copias con las tablas mencionadas en la columna izquierda

11. ☒ Otra (especifique): Carta de tramitación, cheque y tarjeta postal

Figura de los dibujos que puede acompañar al Resumen: 36

Idioma de Presentación de la Solicitud Internacional: Inglés

Casilla No. X FIRMA DEL SOLICITANTE, O AGENTE

Junto a cada firma, indique el nombre de la persona firmante y la capacidad en la cual la persona firma (si tal capacidad no es obvia de la lectura de la solicitud)

WOODCOCK WASHBURN LLP

Steven B. Samuels. Firma

☐Casilla No. VIII(v) Declaración de descripción no perjudicial o excepciones para falta de novedad :

1

1

12

1. Fecha de recibo actual de la solicitud internacional propuesta:		2. Dibujos:
3. Fecha corregida de recibo actual debido a posterior pero oportuno recibo de papeles o dibujos que completan la *solicitud internacional propuesta*		
4. Fecha de recepción oportuna de las correcciones requeridas de acuerdo con el Artículo 11(2) PCT:		☐ Recibido:
5. Autoridad de Búsqueda Internacional (si dos o más son competentes): ISA/	7. Tramite de la copia de búsqueda retrasada hasta que la tasa de búsqueda se pague	☐ no recibido:
Fecha de recibo de la copia de registro por la Oficina Internacional:		

Formato PCT/RO/101(ultima hoja)(Enero 2004

13

13

Esta hoja no hace parte y no cuenta como hoja de la solicitud internacional

PCT

Solicitud Internacional No.
Fecha de Presentación Internacional

Archivo de Referencia del Agente o Solicitante:

MFCP109834/5

SOLICITANTE

MICROSOFT CORPORATION

Calculo de la tasa prescrita

1. Tasa de tramite.....300(T)
2. Tasa de búsqueda.....300(s)

La búsqueda internacional debe ser conducida por ISA/EU

(Si dos o más autoridades internacionales de búsqueda son competentes para iniciar la búsqueda internacional, indique el nombre de la autoridad que ha de llevar a cabo la búsqueda internacional.)

3. Tasa de archivo internacional

Donde los items (b) y/o (c) de la casilla No. IX aplican, ponga el Sub-total de hojas

Donde los items (b) y/o (c) de la casilla No. IX no aplican, ponga el Total de hojas.....160

(1 Primeras 30 hojas.....1134(il)

(i2) numero de hojas que pasen de 30	130	X	Tasa por hojas	=	12 1,560(i2)

(i3) componentes adicionales sólo si la secuencia de listados y/o tablas relacionadas allá se archivan en medios legibles por computador bajo la sección 801(a)(i) o ambas en esa forma y papel, bajo la sección 801(a)(ii):

400 X tasa por hoja = i3

Sume renglones i1,i2 y i3 y anote el total en 1.....U.S$2924

(Los solicitantes de ciertos estados tienen derecho a una reducción del 75% de la tasa internacional. Donde el solicitante tenga dicho derecho(o todos los solicitantes), el total ha anotar en 1 es el 25% de la tasa de archivo internacional)

4. Tasa por el documento prioritario.....40(p)

$ 3564
TOTAL

5. Total a pagar.....

Sume las cantidades en T,S,I y P y anote el total en la casilla Total

Forma de Pago

☒ Autorización para cargar a una cuenta corriente ☐ Giro postal ☐ Efectivo ☐ cupones

☒ Cheque ☐ Retiro bancario ☐ estampillas redimibles ☐ otras(especifique)

AUTORIZACIÓN PARA CARGAR (O CRÉDITO) A UNA CUENTA CORRIENTE

(Esta forma de pago puede no estar autorizada en todas las oficinas receptoras)

☐ Autorización para cargar la totalidad del pago como se indica arriba

☒ *(Esta casilla no debe marcarse sino si las condiciones para la cuenta corriente de la oficina receptora así lo permiten)* Autorización para cargar cualquier déficit o dar crédito si hay sobrepago de acuerdo a la tasa fijada arriba.

☒ Autorización para cargar el valor del documento de prioridad

Oficina receptora RO/.....EU
Cuenta corriente no.....23-3050
Fecha.....Julio 29 2004
Nombre....Steven B. Samuels
Firma.....

De la agencia Internacional

TRATADO DE COOPERACIÓN DE PATENTES

PCT

NOTIFICACIÓN CONCERNIENTE A LA ENTREGA DEL TRÁMITE DEL DOCUMENTO DE PRIORIDAD

(Instrucciones Administrativas PCT Sección 411)

30564228

ROCCI, Steven J
Woodcock Washburn LLP
One Liberty Place
46th Floor
Philadelphia PA 19103
ESTADOS UNIDOS DE AMÉRICA

RECIBIDO
Noviembre 02 200

Fecha de entrega *(día/mes/año)* 15 octubre 2004 (15 10 2004)	
Archivo de referencia del solicitante o agente MSFT 4419	**NOTIFICACIÓN IMPORTANTE**
Solicitud Internacional No PCT/USO4/024569	Fecha de entrega internacional *(día/mes/año)* 29 julio 2004 (29 07 2004)
Fecha de publicación internacional *(día/mes/año)*	Fecha de prioridad (*dayhnumh/year)* 21 August 2003 (21 08 2003)

Solicitante

MICROSOFT CORPORATION et al

1. por medio de este formato que reemplaza cualquier notificación concerniente a la entrega del trámite del documento de prioridad er anteriormente, por medio de la presente se te notifica al solicitante la fecha en que la agencia internacional recibió el documento (s relacionado con solicitudes anteriores cuya prioridad se reivindica. Al menos de que se indiquen lo contrario con las letras NR en la columna de la derecha o por un * que aparece al lado de la fecha de recibo, el documento de prioridad concerniente fue entregado tramitado a la Agencia Internacional en cumplimiento de las reglas 17 1(a) o (b).
2. (Si aplica) las letras NR que aparecen en la columna de la derecha denotan un documento de prioridad que, el día que se envió el Formato por correo no se había recibido por la Agencia Internacional bajo la Reglas 17 1(a) o (b). Donde bajo la Regla 17 1(a) el documento de prioridad debe ser entregado por el solicitante a la oficina receptora de la Agencia Internacional pero el solicitante nc entrega del documento de prioridad dentro del límite de tiempo aplicado en bajo esa Regla, se le pide al solicitante dirigirse a la Re 1(c) que dice que ninguna Oficina designada puede hacer caso omiso de la reivindicación de prioridad a concerniente antes de dar solicitante una oportunidad, tan pronto entre en la fase nacional que suministre el documento de prioridad a dentro del límite de tier que es razonable bajo las circunstancias.
3. (Si aplica) un asterisco (*) que aparece junto a la fecha de recepción en la columna de la mano derecha denotan un documento de prioridad que se entregó para trámite en la Agencia Internacional pero no cumpliendo la Regla 17 1(a) o (b) (ser documento de prk se recibido después del tiempo límite prescripto en la Regla 17 1(a) o la solicitud para preparar y trasmitir el documento de prioridac entregó en la oficina receptora después del límite de tiempo aplicado le bajó Regla 17 1(b). Aunque el documento de prioridad a los entregó en cumplimiento de la Regla 17 1(a) o (b) la Agencia Internacional sin embargo trasmitirá una copia del documento a la Ofi designada para su consideración. En caso de que dicha copia no sea aceptada por la Oficina designada como documento de priori la Reglas 17 1(c) prevé que ninguna Oficina designada puede hacer caso omiso de la reivindicación de prioridad concerniente ante dar al solicitante una oportunidad una vez que haya entrado en la fase nacional para suministrar el documento de prioridad dentro d tiempo límite que es razonable bajo las circunstancias

Fecha de Prioridad	No. de solicitud de Prioridad	País oficina regional u oficina PCT receptora	fecha en que se reciben el documento de prioridad
24 Octubre 2003 (24 10 2003)	10/693 574	US	03 Septiembre 2004 (03 09 2004)

Microsoft Patent Dept
Dic 28 2004

RECIBIDO
DEC 28 2004
Mary Ann Devine
Woodcock Washburn

La Agencia Internacional de Wipo
34 Chemins Des Colombettes
1211 Ginebra 20 Suiza

Facsimil No +41 22 740 14 35

Oficial Autorizado

BrasierJerome

Facsimile No +41 22 338 89 75
Telefono No +41 22 338 8394

15

SISTEMA Y MÉTODOS PARA LAS EXTENSIONES Y HERENCIAS DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR CON UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE

REFERENCIA CRUZADA CON APLICACIONES RELACIONADAS

[0001] Esta aplicación reivindica las prioridades de la Solicitud de Patente E.U. No. 10/693,574 (Atty Docket No. MSFT-2847) presentada en Octubre 24, 2003, y llamada "SISTEMA Y MÉTODOS PARA LAS EXTENSIONES Y HERENCIAS DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR CON UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; Solicitud de Patente E.U. No. 10/646,580 (Atty Docket No. MSFT-2735), presentada en Agosto 21, 2003, y llamada "SISTEMAS Y MÉTODOS PARA MOLDEAR DATOS EN UNA PLATAFORMA DE ALMACENAJE BASADA EN ITEMS"; la solicitud internacional No. PCT/US 03/26144 (Atty Docket No. MSFT-2779) archivadas en Augosto 21, 2003, llamada " SISTEMAS Y MÉTODOS PARA MOLDEAR DATOS EN UNA PLATAFORMA DE ALMACENAJE BASADA EN ITEMS "; contenido que se incorpora acá por referencia.

[0002] Esta aplicación se relaciona en su contenido al invento descrito en las siguientes aplicaciones comúnmente asignadas, contenido que se incorpora en el presente documento por referencia a la aplicación presente en su totalidad (parcialmente resumida acá por conveniencia): solicitud de patente E.U. No. 10/647,058 (Atty. Docket No. MSFT-1748), archivada en Augosto 21, 2003, llamada "SISTEMA Y MÉTODOS PARA REPRESENTAR UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE PERO QUE SON INDEPENDIENTES DE REPRESENTACIÓN FÍSICa "; solicitud de patente E.U No. 10/646,941 (Atty. Docket No. MSFT-1749),archivada en agosto 21, 2003, y llamada "SISTEMAS Y MÉTODOS PARA SEPARAR UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE

16

HARDWARE/SOFTWARE DE SU ORGANIZACIÓN FÍSICA"; Solicitud de Patente de E.U n No. 10/646,940 (Atty. Docket No. MSFT-1750), presentada en agosto 21, 2003, y llamada "SISTEMA Y MÉTODOS PARA LA IMPLEMENTACIÓN DE UN ESQUEMA PARA ORGANIZAR UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; Solicitud de Patente de E.U No. 10/646,632 (Atty. Docket No. MSFT-1751), presentada en agosto 21, 2003, y llamada "SISTEMA Y MÉTODOSPARA LA IMPLEMENTACIÓ DE UN ESQUEMA BÁSICO PARA PROVEER UNA ESTRUCTURA DE ALTO NIVEL PARA ORGANIZAR UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; Solicitud de Patente de E.U No. 10/646,645 (Atty. Docket No. MSFT-1752), presentada en agosto 21, 2003, llamada "SISTEMAS Y MÉTODOS PARA REPRESENTAR LAS RELACIONES ENTRE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; Solicitud de Patente de E.U No. 10/646,575 (Atty. Docket No. MSFT-2733), presentada en agosto 21, 2003, y llamada "SISTEMAS Y MÉTODOS PARA RELACIONAR CON UNA INTERFASE PROGRAMAS DE APLICACIÓN CON UNA PLATAFORMA DE ALMACENAJE BASADA EN ELEMENTOS"; Solicitud de Patente de E.U No. 10/646,646 (Atty. Docket No. MSFT-2734), presentada en agosto 21, 2003, y llamada "PLATAFORMA DE ALMACENAJE PARA ORGANIZAR, BUSCAR, Y COMPARTIR DATOS"; Solicitud de Patente de E.U No. 10/692,779 (Atty. Docket No. MSFT-2829), presentada en octubre 24, 2003, y llamada "SISTEMA Y MÉTODOS PARA LA IMPLEMENTACIÓN DE UN ESQUEMA DE IMÁGENES DIGITALES PARA ORGANIZAR UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; Solicitud de Patente de E.U No. 10/692,515 (Atty. Docket No. MSFT-2844), presentada en octubre 24, 2003, y llamada "SISTEMAS Y MÉTODOS PARA OFRECER UN SERVICIO DE SINCRONIZACIÓN DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE "; Solicitud de Patente de E.UNo. 10/692,508 (Atty. Docket No. MSFT-2845), presentada en

dicha fecha, llamada "SISTEMAS Y MÉTODOS PARA OFRECER UN SERVICIO DE SINCRONIZACIÓN RELACIÓNAL Y JERÁRQUICO PARA UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE"; y Solicitud de Patente de E.U No. 10/693,362 (Atty. Docket No. MSFT-2846), presentada en octubre 24, 2003, y llamada "SISTEMAS Y MÉTODOS PARA LA IMPLEMENTACIÓN DE UN ESQUEMA DE SINCRONIZACIÓN DE UNIDADES DE INFORMACIÓN QUE SE PUEDEN ADMINISTRAR POR UN SISTEMA DE INTERFASE HARDWARE/SOFTWARE".

CAMPO DEL INVENTO

[0003] El invento presente se relaciona en general con el campo de almacenaje de información y su recuperación, y, en particular, con una plataforma activa de almacenaje para organizar, buscar, y compartir diferentes tipos de datos en un sistema computarizado. Varias implementación del invento presente están dirigidas al uso de métodos de extensión y herencia utilizados por un sistema de interfase hardware/software para administrar datos.

ANTECEDENTES

[0004] La capacidad individual de los discos ha venido creciendo aproximadamente en un 70% anual en los últimos diez años. La ley de Moore predijo acertadamente el tremendo aumento en el poder de la unidad procesadora (CPU) que ha ocurrido a través de los años. Tecnologías alambradas e inalámbricas nos han dado una tremenda capacidad de conexión y ancho de banda. Presumiendo que esta tendencia continuará, dentro de pocos años una computador portátil promedio tendrá una capacidad de un terabite (TB) de almacenaje con millones de archivos, y discos de 500 gigabites (GB) serán comunes.

[0005] Los consumidores utilizan sus computadores más que todo para comunicarse y organizar su información personal, así sea información personal tradicional(PIM) o medios tales como música digital y fotografías. La cantidad de contenido digital, y la habilidad para almacenar bits crudos, ha aumentado tremendamente; sin embargo los métodos disponibles a los consumidores para organizar y unificar estos datos no se ha mantenido. Trabajadores con conocimiento pasan gran cantidad de tiempo administrando y compartiendo información, y algunos estudios estiman que un trabajador de conocimiento de hasta 15-25% de su tiempo en actividades no productivas relacionadas con la información. Otros estudios estiman que un trabajador con conocimientos gasta hasta unas 2.5 horas al día buscando información.

[0006] Los departamentos de tecnología de información y los diseñadores invierten una gran cantidad de tiempo y de dinero en construir sus propios almacenes de datos para abstracciones comunes de almacenaje para representar cosas tales como ente, sitios, tiempo, y eventos. No sólo esto resulta en un trabajo doble, pero también hereda islas de datos comunes sin ningún mecanismo de búsqueda común o para compartir los datos. Considere nomás cuántas libretas de direcciones pueden existir en un computador corriendo en un sistema operativo Microsoft Windows. Muchas aplicaciones, tales como clientes de correo y programas de contabilidad personal, mantienen libretas de direcciones individuales, y hay poco compartir entre las aplicaciones de los datos almacenados en sus libretas de direcciones.

En consecuencia, un programa de contabilidad (como Money de Microsoft) no comparte las direcciones de los acreedores con las direcciones que se mantienen en el folder de contactos del correo electrónico (como el Outlook de Microsoft). De hecho, muchos usuarios tienen múltiples dispositivos y deben sincronizar lógicamente sus datos personales entre ellos y a través de una gran variedad de fuentes adicionales, incluyendo teléfonos celulares hasta servicios comerciales como el MSN y AOL; sin embargo, la colaboración de documentos compartidos se consigue más que todo anexando documentos a mensajes de correo electrónico -esto es en forma manual e ineficiente.

[0007] Una razón para ésta falta de colaboración es que los enfoques tradicionales a la organización de información en un sistema de computación se ha centrado en el uso de

sistemas de archivos-folder-directorio ("sistemas de archivos") para organizar una pluralidad de archivos en directorios jerárquicos de folders basados en abstracciones de la organización física del medio de almacenaje utilizado para guardar los archivos. A los sistemas operativos Multic, desarrollados en en los años sesenta, se les puede acreditar el desarrollo del uso de archivos, folders, y directorios para administrar unidades almacenarles de datos en el nivel operacional del sistema. Específicamente, los Multics utilizaban direcciones simbólicas dentro de una jerarquía de archivos (por lo tanto introduciendo la idea de un path) donde las direcciones físicas de los archivos no eran transparentes al usuario (aplicaciones e usuarios finales). Este sistema de archivos no se preocupaba con el formato individual del archivo, y la relación entre archivos se consideraba irrelevante en el nivel operacional del sistema (esto es, fueron de la localidad del archivo dentro de la jerarquía). Desde que llegaron los Multics, los datos almacenables se organizaron en archivos, folders, y directorios en el nivel operacional del sistema. Estos archivos generalmente incluyendo la jerarquía en sí (el "directorio") encarnada en un archivo especial en el sistema de archivos mantienen. Este directorio, a su vez, mantienen una lista de entradas correspondiente a todos los demás archivos en el directorio y la localidad nodal de tal archivo en la jerarquía (aquí llamado folder). Este ha sido el estado del arte por aproximadamente 40 años.

[0008] Sin embargo, a pesar de que provee una representación razonable de la información almacenada en el sistema de almacenaje físico del computador, un sistema de archivos es sin embargo una abstracción del sistema físico de almacenaje, y por lo tanto la utilización de estos archivos requiere un nivel de indirección (interpretación) entre lo que el usuario manipula (unidades que tienen contexto, rasgos, y una relación a las otras unidades) y que el sistema operativo provee (archivos, folders, y directorios). En consecuencia, los usuarios (aplicaciones y/o usuarios finales) no tienen alternativa más que forzar las unidades de información en una estructura de sistema de archivos aún que hacerlo sea ineficiente, inconsistente, o indeseable. Es más, los sistemas de archivo existentes saben poco acerca de la estructura de los datos almacenados en archivos individuales, y por esto, la mayor parte de la información permanece encerrada en archivos que sólo se pueden acceder (y comprensivamente) a las aplicaciones que los escribieron.

En consecuencia, está ausencia de descripción esquemática de la información, y mecanismos para administrar la información, llevan a la creación de bodegas de datos, con poco compartir entre las bodegas individuales. Por ejemplo, muchos usuarios de computadores personales (PC) tienen más de cinco almacenes distintos que contienen información acerca de personas con las que interactúan en algún nivel -por ejemplo, contactos en Outlook, direcciones de cuentas en línea, la libreta de direcciones de Windows, acreedores de Quicken, una lista de amigos en mensajería instantánea (IM)- porque la organización de estos archivos presenta un reto significativo a estos usuarios de PC. Porque la mayoría de sistemas de archivos utilizan una metáfora de folderes anidados para organizar los archivos y folderes, en la medida que el número de archivos aumenta en el esfuerzo necesario para mantener un esquema de organización que sea flexible y eficiente se hace más difícil. En esta situación, sería muy útil tener una clasificación múltiple de un archivo individual; sin embargo, utilizando vínculos suaves o duros en el sistema de archivo existente es engorroso y difícil de mantener.

[0009] En el pasado se han hecho varios intentos fracasados de solucionar estas limitaciones en los sistemas de archivos. Algunos de estos intentos han utilizado la memoria referencial para proveer un mecanismo a través del cual los datos se pueden acceder a diferencia de una dirección física. Sin embargo estos esfuerzos han fracasado porque mientras el contenido de memoria referencial ha probado ser útil en pequeña escala por dispositivos tales como caches y unidades de administración de memoria, el uso a gran escala por dispositivos tales como medios de almacenaje físicos no ha sido posible por varias razones, y por lo tanto esa solución simplemente no existe. Otros intentos utilizando sistemas de bases de datos orientadas a los objetos (OODB) se han hecho, pero estos intentos, mientras tenían características de bases de datos fuertes y buenas representaciones no basadas en archivos, no eran efectivas en manejar la representación de los archivos y no podían aumentar la velocidad, eficiencia, y simplicidad de la estructura jerárquica basada en archivos y folderes en el nivel del sistema de interfase hardware/software. Otros intentos, como los que trataron de utilizar SmallTalk (y otros derivados), probaron ser muy efectivos en el manejo de representaciones de archivos y no-archivos pero carecían de los rasgos de base de datos necesarios para organizar en forma eficiente y utilizar las relaciones

que existían entre los distintos archivos de datos, y por lo tanto la eficiencia general de tal sistema era inaceptable. Sin embargo otros intentos de usar BeOs (y otras investigaciones de tales sistemas) probaron ser inadecuadas en el manejo de representaciones de no-archivos -la misma limitación de los sistemas de archivos tradicionales -a pesar de que eran capaces de representar en forma adecuada los archivos mientras proporcionaban suficientes características de bases de datos.

[0010] La tecnología de las bases de datos es otra área en el arte en donde retos similares existen. Por ejemplo, mientras que el modelo de bases de datos relacionados ha tenido un gran éxito comercial, en verdad vendedores independientes de software (ISV) generalmente ejercitan una pequeña parte de la funcionalidad disponible en software de bases de datos relacionales (como el servidor SQL de Microsoft). En cambio, en la mayoría de las interacciones de una aplicación con nuestro producto es en la forma de "obtener" y "poner". Aunque hay un número de razones aparentes para esto -como ser un agnóstico de las bases de datos por las plataformas- una razón primordial que con frecuencia pasa desapercibida es que las bases de datos no necesariamente proveen la abstracción exacta que la mayoría de vendedores aplicaciones de negocios realmente necesita. Por ejemplo, mientras en el mundo real tienen la noción de "elementos", tales como "clientes" o "órdenes" (junto con una orden empotrada "elementos de línea" como un elemento de y por sí mismo), bases de datos relacionales únicamente hablan en términos de tablas y filas. En consecuencia, mientras la aplicación puede querer tener un aspecto de consistencia, seguridad, llaves, y/o gatillos en el nivel de los elementos (para nombrar unos cuantos), generalmente las bases de datos proveen estos rasgos únicamente en el nivel tablas/fila. Mientras que esto puede funcionar muy bien mientras cada elemento se mapie a una sola fila en una tabla en la base de datos, en el caso de una orden con elementos de líneas múltiples puede haber razón para que un elemento actualmente se mapie a tablas múltiples y, cuando ése es el caso, el sistema simple de base de datos relacionados no provee la abstracción correcta. En consecuencia, una aplicación debe construir una lógica encima de la base de datos para suministrar estas abstracciones básicas. En otras palabras, el modelo básico relacional no provee una plataforma suficiente para el almacenaje de datos en donde aplicaciones de alto nivel se pueden desarrollar porque el modelo básico relacional requiere un nivel de

ausencia de dirección entre la aplicación y el sistema de almacenaje -donde la estructura semántica de los datos puede únicamente ser visible en la aplicación en ciertos instantes. Mientras algunos vendedores de bases de datos están construyendo una funcionalidad a alto nivel en sus productos-tal como ofrecer capacidades relacionales de objetos, nuevos modelos de organización, y así- ninguno suministra por el momento la solución comprensiva que se requiere, donde una solución verdaderamente amplia es una que provee tanto un modelo útil de abstracciones de datos (tales como "elementos", "extensiones", "relaciones", y demás) para abstracciones de dominios útiles (tales como "personas", "localidad", "eventos", etc.).

[0011] En vista de las deficiencias mencionadas en el almacenaje de datos existentes y tecnologías de bases de datos, existe la necesidad para una plataforma de almacenaje nueva que brinde una habilidad mejorada para organizar, buscar, y compartir todo tipo de datos en un sistema de computación -l plataforma de almacenaje que extienda y amplíe la plataforma de datos más allá del sistema de archivos existente y sistemas de bases de datos, y que esté diseñado para hacer el almacén de todo tipo de datos. El presente invento, junto con los inventos relacionados e incorporados por referencia anteriormente a cada, satisface esta necesidad. En particular, el presente invento suministra un método de extensiones y herencias para objetos manipulados por un sistema de interfase hardware/software.

RESUMEN

[0012] El siguiente resumen nos da una visión general de los varios aspectos del invento descritos en el contexto de los inventos relacionados e incorporados por referencia anteriormente (los "inventos relacionados"). Este resumen no tiene como propósito dar una descripción exhaustiva de todos los aspectos importantes del invento, ni tampoco definir el alcance del invento. Más bien, este resumen tiene por objeto servir como una introducción a la descripción detallada y figuras que siguen.

[0013] El invento presente, así como los inventos relacionados, están dirigidos colectivamente a una plataforma de almacenaje para organizar, buscar, y compartir datos.

La plataforma de almacenaje del invento presente extiende y amplía el concepto de almacenaje de datos más allá de los sistemas de archivos existentes y sistemas de bases de datos, y está diseñado para almacenar todo tipo de datos incluyendo datos estructurados, no estructurados, o semi-estructurados.

[0014] La plataforma de almacenaje del presente invento consta de un almacén de datos implementado en un motor de base de datos. El motor de base de datos comprende un motor de base de datos relaciónal con extensiones de objetos relacionales. El almacén de datos implementado un modelo de datos que apoya la organización, búsqueda, compartir, sincronización, y seguridad de los datos. Tipos de datos específicos se describen en esquemas, y la plataforma suministra un mecanismo para extender el conjunto de esquemas para definir nuevos tipos de datos (esencialmente subtipos de los tipos básicos suministrados por el esquema). Una capacidad de sincronización facilita el compartir de datos entre usuarios y sistemas. Capacidades del tipo sistema de archivos se suministra para permitir la Inter operabilidad del almacén de datos con sistemas de archivos existentes pero sin las limitaciones de estos sistemas de archivos tradicionales. Un mecanismo de seguimiento de cambios suministra la habilidad para seguir los cambios en el almacén de datos. La plataforma de almacenaje comprende adicionalmente un conjunto de interfases de programas de aplicación que le permiten a las aplicaciones acceder a las capacidades mencionadas de la plataforma de almacenaje y acceder a los esquemas de datos descritos.

[0015] El modelo de datos implementados por el almacén de datos define unidades de datos almacenados en términos de elementos, items y relaciones. Un item es una unidad de datos almacenados en en el almacén de datos y puede comprender uno o más elementos y relaciones. Un elemento es una instancia de un tipo que comprende una o más campos (también llamados a cada propiedad). Una relación es un vínculo entre dos items. (Como se utilizarán a cada, estos y otros términos específicos se pueden capitalizar para distinguirlos de otros términos utilizados en cercana proximidad; sin embargo, no hay intención de distinguir entre un término capitalizado, por ejemplo "Item", y el mismo término cuando no se capitaliza, por ejemplo "item", y esta distinción no se debe presumir o implicar).

[0016] El sistema de computación consta además de una pluralidad de items donde cada items constituye una unidad discreta de almacenaje de información que se puede manipular

por un sistema de interfase hardware/software; una pluralidad de folderes de items que constituye una estructura organizada de dichos items; un sistema de interfase hardware/software para manipular una pluralidad de Items donde cada Item pertenece a por lo menos un Folder de Item y puede pertenecer a más de un Folder de Item.

[0017] Un Item o algunos de los valores de las propiedades del Item se pueden computar en forma dinámica a diferencia de derivarse de un almacén persistente. En otras palabras, el sistema de interfase hardware/software no requiere que el item se guarde, y algunas operaciones son posibles como la habilidad de enumerar el conjunto de Items actuales por la habilidad de recuperar un Item dado su identificador (que se describe en más detalle en la sección se describe la interfase de programación de la aplicación, o API) de la plataforma de almacenaje -por ejemplo, un item puede ser la localidad de un teléfono celular o la lectura de temperatura en un sensor de temperatura. El sistema de interfase hardware/software puede manipular una pluralidad de items, y puede comprender, aún más, Items interconectados por una pluralidad de Relaciones administradas por el sistema de interfase hardware/software.

[0018] Un sistema de interfase hardware/software para el sistema de computación comprende un esquema básico para definir un conjunto básico de Items que dichosistema de interfase hardware/software entiende y puede procesar directamente en una forma predeterminada y predecible.

Para manipular una pluralidad de Items, un sistema de computación interconecta dichos Items con una pluralidad de Relaciones y administra dichas Relaciones en el nivel del sistema de interfase hardware/software.

[0019] El API de la plataforma de almacenaje suministra una clase de datos para cada item, extensiones de item, y relaciones definidas en el conjunto de esquemas de la plataforma de almacenaje. Adicionalmente, la interfase de programación de la aplicación suministra un conjunto de marcos de clases que definen un conjunto común de comportamientos para esa clase de datos y que, junto con las clases de datos, suministran el modelo básico de programación para la plataforma de almacenaje API. La plataforma de almacenaje API suministra un modelo de búsqueda simplificada que le permiten a los programadores de la aplicación formará búsquedas pasadas en varias propiedades de los

items en el almacén de datos, en una manera que aísla al programador de la aplicación de los detalles del lenguaje de búsqueda del motor subyacíente es la base de datos. La plataforma de almacenaje API también recoge los cambios hechos a un item por un programa de aplicación y luego los organiza en las actualizaciones correctas requeridas por el motor de la base de datos (o cualquiera motor de almacenaje) en el cual la base de datos esta implementada. Esto permiten a los programadores de la aplicación hacer cambios a un item en la memoria, mientras dejan la complejidad de las actualizaciones del almacén de datos a la API.

[0020] A través de su almacenaje básico común y datos esquematizados, la plataforma de almacenamiento del presente invento permite el desarrollo de aplicaciones más eficientes para el consumidor, trabajadores con conocimiento y empresas. Ofrece una interfase de programación de aplicaciones rica y extensible que no sólo hace disponibles las capacidades inherentes en su modelo de datos, pero también comprende y extiende los sistemas de archivos existentes y métodos de acceso a las bases de datos.

[0021] En vista de esta amplia estructura de inventos interrelacionados (descritos en detalle en la sección 2 de la Descripción Detallada), el presente invento está dirigido específicamente al uso de Extensiones para extender la funcionalidad de los Items y tipos de Propiedades, así como el uso de Herencias para facilitar la búsqueda de organización eficiente entre Items relacionados (descritos en detalle en la sección 3 de la Descripción Detallada). Otros rasgos y ventajas del invento pueden ser aparentes de la siguiente descripción detallada del invento y de los dibujos que la compañía.

DESCRIPCIÓN BREVE DE LOS DIBUJOS

[0022] El resumen anterior, así como la siguiente descripción detallada del invento, se entiende mejor cuando se lee en conjunto con los dibujos anexos. Con motivo de ilustrar el invento, se muestran en los dibujos implementaciones ejemplares de varios aspectos del invento; sin embargo, el invento no está limitado a los métodos específicos e instrumentos detallados. En los dibujos:

[0023] La figura 1 es un diagrama de bloque que representa un sistema de computación en donde aspectos del presente invento se pueden incorporar;

[0024] La figura 2 es un diagrama de bloque que muestra un sistema de computación dividido en tres grupos de componentes: el componente de hardware, el componente de sistema de interfase hardware/software , y un componente de programas de aplicación;

[0025] La figura 2A muestra una estructura jerárquica basada en un árbol para grupos de archivos en folderes en un directorio en un sistema operativo basado en archivos;

[0026] La figura 3 es un diagrama de bloque que muestra una plataforma de almacenaje;

[0027] La figura 4 muestra una una relación estructural entre Items, Folderes de Items, y Categorías;

[0028] La figura 5A es su diagrama de bloque que muestra la estructura de una Item;

[0029] La figura 5B es su diagrama de bloque que muestra los tipos de propiedad compleja de una Item de la figura 5A;

[0030] La figura 5C es un diagrama de bloque que muestra el Items "localidad" donde sus tipos complejos se describen (listados explícitamente);

[0031] La figura 6A muestra un Item como un subtipo de un Item encontrado en el Esquema Base;

[0032] La figura 6 de es su diagrama de bloque que muestra un subtipo de Item de la figura 6 a donde el tipo heredado se lista en forma explícita (adicional a las propiedades inmediatas);

[0033] La figura 7 es un diagrama de bloque que muestra el esquema básico que incluye sus dos clases de tipos de niveles superiores, Item y PropiedadBase, y el esquema de tipos básicos derivados;

[0034] La figura 8 a es un diagrama de bloque que muestra Items en el Esquema Básico;

[0035] La figura 8 de es su diagrama de bloque que muestra los tipos de propiedad en el Esquema Básico;

[0036] La figura 9 es un diagrama de bloque que muestra un Folder De Item y sus Items miembros;

[0037] La figura 10 es un diagrama de bloque que muestra una Categorías (que en, de nuevo, es un Item en sí), sus Items miembros, y las Relaciones que interconectan entre la Categorías y sus Items miembros;

[0038] La figura 11 es un diagrama que muestran un tipo de jerarquía de referencia de el modelo de datos de una plataforma de almacenamiento;

[0039] La figura 12 es un diagrama que muestran cómo se clasifican las relaciones;

[0040] La figura 13 es un diagrama que muestra el mecanismo de notificación;

[0041] La figura 14 es un diagrama que muestran un ejemplo en donde dos transacciones están insertando un record nuevo en el mismo árbol-B;

[0042] La figura 15 muestra un proceso de detección de cambio de datos;

[0043] La figura 16 muestra una árbol de directorio ejemplar;

[0044] La figura 17 muestra un ejemplo en donde un folder existente de un sistema de archivos basado en directorio se mueve a la plataforma de almacenaje del almacén de datos;

[0045] La figura 18 muestra el concepto de unos Folderes de Contención;

[0046] La figura 19 muestra la arquitectura básica de una plataforma de almacenaje API;

[0047] La figura de ente representa en forma esquemática los componentes varios de la plataforma de almacenaje API;

[0048] La figura 21A es una representación gráfica de un esquema ejemplar de un Item Contacto;

[0049] la figura 21B es una representación gráfica de los Elementos de un esquema ejemplar de Item Contacto de la figura 21A;

[0050] La figura 22 muestra el marco de tiempo de ejecución de una plataforma de almacenaje API;

[0051] La figura 23 muestras la ejecución de una operación de "FindAll";

[0052] La figura 24 muestras del proceso mediante el cual las clases en la plataforma de almacenaje API se generan el esquema de la plataforma de almacenaje;

[0053] La figura 25 muestra un esquema que se basa en el Archivo API;

[0054] La figura en 26 es un diagrama que muestra un formato enmascarado de acceso utilizado para la seguridad de los datos;

[0055] La figura 27 (parte a, b, y c) muestran una región nueva idéntica y protegida que está siendo excluida de una región de seguridad existente;

[0056] La figura 28 es un diagrama que muestra el concepto de una vista de búsqueda de una Item;

[0057] La figura 29 es un diagrama que muestra una jerarquía de Items ejemplar;

[0058] La figura 30A muestra una interface Interface 1 como un conducto a través del cual el primer y segundo segmento del código se comunica;

[0059] La figura 30B muestra una interfase que comprende las interfases objeto I1 e I2 que en capacitan al primer y segundo segmento del código de un sistema para comunicarse a través del medio M;

[0060] La figura 31A muestra como la función provista por una interfase Interfase 1 se puede subdividir para convertir las comunicaciones de la interfase en múltiples interfases Interfase 1A, Interfase 1B, Interfase 1C;

[0061] La figura 31B muestra como la función provista por la interfase I1 se puede subdividir en múltiples interfases I1a,I1b,I1c;

[0062] La figura 32A muestra un escenario donde una precisión insignificante de parámetro puede ser ignorada o cambiada con un parámetro arbitrario;

[0063] La figura 32B muestra un escenario donde una interfase es reemplazada por una interfase sustituta que está definida para ignorar o añadir parámetros a la interfase;

[0064] La figura 33A muestra un escenario donde los Segmentos de Código primero y segundo se fusionan en un módulo que los contiene a ambos,

[0065] La figura 33B muestra un escenario donde parte o toda la interfase se pueden escribir en línea dentro de otra interfase para formar una interfase de fusion;

[0066] La figura 34A muestra como una o mas piezas de midleware pueden convertir las comunicaciones en la primera interfase para conformarlas a una o mas interfases diferentes;

[0067] La figura 34B muestra como un segmento de código se puede introducir con una interfase para recibir la comunicación de una interfase pero transmitir la funcionalidad a una segunda y tercera interfase;

[0068] La figura 35A muestra como un compilador a-tiempo (JIT) puede convertir las comunicaciones de un segmento del código a otros segmento del código;

[0069] La figura 35B muestra un método JIT que reescribe dinámicamente una o más interfases puede aplicarse a un factor dinámico o de lo contrario alterar dicha interfase;

[0070] La figura 36 muestra una serie de Items Interrelacionados y un subconjunto de sus Relaciones;

[0071] La figura 37 A muestra las inconveniencias de sub tipificar un Items por razones específicas de aplicación;

[0072] La figura 37 B muestra una solución parcial al problema de sub tipificar estándar; y

[0073] La figura 37C muestra una implementación del presente invento para extender un Items con una Extensión que es distinta y separada de que el Contacto en sí, y por lo tanto permite una funcionalidad multitipificadora.

(El resto de la hoja se deja intencionalmente en blanco)

DESCRIPCIÓN DETALLADA

TABLA DE CONTENIDO

I. INTRODUCCIÓN 19

A. UN AMBIENTE EJEMPLAR DE COMPUTACIÓN 19

B. ALMACENAJE TRADICIONAL BASADO EN ARCHIVOS 23

II. PLATAFORMA DE ALMACENAJE WINFS PARA ORGANIZAR, BUSCAR, Y COMPARTIR DATOS 25

C. GLOSARIO 25

D. PERSPECTIVA GENERAL DE UNA PLATAFORMA DE ALMACENAJE 26

E. EL MODELO DE DATOS 27

1. Items 29

2. Identificación de items 33

3. Folderes de items y Categorías 33

4. Esquemas 35

A. Esquema básico 35

B. Esquemas principal 36

5. Relaciones 38

a. Declaración de Relaciones 39

b. Relaciones de pertenencia 40

c. Relaciones empotradas 41

d. Relaciones de Referencia 42

e. Reglas y fijación 43

f. Ordenamiento de las Relaciones 43

6. Extensibilidad 49

a. Extensiones de Items 50

b. Extendiendo tipos de elementos anidados 54

F. MOTOR DE BASE DE DATOS 56

1. Implementación del Almacén de Datos utilizando un UDT 57

2. Mapeo de Items 58

3. Mapeo de Extensiones 59
4. Mapeo de Elementos Anidados 59
5. Identidad de los Objetos 59
6. Nombrado de Objetos SQL 59
7. Nombrado de columnas 60
8. Vistas de búsqueda 61
a. Item 62
1. Vista del Item Maestro 62
2. Vista del Item de búsqueda 62
b. Extensiones de Items 63
1. Vista de la búsqueda de Extensión Maestra 63
2. Vista de la búsqueda de Extensión digitada 64
c. Elementos anidados 64
d. Relaciones 65
1.Vista de busqueda de las Relaciones Maestras 65
2. Vista de busqueda de una instancia de Relaciones 65
e. -66-
9. Actualizaciones 66
10. Cambio de Seguimientos y Lapidas
a. Cambio de Seguimientos 67
1. Cambio de seguimiento en vista de búsqueda "Master" 67
2. cambio de seguimiento en vista de de búsqueda "Digitada"68
b. Lapidas 69
1. Lapidas de Items 69
2. Lapidas de Extensiones 70
3. Lapidas de Relaciones 70
4. Limpieza de Lapidas 71
11. Ayudante de APIs y Funciones 71
a. Función (Almacenaje.Sistema) GetItem71
b. Función (Almacenaje.Sistema). GetExtension 71

c. Función (Almacenaje.Sistema).GetRelationship 72

12. Metadatos 72

1. Esquema de metadatos
2. Instancia de metadatos

G. SEGURIDAD 72

H. NOTIFICACIONES Y CAMBIO DE SEGUIMIENTO 73

I. SINCRONIZACIÓN 74

1. Sincronización de plataforma de almacenaje a plataforma de almacenaje 74
   a. Aplicación de control de Sincronización (Sync) 75
   b. Anotación de esquema 76
   c. Configuración de Sync 77
      i. Fólder comunitario-Mapeos 77
      ii. Reseñas 78
      iii. Horarios 79
   d. Manejo de conflictos 79
      i. Detección de conflictos 80
         a) Conflictos conocidos
         b) Conflictos de constreñimiento 80
      ii. Conflictos en el procesamiento
         a) Resolución automática de conflictos
         b) Logging de conflictos 81
         c) inspeccion de conflictos y resolución 82
         d) convergencia de replicas y propagación de resolución de conflictos 82
2. Sincronizando con plataformas de no almacenaje de datos
   a. Servicio Sync 84
      1. Cambio de enumeración 84
      2. Cambio de aplicación 85
      3. Resolución de conflictos 85
   b. Implementación del adaptador 86

c. Seguridad 86

d. Administración 87

J. INTEROPERABILIDAD DE LOS SISTEMAS TRADICIONALES DE ARCHIVOS 87

K. PLATAFORMAS DE ALMACENAJE API 88

III. EXTENSIONES Y HERENCIAS 95

A. TIPOS DE SISTEMAS 95

B. TIPOS DE FAMILIAS 100

1. Tipos de elementos anidados 101

2. Tipos de Items 101

3. Tipos de relaciones 101

a. Semántica de las relaciones 103

b. Reglas de las relaciones y constreñimientos 105

4. Tipos de Extensiones

C. FUNCIONALIDAD MEJORADA 107

1. Herencias 107

2. Extensiones 108

IV CONCLUSIÓN 109

I. INTRODUCCIÓN

[0074] El tema del presente invento se describe específicamente para cumplir con los requerimientos estatutarios. Sin embargo, la descripcion en si no tiene la intención de limitar el alcance de esta patente. Más bien, los inventores han considerado que el tema que se reivindica puede también implementarse de otras maneras para incluir distintas etapas o combinaciones de etapas similares a las que se describe en este documento, en conjunto con otras tecnologías presentes y futuras. Aún más, aunque el termino "etapas" se puede usar acá como connotación de distintos elementos de métodos empleados, el termino no se debe interpretar como implicando algun orden particular entre las varias etapas aquí presentadas al menos y excepto cuando el orden individual de las etapas se describe en forma explícita.

A. UN AMBIENTE EJEMPLAR DE COMPUTACIÓN

[0075] Varias implementaciones del presente invento se pueden ejecutar en un computador. La figura 1 y la siguiente discusión tienen como intención dar una descripción general corta de un ambiente de computación adecuado en el cual este invento se puede implementar. Aunque no se requiere, varios aspectos del invento se pueden describir en un contexto general de instrucciones ejecutables de computador, como módulos de programas que un computador puede ejecutar, tal como una estación de servicio al cliente o un servidor. Generalmente, los módulos de programas incluyen rutinas, programas, objetos, componentes, estructuras de datos y demás que ejecutan tareas particulares o implementan tipos de datos abstractos particulares. Aún más, el invento se puede practicar con otras configuraciones de sistemas de computación, incluyendo dispositivos de mano, sistemas con multiprocesadores, dispositivos electrónicos programables basados en microprocesadores, redes de PCs, minicomputadores, estaciones y demás. El invento también se puede poner en práctica en ambientes de computación distribuidos donde las tareas se ejecutan a través de un proceso remoto que está vinculado a través de una red de

comunicación. En un ambiente de computación distribuida, los módulos de programas pueden estar localizados tanto en dispositivos de memoria remota como en la local.

[0076] Cómo se muestran en la figura 1, un sistema de computación ejemplar de propósito general incluye un computador personal convencional 20 o algo parecido, una unidad procesadora 21, una memoria de sistema 22, un bus de sistema 23, que se acoplan con varios componentes del sistema incluyendo la memoria de sistema a la unidad procesadora 21. El bus del sistema 23 puede tener una de varias estructuras de bus que incluyen un bus de memoria o controlador de memoria, un bus periférico y un bus local que puede utilizar una variedad de arquitecturas. La memoria de sistemas incluye una memoria de sólo lectura (ROM) 24 y una memoria de acceso aleatorio (RAM) 25. Un sistema básico de entrada/salida 26 (BIOS), que contienen la rutina básica que ayuda a transferir información entre los elementos dentro del computador personal 20, como el arranque, que se almacenan en ROM 24 del computador personal 20 puede incluir adicionalmente un drive de disco duro 27 para leer y escribir al disco duro (que no se muestran), un drive de disco magnético 28 para leer y escribir a un disco magnético removible 29, y un drive de disco óptico para leer y escribir a un disco óptico removible 31 tal como un CD ROM o algún otro medio óptico. El disco duro 27, el drive de disco magnético 28, el drive de disco óptico 30, mundo están conectados al bus del sistemas 23 por una interfase de drive de disco duro 32, una interfase de drive de disco magnético 33, y una interfase de drive de disco óptico 34, respectivamente. Los drives y sus medios legibles por computador proveen un almacenaje no volátil de instrucciones legibles por computador, estructuras de datos, módulos de programas y otros datos para el computador personal 20. Aunque el ambiente ejemplar descrito acá emplea discos duros, discos magnéticos removibles 29 y discos ópticos removibles 31, lo deben apreciar aquellos diestros en el arte que otros tipos de medios legibles por computador que pueden almacenar datos que son accesibles a un computador, tales como casetes magnéticos, tarjetas de memoria flash, discos digitales de vídeo, cartuchos Bernoulli, memorias de acceso aleatorio (RAM), memorias de sólo lectura (ROM) y cosas por el estilo se pueden utilizar en un ambiente operativo ejemplar. De la misma manera, un ambiente ejemplar también puede incluir muchos tipos de dispositivos

de monitoreo tales como sensores de calor y seguridad o alarmas contra incendios, y otras fuentes de información.

[0077] Un número de módulos de programas se pueden almacenar en el disco duro, el disco magnético 29, en el disco óptico 31, en ROM 24 o en RAM 25, incluyendo un sistema operativo 35, uno o más programas de aplicación 36, otros módulos de programas 37 y datos de programas 38. Un usuario puede entrar comandos de información dentro del computador personal 20 a través de dispositivos de entrada como un teclado 40 o un dispositivo de señalización 42. Otros dispositivos de entrada (que no se muestran) pueden incluir un micrófono, un joystick, pad de juegos, disco satelital, escáner y demás. Estos y otros dispositivos de entrada frecuentemente están conectados a la unidad procesadoras 21 a través de una interfase de puertos seriales 46 que se acopla al bus del sistemas, pero que pueden estar conectados por otras interfases, tales como puertos paralelos, puertos de juegos o un bus serial universal (USB). También se conecta al bus del sistemas 23 un monitor 47 o algún otro tipo de dispositivo de visualización a través de una interfase, tal como un adaptador de vídeo 48.

Adicional al monitor 47, los computadores personales incluyen típicamente otros dispositivos periféricos de salida (que no se muestra), tales como parlantes o impresoras. El sistema ejemplar de la figura 1 también incluye un adaptador anfitrión 55, una Sistema de bus de Interfase Pequeña de Computación (SCSI) 56, si el dispositivos de almacenaje es externo 62 se conecta al bus SCSI 56.

[0078] El computador personal 20 puede operar en un ambiente de red utilizando conexiones lógicas con uno o más computadores remotos, como el computador remoto 49. El computador remoto 49 puede ser otro computador personal, un servidor, un ruteador, un PC, el dispositivo de un colega u otro nodo común de red, y típicamente incluye muchos o todos de los elementos descritos anteriormente relativos al computador personal 20, aunque únicamente un dispositivo de almacenaje de memoria 50 se ha ilustrado en la figura 1. Las conexiones lógicas que se muestran en la figura 1 incluyen una red de área local (LAN) 51 y una red de área amplia (WAN) 52. Éstos ambientes de red son comunes en las oficinas, negocios a través de las redes, intranets e Internet.

[0079] Cuando se utiliza un ambiente de red LAN, el computador personal 20 está conectado al LAN 51 a través de una interfase del red o adaptador 53. Cuando se utiliza en un ambiente de red WAN, el computador personal 20 incluye típicamente un modem 54 u otros medios para establecer comunicaciones con la red de área amplia 52, tal como el Internet. El modem 54, que puede ser interno o externo, se conecta al bus del sistema 23 a través de una interfase de puerto serial 46. En un ambiente de red, los módulos de programas que se muestran relativos al computador personal 20, o porciones de él, pueden almacenarse en un dispositivo de almacenaje de memoria remoto. Se podrá apreciar que las conexiones de red que se muestran son ejemplares y que se pueden utilizar otros medios para establecer vínculos de comunicación entre computadores.

[0080] Cómo se ilustró en el diagrama de bloque de la figura 2, un sistema de computación 200 puede dividirse aproximadamente en tres grupos de componentes: un componente de hardware 202, un componente de interfase de sistema 204, y un componente de programas de aplicación 206 (llamado también "componente del usuario" o "componente de software" en ciertos contextos en este documento).

[0081] En varias implementaciones de un sistema de computación 200, y refiriéndonos de nuevo al componente de hardware de la figura 1 puede comprender una unidad central de procesamiento (CPU) 21, la memoria (ambas ROM 24 y RAM 25), un sistema básico de entrada/salida (BIOS) 26, y varios dispositivos de entrada/salida (I/O) tales como el teclado 40, un ratón 42, un monitor 47 y/o una impresora (no se muestra), entre otras cosas. El componente de hardware 202 comprende una infraestructura física básica para el sistema de computación 200.

[0082] El componente de aplicaciones de programas 206 comprende varios programas de software incluyendo pero no limitado a compiladores, bases de datos, procesadores de palabras, programas de negocios, videojuegos, y demás. Los programas de aplicación proveen los medios mediante los cuales los recursos del computador se utilizan para resolver problemas, proveer soluciones, y procesar datos para varios usuarios (máquinas, otros sistemas de computación, y/o usuarios finales).

[0083] El sistema de interfase hardware/software 204 comprende (y, en algunas implementaciones, consiste únicamente de) un sistema operativo que consiste de, en la

mayoría de casos, una cubierta y un kernel. El "sistema operativo" (OS) es un programa especial que actúa como intermediario entre los programas de aplicación y el hardware del computador. El componente de interfase de sistema hardware/software 204 puede también comprender un manejador de máquina virtual (VMM), un Lenguaje Común Runtime (CLR) o su equivalente funcional, una Máquina Virtual Java (JVM) o su equivalente funcional, o cualquier otro componente de software reemplazando o adicional al sistema operativo que corre en el sistema de computación. El propósito del sistema de interfase hardware/software es proveer un ambiente en donde un usuario pueda ejecutar programas de aplicación. La meta de cualquier sistema de interfase hardware/software es hacer que el sistema de computación se utilice en forma conveniente, y así como también utilizar el hardware del computador de manera eficiente.

[0084] El sistema de interfase hardware/software generalmente se carga en un sistema de computación al inicio y luego maneja todos los programas de aplicación en el sistema de computación. Los programas de aplicación interactúan con el sistemas de interfase hardware/software solicitando servicios a través de un programas de interfase de aplicaciones (API). Algunos programas de aplicación le permiten a un usuario final interactuar con el sistemas de interfase hardware/software a través de una interfase de usuarios tal como un lenguaje de comandos o interfase gráfica de usuario (GUI).

[0085] Un sistema de interfase hardware/software tradicionalmente ejecuta una variedad de servicios para las aplicaciones. En un sistema de interfase hardware/software múlti-tarea donde varios programas pueden estar corriendo a la vez, el sistemas de interfase hardware/software determina que aplicaciones deben correr y en que orden y cuánto tiempo se le permiten a cada aplicación antes de cambiar a otra aplicación.

El sistema de interfase de hardware/software también administra el compartir de la memoria interna entre múltiples aplicaciones, y maneja la entrada y salida, hacia y de, los dispositivos de hardware acoplados, tales como discos duros, impresoras y puertos de marcado. El sistema de interfase de hardware/software también envía mensajes a cada aplicación (y, en algunos casos, al usuario final) concernientes al status de la operación o cualquier error que pueda haber ocurrido. El sistema de interfase de hardware/software también puede descargar la administración de trabajos en cola (por ejemplo, imprimir) de

tal manera que la aplicación que inicia se libera de este trabajo y puede reasumir otros procesos y/o operaciones. En computadores que pueden proveer procesamiento paralelo, un sistema de interfase de hardware/software también puede dividir un programa de tal manera que corra en más de un procesador a la vez.

[0086] Una cubierta de sistema de interfase de hardware/software (que aquí simplemente se llamará "una cubierta") es una interfase de usuario final con un sistema de interfase de hardware/software. (Una cubierta también se puede llamar "un intérprete de comandos" o, en un sistema operativo, una "cubierta de sistema operativo"). Una cubierta es la capa más externa de un sistema de interfase de hardware/software que se puede acceder directamente con un programa de aplicación y/o un usuario final. En contraste con una cubierta, un Kernel es el nivel más interno de un sistema de interfase de hardware/software que interactúa directamente con los componentes de hardware.

[0087] Mientras se prevé que numerosas implementaciones del presente invento son particularmente apropiadas a sistemas de computadores, nada en este documento pretende limitar el invento a estas implementaciones. Por el contrario, como se utiliza acá el termino "sistema de computación" pretende incorporar cualquier dispositivo capaz de almacenar y procesar información y/o capaz de utilizar información almacenada para controlar el comportamiento o ejecución del dispositivo en si, sin importar si estos dispositivos son de naturaleza electrónica, mecánica, lógica, o virtual.

B. ALMACENAJE TRADICIONAL BASADO EN ARCHIVOS

[0088] En la mayoría de sistemas de computación, los "archivos" son unidades almacenables de información que pueden incluir un sistema de interfase de hardware/software como también programas de aplicación, conjuntos de datos, y demás. En todos los sistema de interfase de hardware/software (Windows, Unix, Linux, Mac OS, sistemas de máquinas virtuales, y demás), los archivos son unidades discretas básicas de información (almacenables y accesibles) que se pueden manipular (por ejemplo, datos, programas y demás) por un sistema de interfase de hardware/software. Los grupos de

archivos generalmente se organizan en "folderes". En Microsoft Windows, Macintosh OS, y otros sistema de interfase de hardware/software, un folder es una colección de archivos que se pueden recuperar, mover, y manipular como una unidad de información unica. Estos folderes, a su vez, se organizan en un orden de árbol jerárquico llamado "directorio" (que se discute más detenidamente a continuación). En algunos otros sistema de interfase de hardware/software, tales como DOS,z/OS si la mayoría de sistemas operativos basados enUnix, el termino "directorio" y/o "folder" son intercambiables, y los primeros sistemas de computación Apple (por ejemplo el Apple IIe) utilizan el termino "catálogos" en vez de directorio; sin embargo, como se utiliza acá, todos estos términos son sinónimos que intercambiables y la intención es que incluyen a todos los términos equivalentes que hagan referencia a estructuras de almacenamiento de información jerárquica y sus componentes de folderes y archivos.

[0089] Tradicionalmente, un directorio (a.k.a un directorio de folderes) es una estructura de árbol jerárquica donde los archivos están agrupados en folderes y los folderes a su vez se organizan de acuerdo a su localidad nodal que comprende al árbol del directorio. Por ejemplo, como se muestran en la figura 2A, un sistema de archivos DOS basado en folderes (o "directorio principal") 212 puede comprender una pluralidad de folderes 214, cada uno de los cuales puede tener folderes adicionales (como "sub folderes" de ese folder particular) 216, y cada uno de estos puede tener folderes adicionales 218 ad infinitum. Cada uno de estos folderes puede tener uno o más archivos 220 aunque, en el nivel de sistema de interfase de hardware/software, los archivos individuales en un folder no tienen nada en común más que su localización en el árbol jerárquico. No es de sorprender, que este enfoque para organizar archivos en jerarquías de folderes indirectamente refleja la organización física típica de un medio de almacenaje que se utiliza para guardar estos archivos (por ejemplo discos duros, discos floppy,CD-ROMs,etc).

[0090] Adicional a lo antedicho, cada folder es un contenedor para sus sub folderes y sus archivos -eso es, cada folder tiene sus subfolderes y archivos. Por ejemplo, cuando un folder se elimina a través del sistema de interfase de hardware/software, los sub folderes de ese folder también quedan eliminados (que, en el caso de cada sub folder incluye también sus propios sub folderes respectivamente).

De la misma manera, cada archivo pertenece a un único folder y, aunque un archivo se puede copiar y la copia situar en diferentes folderes, una copia del archivo es una unidad en si separada y discreta que no tiene ninguna conexión con el original (por ejemplo, cambios al archivo original no se reflejan en la copia en el nivel del sistema de interfase de hardware/software).

A este respecto, los archivos y los folderes son característicamente de naturaleza "física" porque los folderes son tratados como contenedores físicos, y los archivos son tratados como elementos físicos separados y discretos dentro de estos contenedores.

II. Plataforma de almacenaje WINFS para organizar, buscar, y compartir datos

[0091] El invento presente, en combinación con los inventos relacionados e incorporados por referencia como se mencionó anteriormente, está dirigido a las plataformas de almacenaje para organizar, buscar, y compartir datos. La plataforma de almacenaje del presente invento extiende y amplía la plataforma de datos más allá de los sistemas de archivos existentes y sistemas de bases de datos que se mencionaron anteriormente, y esta diseñado para almacenar todo tipo de datos, incluyendo una nueva forma de datos llamados items.

C. GLOSARIO

[0092] Tal cómo se utilizan acá y en las reivindicaciones, los siguientes términos tienen el siguiente significado:

- Un "Item" es una unidad almacenable de información accesible al sistema de interfase de hardware/software que, a diferencia de un archivo simple, es un objeto que tiene un conjunto básico de propiedades que se comparte con todos los objetos expuestos al usuario final de una cubierta de un sistema de interfase de

hardware/software. Los items también tienen propiedades y relaciones que son comunes a todos los tipos de items incluyendo rasgos que permiten que se introduzcan nuevas propiedades y relaciones (que se detallan más abajo).

- Un "sistema operativo" (OS) es un programa especial que actúa como intermediario entre los programas de aplicación y el hardware del computador. Un sistema operativo comprende, en la mayoría de casos, una cubierta y un Kernel.
- Un " sistema de interfase de hardware/software" es un software, o una combinación de hardware y software, que sirve de interfase entre los componentes de hardware presentes en un sistema de computación y las aplicaciones que se ejecutan en el sistema de computación. Un sistema de interfase de hardware/software típicamente comprende (y, en algunas implementaciones, consiste únicamente de) un sistema operativo. Un sistema de interfase de hardware/software puede comprender un administrador de máquina virtual (VMM), un Lenguaje Comun Runtime (CLR) o su equivalente funcional, una Máquina Virtual Java (JVM) o su equivalente funcional, u otros componentes de software adicionales o reemplazando el sistema operativo en un sistema de computación. El propósito del sistema de interfase de hardware/software es proveer un ambiente en donde el usuario pueda ejecutar programas de aplicación, que el sistema de computación sea conveniente de usar, así como el uso eficiente del hardware del computador.

D. PERSPECTIVA GENERAL DE LA PLATAFORMA DE ALMACENAJE

[0093] En referencia a la figura 3, la plataforma de almacenaje 300 consta de un un almacén de datos 302 implementado en un motor de base de datos 314. En una implementación, el motor de la base de datos consta de un motor de base de datos relaciónal con extensiones de objetos relacionales. En otra implementación, el motor relaciónal de la base de datos 314 comprende un motor de base de datos relaciónal de servidor de Microsoft SQL. El almacén de datos 302 implementa un modelo de datos 304

que apoya la organización, búsqueda, compartir, sincronización, y seguridad de los datos. Tipos específicos de datos se describen en esquemas, tales como el esquema 340, y la plataforma de almacenaje 300 proveer herramientas 346 para desplegar esos esquemas como también para extender esos esquemas, como se describe más abajo.

[0094] Un mecanismo de cambio de seguimiento 306 se implementa con el almacén de datos 302 para proveer la habilidad de seguir los cambios en el almacén de datos. El almacén de datos 302 también provee capacidades de seguridad 308 y capacidades de promoción/degradación 310, ambos se discuten en más detalle más abajo. El almacén de datos 302 también suministra una serie de interfaces de programación de aplicaciones 312 para exponer las capacidades del almacén de datos 302 y de otros componentes de la plataformas de almacenaje en programas de aplicación (por ejemplo, los programas de aplicación 350a, 350b, y 350c) que utilizan la plataforma de almacenaje. La plataforma de almacenaje del presente invento contiene además una interfase de programación de aplicaciones (API) 322 que le permite a las aplicaciones de programa, tales como los programas de aplicaciones 350a, 350b, y 350c, acceder a todas las capacidades descritas de la plataforma de almacenaje y acceder a los esquemas de datos descritos. Los programas de aplicación pueden utilizar la plataforma de almacenaje API 322 en combinación con otras APIs, tales como OLE DB API 324 y Windows de Microsoft WIN32 API 326.

[0095] La plataforma de almacenaje 300 del presente invento suministra una variedad de servicios 328 a los programas de aplicación, incluyendo un servicio de sincronización 330 que facilita el compartir de datos entre usuarios de sistemas. Por ejemplo el servicio de sincronización 330 puede habilitar la Inter operabilidad con otros almacenes de datos 340 que tienen en el almacén de datos 300 todos el mismo formato, como también el acceso a los almacenes de datos 342 que tienen otros formatos. La plataforma de almacenaje 300 también suministra capacidades de sistema de archivos que le permiten la interoperabilidad del almacén de datos 302 con sistemas de archivos existentes, tales como el sistemas de archivos Windows NTFS 318. En algunas implementaciones, la plataforma de almacenaje 320 puede también suministra programas de aplicaciones con capacidades adicionales que permiten actuar sobre los datos que permiten la interacción con otros sistemas. Estas capacidades se pueden implementar en la forma de servicios adicionales 328, tales como el

servicio Info Agent 334 y el servicio de notificación 332, así como en la forma de otras utilidades 336.

[0096] En algunas implementaciones, la plataforma de almacenaje se implementa en, o forma parte integral, de un sistema de interfase de hardware/software de un sistema de computación. Por ejemplo, y sin ninguna limitación, la plataforma de almacenaje del presente invento se pueden implementar en, o forma una parte integral de, un sistema operativo, un administrador de máquina virtual(VMM), un Lenguaje Común Runtime (CLR) o su equivalente funcional, o una Máquina Virtual Java (JVM) o su equivalente funcional. A través de su base común de almacenamiento, y datos esquematizados, la plataforma de almacenaje del presente invento provee un desarrollo de aplicaciones más eficiente para el consumidor, trabajadores con conocimiento y empresas. Ofrece una superficie de programación rica y extensible que no sólo hace disponibles las capacidades inherentes en el modelo de datos, pero también es compatible y extiende los sistemas de archivos existentes y los métodos de acceso a las bases de datos.

[0097] En la siguiente descripción, y en varias de las figuras, la plataforma de almacenaje 300 del presente invento se puede llamar "WinFs". Sin embargo, el uso de este nombre en referencia a la plataforma de almacenaje es únicamente por conveniencia de descripción y no pretende ser una limitante.

E. EL MODELO DE DATOS

[0098] El almacén de datos 302 de la plataforma de almacenaje 300 del presente invento implementa un modelo de datos que apoya la organización, búsqueda, compartir, sincronización, y seguridad de los datos que residen en el almacén. En el modelo de datos del presente invento, una "item" es un la unidad de almacenaje de información fundamental. El modelo de datos suministra un mecanismo para declarar los items y las extensiones de items para establecer relaciones entre items y para organizar los items en folderes de items y en categorías tal como se describe más abajo.

[0099] el modelo de datos depende de dos mecanismos primitivos, tipos y Relaciones. Tipos son estructuras que suministran un formato que gobierna la forma de una instancia de un Tipo. El formato se expresa en un conjunto ordenado de Propiedades. Una Propiedad es un hombre para un valor o un conjunto de valores de una Tipo dado. Por ejemplo un tipo USPostalAddress puede contener las propiedades Calle, Ciudad,Zip, Estado en donde Calle, Ciudad,Zip y Estado son de tipo String y Zip es de Tipo Int32. Calle puede ser un valor múltiple (por ejemplo un conjunto de valores) que permite que la dirección tenga más de un valor para la propiedad Calle. El sistema define ciertos tipos primitivos que se pueden usar en la construcción de otros tipos -estos incluyen String, Binary, Boolean, Int16, Int32, Int64, Single, Double, Byte, DateTime, Decimal y GUID. Las propiedades de un Tipo se pueden definir utilizando cualquiera de los tipos primitivos o (con algunas restricciones anotadas abajo) cualquiera de los tipos construidos. Por ejemplo el Tipo Localidad se puede definir con las Propiedades Coordinadas y Dirección donde la Propiedad Dirección es del Tipo USPostalAddress como se describió más arriba. Las propiedades pueden ser requeridas o opcionales.

[0100] Las relaciones se pueden declarar y representan un mapeo entre conjuntos de instancias de dos tipos. Por ejemplo puede haber una Relación declarada entre un Tipo Persona y el Tipo Localidad llamados LivesAt que define que personas viven en que sitios. La Relación tiene un nombre, dos puntos finales, un es la fuente y el otro el objetivo. Las Relaciones también pueden tener un conjunto ordenado de propiedades. Ambos puntos finales Fuente y Objetivo tienen un Nombre y un Tipo. Por ejemplo la Relación LivesAt tiene una Fuente llamada Ocupante del Tipo Persona y un Objetivo llamado Ocupante del Tipo Localidad y adicionalmente tiene las propiedades StartDate y EndDate que indican el período de tiempo en que el ocupante vivió en esa dirección. Notése que una persona puede vivir en varias direcciones con el transcurso del tiempo y una vivienda puede tener múltiples ocupantes por lo tanto el sitio más adecuado para poner la información del StartDate y el EndDate es en la relación propiamente dicha.

[0101] Las relaciones definen un mapeo entre instancias que están constreñidas por el tipo dado como uno de los puntos finales. Por ejemplo la relación de LivesAt no puede ser una relación en la cual un automóvil sea el Ocupante porque automóvil no es una Persona.

[0102] El modelo de datos no permite la definición de una relación sub-tipo o super-tipo entre tipos. La relación sub-tipo o super-tipo también conocida como relación BaseTipo se define de tal manera que si el Tipo A es un TipoBase para el TipoB tiene que ser el caso que cada instancia de B tiene que ser también una instancia de A. otra forma de expresar esto es que cada instancia que se conforma con B también se tiene que conformar con A. si, por ejemplo A tiene una propiedad Nombre del tipo String mientras que B tiene una propiedad Edad del Tipo Int16, se deduce que cada instancia debe tiene que tener un Nombre y una Edad. La jerarquía del tipo se puede ver como un árbol con un solo supertipo en la raíz. Las ramas de la raíz proveen el primer nivel de subtipos, las ramas en este nivel proveen un segundo nivel de subtipos y así hasta los subtipos que en sí no tienen más subtipos. El árbol no tiene que tener una profundidad uniforme pero no puede contener ningún ciclo. Un Tipo dado puede tener ceros con muchos subtipos y ceros con un súpertipo. Una instancia dada se puede conformar máximo a un tipo junto con el supertipo de ese tipo. Para ponerlo de otra forma, para una instancia dada en cualquier nivel en el árbol, la instancia se puede conformar máximo a un subtipo en ese nivel. Se dice que un tipo es Abstracto si instancias de ese tipo pueden ser también instancias de un subtipo del tipo.

1. Items

[0103] Un Item es una unidad de información que se puede guardar, y a diferencia de un simple archivo, es un objeto que tiene un conjunto básico de propiedades compatibles con todos los objetos expuestos a un usuario final o programas de aplicaciones por la plataforma de almacenaje. Los Items también tienen propiedades y relaciones que son compatibles con todos los tipos de Items incluyendo rasgos que permiten introducir nuevas propiedades y relaciones, como se discuten más abajo.

[0104] Los Items son objetos para operaciones comunes tales como copiar, eliminar, mover, abrir, escribir, resguardar, recuperar, replicar, y demás. Los Items son unidades que se pueden almacenar y recuperar, y todas las formas de información que se puede

manipular por la plataforma de almacenaje existen como Items, propiedades de Items, o Relaciones entre Items, cada uno de los cuales se discute en más detalle más abajo.

[0105] Se busca que los Items representen unidades fáciles de entender de datos del mundo real como Contactos, Personas, Servicios, Localidades, Documentos (de todas clases), y demás. La figura 5A es un diagrama de bloque que muestra la estructura de los items. El nombre no cualificado de un Items es "localidad". El nombre cualificado del items es "Core. Localidad" que indica que esta estructura de item se define como un tipo específico de Items en el Esquema Core. (El Esquema Core se discuten más abajo.)

[0106] El Items Localidad tiene una pluralidad de propiedades que incluyen EAddresses, MetropolitanRegion, Neighborhood, and PostalAddresses. El tipo específico de propiedad para cada uno se indica inmediatamente siguiendo el nombre de propiedad y separado del nombre de propiedad por dos puntos (":"). A la derecha del nombre de tipo, el número de valores permitido para el tipo de propiedad se indica entre corchetes ("[ ]") donde un[0 ("[0") a la derecha de los dos puntos (":") indican un número no específico y/o ilimitado ("muchos"). Un "1" a la derecha de los dos puntos indica que puede haber máximo un valor. Un cero ("0") a la izquierda de los dos puntos indica que la propiedad es opcional (no puede haber ningún valor). Un "1" a la izquierda de los dos puntos indican que puede haber máximo un valor (la propiedad es requerida). Neighborhood y MetropolitanRegion son ambos del tipo "nvarchar" (o equivalente) que es un tipo de datos predefinido o "tipo simple" (y que aquí se denotan por la ausencia de capitalización). EAddresses y PostalAddresses, sin embargo, son propiedades de tipos definidos o "tipos complejos" (como se denota por la capitalización) de tipos EAddress y PostalAddress respectivamente. Un tipo complejos es un tipo que se deriva de uno con más tipos simples de datos de otros tipos complejos. Los tipos complejos para las propiedades de un Items también constituyen un "elemento anidado" desde que los detalles del tipo complejos están anidados dentro del Items inmediato para definir sus propiedades, y la información perteneciente a estos tipos complejos se mantiene con el Item que tiene estas propiedades (dentro de los límites del Items, como se discuten más tarde). Estos conceptos de tipificación son bien conocidos y apreciados por aquellos diestros en el arte.

[0107] La figura 5B es un diagrama de bloque que muestra los tipos de propiedades complejas EAddress y PostalAddress. La propiedad del tipo PostalAddress define que un Items de tipo de propiedad PostalAddress puede tener un cero o un valor de City, cero o un valor CountruCode, cero o un valor MailStop, y cualquier número (cero a muchas) de TiposPostalAddress, y así. De esta manera la forma de los datos para una propiedad particular de un Item se define. El tipo propiedad EAddtress se define en forma similar. Aúnque en esta aplicación se utilizan en forma ocasional, otra manera de representar los tipos complejos en el Item Localidad es dibujar el Item con las propiedades individuales para cada tipo complejo que se enumere. La figura 5C es un diagrama de bloque que muestra el Item Localidad donde sus tipos complejos están descritos con mas detalle. Sin embargo, se debe entender que esta representación alternativa del Item Localidad en esta figura 5C es exactamente para el mismo Item ilustrado en la figura 5A. La plataforma de almacenaje del presente invento también permite la sub tipificación donde un tipo de propiedad puede ser un subtipo de otro (donde un tipo de propiedad hereda las propiedades de otro, tipo de propiedad paterno).

[0108] Similar pero distinto de las propiedades y sus tipos de propiedad, los Items inherentemente representan su propio Tipo de Items que puede también ser el sujeto de un subtipo. En otras palabras, la plataforma de almacenaje en varias implementación es del presente invento permite que un Items pueda ser un subtipo de otro Items (donde el Item hereda las propiedades de otro Item paterno). Más aún, para varias implementación del presente invento, cada item es un subtipo del "Item" tipo de Item que es el primer tipo de item fundamental encontrado en el Esquema Base. (El Esquemas Base se discutirá en más detalle más adelante) la figura 6A muestra un Item, el Item Localidad en esta Instancia, como siendo uso tipo del tipo de Item del Item encontrado en el Esquema Base. En este dibujo, la flecha indica el Item Localidad (como todos los otros items) es un subtipo del tipo de Item del Item. El tipo de Item del Item, como el Item fundamental de donde todos los demás Items se derivan, tiene un número importante de propiedades tales como ItemId y varios timestamps, y por lo tanto define las propiedades estándar de todos los Items en el sistema operativo. En la presente figura, estas propiedades del tipo de Item del Item se hereda en por Localidad y por lo tanto se convierten en propiedades de Localidad.

[0109] Otra manera de representar las propiedades en el Item Localidad heredadas del tipo de Item del Item es dibujar Localidad con las propiedades individuales para cada tipo de propiedad del Item paterno descrito. La figura 6B es un diagrama de bloque que muestra el Item Localidad donde sus tipos heredados se describen adicionalmente a sus propiedades inmediatas. Se debe notar y entender que este Item es el mismo Item que se muestra en la figura 5A, aunque en la presente figura Localidad se ilustra con todas sus propiedades, ambas inmediatas -que se muestran en ésta figura y en las 5A- y que se heredan y que se muestran en ésta figura pero no en la figura 5A (mientras que en la figura 5A estas propiedades se muestran con una flecha se hace referencia a que el Item Localidad es un subtipo del tipo de Item del Item.

[0110] los Items son objetos en su propio derecho; por lo tanto, si usted elimina un Item, todas las propiedades inmediatas y heredadas del Item también quedaron eliminadas.

En forma similar, cuando se recupera un Items, lo que se recupera es el Items y todas su sus propiedades inmediatas y heredadas (que incluyen la información perteneciente a sus tipos de propiedades complejas). Algunas implementaciones del presente invento le permiten a uno solicitará un uso conjunto de propiedades cuando se recupera un Item específico; sin embargo, la implementación por defecto es proveer al Item con todos sus propiedades inmediatas y heredadas cuando se recupera. Aún más, las propiedades del Item se pueden extender añadiendo nuevas propiedades a las propiedades existentes para el tipo de Item. Estas "extensiones" son de ahí en adelante propiedades bona fide del Item hizo tipos de ese tipo de Item pueden incluir automáticamente las propiedades de la extensión.

[0111] El "límite" del Item se representa con sus propiedades (que incluyen propiedades complejas, extensiones, y demás). Y el límite del Item también representa el límite de una operación ejecutada en un Item como copiar, eliminar, mover, crear y demás. Por ejemplo en varias implementaciones del presente invento, cuando un Item se copia, todo lo que está dentro del límite del Item también se copia. Para cada Item, el límite comprende los siguientes:

- El Tipo de Item del Item y si el Item es un subtipo de otro Item (como es el caso en varias implementaciones del este invento donde todos los Items se derivan de un solo Item y el tipo de Item en el Esquema Base), cualquiera

subtipo de información aplicable (eso es, información pertinente al tipo de Item Packard). Si el Item original que se está copiando es un subtipo de otro item 1, la copia puede también ser un subtipo de ese mismo Item.

- Las propiedades de tipo complejo y las extensiones, si las hay. Si el Item tiene propiedades del tipo complejo (nativas o extendidas), la copia también puede tener el mismo tipo complejo.
- El record del Item en "relaciones de pertenencia", eso es, la lista propia del Item sobre los otros Items (el "Item objeto") que le pertenecen al presente Item (el "Item poseedor"). Esto es particularmente relevante en relación a los folderes de Item, que se discuten en más detalle más adelante, y la regla que se menciona más abajo de que todos los Items deben pertenecer a por lo menos un folderes de Item. Aún más, en cuanto a items empotrados -que se discuten en detalle más adelante- un item empotrado es considerado como parte del Item en el cual estaba empotrado para operaciones tales como copia, y demás.

2. Identificación de los Items

[0112] Los items se identifican en forma única dentro del espacio global de los items con un ItemID. El tipo Items.Base del fin de un campo ItemID del tipo GUID almacena la identidad de los Items. Un Item debe tener exactamente una identidad en el almacén de datos 302.

[0113] La referencia de un item es una estructura de datos que contiene información para localizar e identificar un Item. En el modelo de datos, un tipo abstracto se define con un nombre de ItemReferencia de donde todos los tipos de referencia de los items se derivan. El tipo ItemReferencia define un método virtual llamado Resolución. El método de Resolución resuelve el ItemReferencia y retorna un Item. Sobre este método priman los subtipos concretos del ItemReferencia, que implementan una función que recupera un Item dada su referencia. El método Resolución se invoca como parte de la plataforma de almacenaje API 322.

[0114] El ItemIDReferencia es un subtipo del ItemReferencia. Define un Localizador y un campo de ItemID. El campo Localizador nombra (por ejemplo, identifican) el dominio de un item. Se procesa a través de una resolución de localización que puede resolver el valor del Localizador para el dominio de un item. El campo ItemID es del tipo ItemID.

[0115] El ItemPathReference es una especialización del ItemReferencia que define el Localizador y un campo de Path. El campo Localizador identifica el dominio de un item. Se procesa por un método de resolución de localización que puede resolver el valor del Localizador para el dominio de un Item. El campo Path contiene un path (relativo) en el namespace de la plataforma de almacenaje enraizado en el dominio del item dado por el Localizador.

[0116] Este tipo de referencias no se puede usar en una operación dada. La réferencias generalmente se tienen que resolver a través de un proceso de resolución de Path. El método Resolución de la plataforma de almacenaje API 322 provee esta funcionalidad.

[0117] Las formas de referencias mencionados anteriormente se representan a través de una jerarquía de tipos de referencias que se muestra en la figura 11. Tipos de referencia adicionales que se heredan de estos tipos se pueden definir en el esquema. Se pueden utilizar en una relación declarada como el tipo del campo objetivo.

3. Folderes de items y categorías

[0118] Cómo se menciona más adelante y en más detalle, los grupos de Item se pueden organizar en Items llamados Folderes de Item (que no se deben confundir con los folderes de archivos). A diferencia de los sistemas de archivos, sin embargo, un Item puede pertenecer a más de un Folder de Item, de tal manera que cuando un Item se accede en un Fólder de Item y se revisa, este Item revisado se puede acceder a través de otro fólder de Item. En esencia, aunque el acceso a un Item puede ocurrir de distintos Folderes de Items, lo que realmente está siendo localizado es de hecho el mismo Item. Sin embargo, un Folder de Item no necesariamente es dueño de todos los Items miembros, o simplemente pueden ser copropietario en conjunto con otros folderes, de tal manera que la eliminación de un

Folder de Item no necesariamente resulta en la eliminación del Item. Sin embargo, en varias implementaciones del presente invento, un Item debe pertenecer a por lo menos un Folder de Item de tal manera que si el Folder de Item para un Item particular se eliminan entonces, en algunas implementaciones, el Item es automáticamente eliminado o, en una implementación alternativa, el Item automáticamente se convierte en un miembro del Folder de Item por defecto (por ejemplo, un Folder de Item "Papelera" es conceptualmente similar al folder llamado de igual manera y que se utiliza en varios sistemas basados en archivos y folderes).

[0119] Cómo se discute en más detalle más adelante, los Items también pueden pertenecer a Categorías basadas en características que se describen comúnmente tales como (a) un Tipo de Item (o Tipos), (b) una propiedad inmediata o heredada específica (o propiedades), o (c) un valor específico (o valores) correspondientes a una propiedad de un Item. Por ejemplo, un Item comprende unas propiedades específicas para la información de un contacto personal que pueden pertenecer automáticamente a la ContactCategory, y cualquier Item que tiene propiedades de información de contacto igualmente pertenecería a esta Categorías. De la misma manera cualquier Item que tenga una propiedad de localización con un valor de "Ciudad de Nueva York" pueden automáticamente pertenecer a la NewYorkCityCategory.

[0120] Las Categorías son conceptualmente diferentes a los Folderes de Items en cuanto a que un Folder de Item puede tener Items que no están interrelacionados (por ejemplo, sin una característica comúnmente descrita), cada Item en una Categorías tiene un tipo, propiedad a, o valor común ("commonality") que se describe es para esa Categorías, y es esta commonality que forma la base para su relación con los demás Items en la Categoría. Aún más, mientras que la membresía de un Item con un Folder en particular no es compulsiva en ningún aspecto de ese Item con en ciertas implementaciones todos los Items que tienen una commonality categóricamente relacionadas con una Categoría pueden convertirse automáticamente en miembros de la Categoría en el nivel del sistema de interfase de hardware/software. Conceptualmente, las Categorías también se pueden ver cómo Folderes de Items virtuales cuya membresía está basada en en el resultado de una consulta específica (tal como el contexto de una base de datos), y los Items que satisfacen

53

las condiciones de ésta consulta (definida por las commonalities de la Categoría) comprenderían entonces la membresía a la Categoría.

[0121] La figura 4 muestra las relaciones estructurales entre los Items, Folderes de Items, y las Categorías. Una pluralidad de Items 400, 404, 406, 408, 410, 412, 414, 416, 418, y 420 son miembros de varios Folderes de Items 422, 424, 426, 428, y 430. Algunos Items pueden pertenecer a más de un Folder de Items, por ejemplo, Item 402 pertenece al Folder de Item 422 y 424. Algunos Items, por ejemplo, Items 402, 404, 406, 408, 410, y 412 son también miembros de una o más Categorías 432, 434, y 436, mientras que en otros momentos, por ejemplo los Items 414, 416, 418, y 420 pueden no pertenecer a ninguna Categorías (aunque esto es improbable en ciertas implementaciones donde la posesión de cualquier propiedad automáticamente implica la membresía en una Categorías, y por lo tanto un Item tendría que no tener ningún rasgo para no pertenecer a ninguna categorías en tal implementación). A diferencia de la estructura jerárquica de los folderes, tanto Categorías como Folderes de Items tienen estructuras que se parecen más a las gráficas dirigidas que se muestran. En cualquier caso, los Items, Folderes de Items, y Categorías son todos Items (aunque pueden ser distintos Tipos de Items).

[0122] A diferencia de los archivos, folderes, y directorios, los Items, Folderes de Items y las Categorías del presente invento no son característicamente de naturaleza "física" porque no tienen el equivalente conceptual de contenedores físicos, y por lo tanto los Items pueden existir en más de una localidad. La habilidad de los Items para existir en más de una localidad de Folderes de Items y estar organizados en Categorías le da a la manipulación y manejo de datos una capacidad enriquecida y mejorada al igual que a las estructuras de almacenaje en el nivel de interfase de hardware/software más allá de lo que hoy se encuentra disponible en el arte.

4. Esquemas

a) Esquema Base

[0123] Para fijar un fundamento universal para la creación y uso de los Items, varias implementaciones de la plataforma de almacenaje del presente invento comprenden un Esquema Base que establece un marco conceptual para crear y organizar Items y propiedades. El Esquema Base define cierto tipo especial de Items y propiedades, y los rasgos de estos tipos fundamentales especiales de donde los subtipos se pueden derivar. El uso de estos Esquemas Base le permite a los programadores distinguir conceptualmente los Items (y sus respectivos tipos) de las propiedades (y sus respectivos tipos). Es más, el Esquema Base sienta las bases de las propiedades que todos los Items pueden tener dado que todos los Items (y sus correspondientes Tipos de Items) se derivan de este Item fundamental en el Esquema Base (y su correspondiente Tipo de Item).

[0124] Como se muestra en la figura 7, respecto a ciertas implementaciones del presente invento, el Esquema Base define tres tipos de alto nivel: Item, Extensión, y PropiedadBase. Como se muestra, el tipo de Item se define por la propiedad de este tipo de Item "Item" fundamental. A diferencia, la propiedad de alto nivel "PropiedadBase" no tiene propiedades predefinidas y es simplemente un ancla de donde todos los demás tipos de propiedades se derivan y a través del cual todos los tipos de propiedades derivadas se interrelacionan (al ser comúnmente derivadas de un mismo tipo de propiedad). El tipo de propiedad Extensión define que extensión de Item se extiende como también identificación para distinguir una extensión en la medida que un Item puede tener múltiples extensiones.

[0125] Un ItemFolder es un subtipo del Item tipo de Item, adicional a las propiedades heredadas del Item, tiene una Relación para establecer vínculos con sus miembros (si los hay), donde ambas IdentityKey y Propiedades son subtipos de PropiedadBase. CategoryRef, a su vez, es un subtipo de IdentityKey.

b) Esquema del Núcleo

[0126] Varias implementaciones de la plataforma de almacenaje del presente invento comprenden además un Esquema Base que suministra un marco conceptual para las estructuras de alto nivel de los tipos de Items. La figura 8A es un diagrama de bloques que muestra los items en un Esquema Base, y la figura 8B es un diagrama de bloque que muestra los tipos de propiedad en el Esquema Base. La distinción que se hace entre archivos con distintos extensiones ([0. com, [0.exe, [0.bat, [0.sys, etc.) y demás tipos de criterio en los sistemas basados en archivos y folderes es análogo a la función del Esquema Base. En el sistema de interfase de hardware/software basado en Items, el Esquema Base define una serie de tipos de Item base que, directamente (por el tipo de Item) o indirectamente (por el subtipo de Item), caracteriza a todos los Items en uno o más tipos de Item Esquema Base que el sistema de interfase de hardware/software basado en items pueden entender y directamente procesar en una forma predeterminada y predecible. Los tipos de Item predefinidos reflejan los Items más comunes en el sistema de interfase de hardware/software basado en Items y por lo tanto el nivel de eficiencia aumenta con el sistema de interfase de hardware/software basado en Items que entiende estos tipos de Items predefinidos que comprenden el Esquema Base.

[0127] En ciertas implementaciones el Esquema Base no es extensible -esto es, ningún tipo tradicional de Items puede ser directamente subtipificado directamente del tipo de Item en el Esquema Base excepto por el tipo de Items específico predefinido y derivado que es parte del Esquema Base. Al prevenir las extensiones en el Esquema Base (esto es, prevenir la división de nuevos Items en el Esquema Base), la plataforma de almacenaje requiere el uso de tipos de Items del Esquema Base desde que cada tipo de Item subsiguiente es necesariamente un subtipo del tipo de Item del Esquema Base. Esta estructura permiten un grado de flexibilidad razonable al definir tipos de Items adicionales mientras se mantienen los beneficios de tener un conjunto de tipos de Items predefinido.

[0128] En varias implementaciones del presente invento, y en referencia a la figura 8A, los tipos de Items específicos permitidos por el Esquema Base pueden incluir uno o más de los siguientes:

- Categorías: items de este tipo de Items (y subtipos derivados de ahí) representan una Categoría válida en el sistema de interfase de hardware/software basado en Items.
- Productos: Items que son cosas de valor identificables.
- Dispositivos: Items que tienen una estructura lógica y apoyan las capacidades de procesamiento de información.
- Documentos: Items con un contenido que no lo interpreta el sistema de interfase de hardware/software basado en Items pero en cambio es interpretado por programas de aplicación correspondientes al tipo de documento.
- Eventos: Items que gravan ciertas ocurrencias en el ambiente.
- Localización: items que representan un lugar físico (por ejemplo: sitios geográficos).
- Mensajes: Items de comunicación entre dos o más principales (definidos más adelante).
- Principales: Items que tienen por lo menos una identidad probable definitiva aparte de un ItemID (por ejemplo, la identificación de una persona, organización, grupo, familia, autoridad, servicio, etc.).
- Declaraciones: Items que tienen información especial en cuanto al ambiente que incluyen, sin limitación, políticas, suscripciones, credenciales, y demás.

[0129] De la misma manera, en referencia a la figura 8B, los tipos de propiedad específica que el Esquema Base permite pueden incluir uno o más de los siguientes:

Certificados (que se derivan de un tipo de PropertyBase en el Esquema Base)

Principal Identity Keys (derivado del tipo de IdentityKey en el Esquema Base)

Postal Address (derivado de Property type en el Esquema Base)

Rich Text (derivado de Property type en el Esquema Base)

EAddress (derivado de Property type en el Esquema Base)

IdentitySecurityPackage (derivado del tipo de Relación en el Esquema Base)

RoleOccupancy (derivado del tipo de relación en el Esquema Base)

BasicPresence (derivado del tipo de relación en el Esquema Base)

Estos Items y Propiedades son descritos adicionalmente por sus propiedades respectivas que se muestran en la figura 8A y 8B.

5. Relaciones

[0130] Las Relaciones son relaciones binarias donde un Item se designa como fuente y otro Item se designa como objetivo. El Item fuente y el Item objetivo están relacionados por una Relación. El Item fuente generalmente controla el tiempo de vida de la relación. Esto es, que cuando el Item fuente se elimina, la relación entre los Items también se elimina.

[0131] Las relaciones se clasifican en: relaciones de Referencia y de Contención. Las relaciones de contención controlan el tiempo de vida de los Items objetivos como mientras que las relaciones de referencia no tienen ningún tipo de semántica que se refiera al tiempo de vida. La figura 12 muestra la manera en que las relaciones se clasifican.

[0132] Las relaciones de Contención se clasifican adicionalmente en relaciones de Pertenencia y de Incrustación. Cuando todos las relaciones de pertenencia de un Item se eliminan, el Item se elimina. Una relación de pertenencia controla el tiempo de vida del objetivo a través de un mecanismo referencial de conteo. Las relaciones de incrustación permiten el modelado de Items compuestos y se pueden ver como una relación de pertenencia exclusiva. Un Item puede ser el objetivo de una o más relaciones de pertenencia; pero un Item sólo puede ser objetivo de una relación de incrustación. Un Item que es el objetivo de una relación de incrustación no puede ser el objetivo de ninguna otra relación de pertenencia o incrustación.

[0133] Las relaciones de Referencia no controlan el tiempo de vida de un Item objetivo. Ellos pueden estar indefinidos -el Item objetivo puede no existir. Las relaciones de Referencia se pueden usar para moldear referencias a Items en cualquier sitio del espacio global nombrado de los Items (por ejemplo, incluyendo almacenes de datos remotos).

[0134] Recuperar un Item automáticamente no recupera sus relaciones. Las aplicaciones pueden solicitar explícitamente la relación de un Item. Adicionalmente, modificar una relación no necesariamente modifica el item fuente u objetivo; en forma similar, añadir relaciones no afecta el Item fuente/objetivo.

a) Declaración de Relaciones

[0135] Los tipos de relaciones explícitas se definen con los siguientes elementos:

- En el atributo Nombre se define un nombre específico para la relación.
- En tipo de relación se utiliza uno de los siguientes: Pertenencia, Incrustación, Referencia. Esto se especifica en el atributo Tipo.
- Puntos finales fuente y objetivo. Cada punto final especifica un nombre y el tipo de Item que referencia.
- El campo de punto final fuente generalmente es del tipo ItemID (no declarado) y desde hace referencia a un Item en el mismo almacén de datos donde está la instancia de la relación.
- Para relaciones de Pertenencia y Incrustación, el campo de punto final debe ser del tipo ItemIDReference y debe hacer referencia a un Item en el mismo almacén de datos donde está la instancia de la relación.
- Para las relaciones de Referencia el punto final objetivo puede ser del tipo ItemReference y puede hacer referencia a un Item en una plataforma de almacén de datos distinto a donde está la instancia de la relación.
- Opcionalmente uno o más campos del tipo escalar o PropertyBase puede declararse. Estos campos pueden contener datos asociados a la relación.
- Las instancias de las Relaciones se guardan en una tabla global de relaciones.
- Cada instancia de relación se identifica en forma única con la combinación (ItemID,ID de relación). El ID de una relación es único dentro de un ItemID para todas las relaciones cuya fuente es un Item dado sin importar su tipo.

[0136] El Item fuente es el propietario de la relación. Mientras que un Item designado como propietario controla el tiempo de vida de la relación, las relaciones y está separada del Item que relaciona. La plataforma de almacenaje API 322 habilita un mecanismo para exponer las relaciones asociadas con un Item.

[0137] Este es un ejemplo de una declaración de relación:

```
<Relationship Name="Employment" BaseType="Reference" >
<Source Name="Employee" ItemType="Contact.Person"/>
<Target Name="Employer" ItemType="Contact.Organization"
  ReferenceType="ItemIDReference" />
<Property Name="StartDate" Type="the storage platformTypes.DateTime" />
<Property Name="EndDate" Type="the storage platformTypes.DateTime" />
<Property Name="Office" Type="the storage platformTypes.DateTime" />
</Relationship>
```

[0138] Este es un ejemplo de una relación de Referencia. La relación no se puede crear si el Item de la persona a la que referencia con la referencia fuente no existe. También, si el Item de la persona se elimina, la instancia de la relación entre la persona y la organización se elimina. Sin embargo sí el Item Organización se elimina, la relación no se elimina y queda en espera.

b) Relaciones de Pertenencia

[0139] Las relaciones de Pertenencia moldean el manejo contable de referencias basadas en el tiempo de vida de los Items objetivos.

[0140] Un Item puede ser el punto final fuente para una relación cero o más con un Item. Un Item que no es un Item incrustado puede ser objetivo en una o más relaciones de pertenencia.

[0141] El tipo de referencia del punto final objetivo debe ser un ItemIDReference y debe hacer referencia a un Item en el mismo almacén donde se encuentra la relación.

[0142] Las relaciones de Pertenencia administran el manejo del tiempo de vida del punto final objetivo. La creación de una relación de pertenencia y del Item que es su objetivo es una operación atómica. Se pueden crear instancias de relaciones de pertenencia adicionales que apuntan al mismo Item. Cuando la última relación de pertenencia con un Item dado como punto final objetivo se elimina el Item objetivo también se elimina.

[0143] Los tipos de Items de punto final que se especifican en la declaración de relación generalmente se verán forzados cuando una instancia de la relación se crea. Los tipos de Items de punto final no se pueden cambiar luego de que la relación se haya establecido.

[0144] Las relaciones de Pertenencia juegan un papel importante en la formación del namespace del Item. Ellas contienen la propiedad "Name" que definen el nombre del Item objetivo relativo al Item fuente. El nombre relativo es único para todos los objetivos de relaciones de pertenencia derivados de un Item dado. La lista ordenada de estos nombres relativos comenzando por el Item raíz en un Item forma el nombre completo para el Item.

[0145] Las relaciones de pertenencia forman un gráfico dirigido acíclico (DAG). Cuando una relación de pertenencia se crea el sistema se asegura que un ciclo no se cree, asegurando de esta manera que el namespace forma un DAG.

[0146] Mientras las relaciones de pertenencia controlan el tiempo de vida de los Items objeto, no controla la consistencia operacional del item de punto final objetivo. El Item objeto es independiente operacionalmente del Item a quien le pertenece a través de la relación de pertenencia. Copiar, Mover, Resguardar y otras operaciones en el Item que es fuente de una relación de pertenencia no afectan al Item que es el objetivo de la misma relación -por ejemplo, resguardar un Folder de Item automáticamente no resguarda todos los Items en el folder (objetos de la relación Foldermember).

[0147] bien El siguiente es un ejemplo de una relación de pertenencia:

```
<Relationship Name="FolderMembers" BaseType="Holding" >
<Source Name="Folder" ItemType="Base.Folder"/>
<Target Name="Item" ItemType="Base.Item"
  ReferenceType="ItemIDReference" />
</Relationship>
```

[0148] Las relaciones FolderMembers permiten que el concepto de Folder sea un término genérico para una colección de Items.

c) Relaciones incrustadas

[0149] El modelo de relaciones incrustadas moldea el concepto de control exclusivo del tiempo de vida de un Item objeto. Permiten el concepto de Items compuestos.

[0150] La creación de una relación incrustada y el Item al que apunta es una operación atómica. Una Item puede ser la fuente de una relación cero o más incrustada. Sin embargo, un Item puede ser el objeto de una y sólo una relación de incrustación. Un Item que es objeto de una relación incrustada no puede ser el objetivo de una relación de pertenencia.

[0151] El tipo de referencia objeto de punto final debe ser un ItemIDReference y debe hacer referencia a un Item en el mismo almacén de datos en donde se encuentra la instancia de la relación.

[0152] Los tipos de los Items de punto final especificados en la declaración de relación generalmente serán forzados cuando se crea una instancia de la relación. Los tipos de Items de punto final no se pueden cambiar luego de que la relación se haya establecido.

[0153] Las relaciones incrustadas controlan la consistencia operacional de los puntos finales objeto. Por ejemplo la operación de serializar un Item puede incluir la serialización de todas las relaciones incrustadas que es fuente de ese Item como también de todos los objetos; copiar un Item también copia todos los Items incrustados.

[0154] El siguiente es un ejemplo de una declaración:

```
<Relationship Name="ArchiveMembers" BaseType="Embedding" >
<Source Name="Archive" ItemType="Zip.Archive" />
<Target Name="Member" ItemType="Base.Item "
  ReferenceType="ItemIDReference" />
<Property Name="ZipSize" Type="the storage platformTypes.bigint" />
<Property Name="SizeReduction" Type="the storage platformTypes.float" />
</Relationship>
```

d) Las relaciones de referencia

[0155] Las relaciones de referencia no controlan el tiempo de vida del Item que referencian. Aún más, las relaciones de referencia no garantizan la existencia del objeto, ni tampoco garantizan el tipo del objeto como se especifica en la declaración de la relación. Esto significa que la relación de referencia puede estar sin fundamento. También, la relación de referencia puede hacer referencia a un Item en otro almacén de datos. La relación de Referencia se puede como un concepto parecido a los vínculos en una página web.
[0156] Un ejemplo de una declaración de relación de referencia es el siguiente:

```
<Relationship Name="DocumentAuthor" BaseType="Reference" >
<Sourc  ItemType="Document" ItemType="Base.Document"/>
<Target ItemType="Author" ItemType="Base.Author"
   ReferenceType="ItemIDReference" />
<Property Type="Role" Type="Core.CategoryRef" />
<Property Type="DisplayName" Type="the storage platformTypes.nvarchar(256)" />
</Relationship>
```

[0157] Cualquier tipo de referencia es permitido en el puntos finales objeto. Los Items que participan en una relación de referencia pueden ser de cualquier tipo de Item.
[0158] Las relaciones de Referencia se usan para moldear las relaciones de administración que no controlan el tiempo de vida entre los Items. Como la existencia del objeto no es necesaria, la relación de referencia es apropiada para moldear relaciónes que se acoplan en forma suelta. Las relaciones de referencia se podemos dar para apuntalar a Items en otros almacenes de datos incluyendo almacenes en otros computadores.

e) Reglas y Constreñimientos

[0159] Las siguientes reglas adicionales y constreñimiento le aplican a las relaciones:

Un Item debe ser objeto de (exactamente una relación de incrustación) o (una o más relaciones de pertenencia). Una decepción es el Item raíz. Un Item puede ser objeto de cero o más relaciones de referencia.

Un Item que es objeto de una relación de incrustación no puede ser fuente de una relación de pertenencia. Puede ser fuente de una relación de referencia.

Un Item no puede ser fuente de una relación de pertenencia si se promociona del archivo. Puede ser fuente de una relación de incrustación o relación de referencia.

Un Item que es promocionado de un archivo no puede ser objeto de una relación de incrustación.

f) Ordenamiento de las Relaciones

[0160] En por lo menos una implementación, la plataforma de almacenaje del presente invento permiten el ordenamiento de las relaciones. El ordenamiento se logra a través del nombre de propiedad "Orden" en la definición base de la relación. No hay ningún constreñimiento de exclusividad en el campo Orden. El orden de la relación con el mismo valor de propiedad "orden" no está garantizado, sin embargo sí está garantizado que se pueden ordenar después de relaciones con un valor inferior de "orden" y antes que relaciones con un valor del campo superior a "orden".

[0161] Las aplicaciones pueden obtener las relaciones en el orden por defecto al ordenar las combinaciones (SourceItemID, RelationshipID, Orden). Todas las instancias de relación que vienen de un Item dado están ordenadas como una colección única sin importar el tipo relaciones en la colección. Estos sin embargo garantiza que todas las relaciones de un mismo tipo (por ejemplo, FolderMember) son un subconjunto ordenado de la colección de relaciones para un Item dado.

[0162] El almacén de datos API 312 implementa una serie de operaciones para manipular las relaciones que permiten el ordenamiento de las relaciones. Se introducen los siguientes términos para ayudar a explicar las operaciones:

- *RelFirst* es la primera relación en la colección ordenada con valor de orden *OrdFirst*;
- *RelLast* es la última relación en la colección ordenada con valor de orden *OrdLast*;
- *RelX* es una relación dada en la colección con valor de orden *OrdX*;
- *RelPrev* es la relación más cercana en la colección a *RelX* con valor de orden *OrdPrev* inferior a *OrdX*; y
- *RelNext* es la relación más cercana en la colección a RelX con valor de orden OrdNext mayor que *OrdX*.

[0163] Las operaciones incluyen pero no se limitan a:

- *InsertBeforeFirst(SourceItemID, Relationship)* inserta la relación como la primera relación en la colección. El valor de la propiedad "Orden" de la nueva relación puede ser entonces inferior a OrdFirst.
- *InsertAfterLast(SourceItemID, Relationship)* inserta la relación como la última relación en la colección. El valor de la propiedad "Orden" de la nueva relación puede ser entonces mayor a OrdLast.
- *InsertAt(SourceItemID, ord, Relationship)* inserta una relación con el valor especificado para la propiedad "Orden".
- *InsertBefore(SourceItemID, ord, Relationship)* inserta la relación antes de la relación con el valor de orden dado. A la nueva relación se le puede asignar un valor de "Orden" que está entre Ordprev y Ord, no inclusive.
- *InsertAfter(SourceItemID, ord, Relationship)* inserta la relación luego de la relación con el valor de orden dado. A la nueva relación se le puede asignar un valor de "Orden" que está entre Ord y OrdNext, no inclusive.
- *MoveBefore(SourceItemID, ord, RelationshipID)* mueve la relación con el ID de relación dado antes de la relación con el valor de "Orden" dado. A la relación se le puede asignar un nuevo valor de "Orden" que está entre OrdPrev y ord, no inclusive.
- *MoveAfter(SourceItemID, ord, RelationshipID)* mueve la relación con un ID de relación dado luego de la relación con el valor de "Orden" especificado. A la

relación se le puede asignar un nuevo valor de orden que está entre ord y OrdNext, no inclusive.

[0164] Como se mencionó previamente, todos los Item debe ser miembros de un Folder de Item. En términos de las Relaciones, todo Item debe tener una relación con un Folder de Item. En varias implementaciones del presente invento, ciertas relaciones se representan con Relaciones que existen entre Items.

[0165] Como se implementa en varios ejemplos del presente invento, una Relación provee una relación directa binaria que se "extiende" en un Item (la fuente) a otro Item (del objeto). Una Relación le pertenece al Item fuente (el item que la extendió), y así la Relación se elimina si la fuente se elimina (por ejemplo, la Relación se elimina cuando el Item fuente se elimina). Es más, en ciertos casos, una Relación puede compartir la pertenencia de (o ser parte dueña) de él Item objeto, y tal pertenencia se puede reflejar en la propiedad IsOwned (o su equivalente) de la Relación (como se muestra en la figura 7 para el tipo de propiedad Relación). En estas implementaciones, la creación de una nueva relación IsOwned automáticamente incrementa el conteo referencial en el Item objeto, y la eliminación del tal Relación puede disminuir el conteo referencial en el Item objeto. Para estas implementaciones específicas, los Items continúan existiendo si tienen un conteo referencial mayor que cero, y automáticamente se eliminan cuando el conteo llega a cero. De nuevo, en un Folder de Item hay Item que tiene (o es capaz de tener) un conjunto de Relaciones con otros Items, estos otros Items comprenden la membresía del Folder Item. Otras implementaciones de Relaciones son posibles y se anticipan con el presente invento para lograr la funcionalidad que aquí se describe.

[0166] Sin importar que implementación se aplique, una Relación es una conexión, que se puede seleccionar, de un objeto con otro. La habilidad de un Item de pertenecer a más de un Folder de Item, como también a una o más Categorías, y si estos Items, Folderes, y Categorías son públicos o privados, se determina por el significado dado a la existencia (o ausencia) en una estructura basada en Items. Estas Relaciones lógicas son los significados asignados a un conjunto de Relaciones, sin importar una implementación física, que se emplean específicamente para lograr la funcionalidad acá descrita. Las Relaciones Lógicas

77

se establecen entre un Item y su Folder de Item o Categoría (y viceversa) porque, en esencia, los Folder de Item y las Categorías son cada uno Items de tipo especial. En consecuencia, se puede actuar sobre los Folder de Item y las Categorías de la misma manera que se puede hacer con cualquier otro Item, -copiar, añadir a un e-mail, incrustar en un documento, y demás sin limitación-. Los Folderes de Item y Categorías se pueden serializar y deserializar (importados y exportados) utilizando el mismo mecanismo que se utiliza con otros Items. (Por ejemplo, en XLM todos los items pueden tener un formato seríalizado, y este formato aplica igualmente a Folderes de Items, Categorías, y Items.)

[0167] Las Relaciones anteriormente mencionadas, que representan las relaciones entre un Item y un Folder de Item pueden extenderse lógicamente de un Item a su Folder de Item, del Folder de Item al Item, o ambos. Una Relación que se extienden lógicamente de un Item a su Folder de Item denota que el Folder de Item es público para ese Item y comparte su información de membresía con el Item; a la inversa, la falta de una Relación lógica de un Item a su Folder de Item denota que el Folder de Item es privado al Item y no comparte su información de membresía con el Item. En forma similar, una Relación que se extiende lógicamente de un Folder de Item aún Item denota que el Item es público y se puede compartir con ese Folder de Item, mientras que la ausencia de una Relación lógica de un Folder de Item al Item denota que el Item es privado y no se puede compartir. En consecuencia, cuando un Folder de Item se exporta a otro sistema, es el Item "público" que se comparte en el nuevo contexto, y cuando un Item busca su Folder de Item para otros Items que se pueden compartir, es el Folder de Item "público" que le da al Item la información concerniente al Item que le pertenece.

[0168] La figura 9 es un diagrama de bloque que muestra un Folder de Item (que de nuevo, es un Item en sí), sus Items, y las Relaciones que lo interconectan entre los Folderes de Item y sus Item miembros. El Folder de Item 900 tiene como miembros una pluralidad de Items 902, 904 y 906. El Folder de Item 900 tiene una Relación 912 con el Item 902 que denota que el Item 902 es público y se puede compartir con el Folder de Item 900, sus miembros 904 y 906, y cualquier otro Folder de Item, Categoría, o Item (que no se muestra) que se puede comunicar con el Folder de Item 900. Sin embargo, no hay Relación del Item 902 al Folder de Item 900 que denota que el Folder de Item 900 es privado al Item 902 y no

comparte su información de membresía con el Item 902. Item 904, por el contrario, tiene una Relación 924 con el Folder de Item 900 que denota que el Folder de Item 900 es público y comparte su información de membresía con el Item 904. Sin embargo no hay Relación de él Folder de Item 900 al Item 904 que denota que el Item 904 es privado y no se comparte con el Folder de Item 900, sus otros miembros 902 y 906, y cualquier otro Folder de Item, Categoría, o Item (que no se muestra) que puede tener acceso al Folder de Item 900. En contraste con su Relación (o ausencia) con los Items 902 y 904, el Folder de Item 900 tiene una Relación 916 con el Item 906 y el Item 906 tiene una Relación 926 de vuelta con el Folder de Item 900, que juntos denotan que el Item 906 es público y se comparte con el Folder de Item 900, sus miembros 902 1904, y cualquier otro Folder de Item, Categoría, o Item (que no se muestra) que puede tener acceso al Folder de Item 900, y que el Folder de Item 900 es público y comparte su información de membresía con el Item 906.

[0169] Como se discutió previamente, los Items en un Folder de Item no necesitan compartir una commonality porque los Folderes de Item no están "descritos". Las Categorías, a diferencia, son descritas por una commonality que es común a todos los Items miembros. En consecuencia la membresía a una Categoría esta inherentemente limitada a los Items que tienen la commonality descrita y, en ciertos implementaciones, todos los Items que satisfacen la descripción de una Categoría automáticamente se convierten en miembros de esa Categorías. Por lo tanto, mientras que los Folder de Item permiten estructuras de tipo trivial para representar su membresía, las Categorías permiten la membresía basada en la commonality descrita.

[0170] Claro está, las descripciones de las Categorías son lógicas por naturaleza, y por lo tanto una Categoría se puede describir con una representación lógica de tipos, propiedades, y/o valores. Por ejemplo, una representación lógica para una Categoría puede ser su membresía que comprende los Items que tienen una o dos propiedades o ambas. Si estas propiedades descritas por la Categoría son "A" y "B", entonces la membresía a la Categoría puede comprender Items que tienen la propiedad A y no la B, Items que tienen la propiedad B y no la A, y items que tienen ambas propiedades A y B. Esta representación lógica de propiedades se describe con el operador lógico "OR" donde el conjunto de miembros

descrito por la Categoría son los Items que tienen la propiedad A y B. Operadores lógicos similares (incluyendo sin limitación "AND", "XOR", y "NOT" solos o en combinación) también se pueden utilizar para describir una categoría como lo podrán apreciar aquellos diestros en el arte.

[0171] A pesar de las diferencias entre un Folder de Item (que no se describe) y las Categorías (descritas), las Relaciones de Categorías con un Item y las Relaciones de Items con Categorías son esencialmente la misma como se describió anteriormente con los Folder de Item y los Items en muchos implementaciones del presente invento.

[0172] La figura 10 es un diagrama de bloque que muestra una Categoría (que, de nuevo, es un Item en sí), sus Items miembros, y la Relación de interconexión entre la Categoría y sus Items miembros. La Categoría 1000 tiene como miembros a una pluralidad de Items 1002, 1004 y 1006, todos los cuales comparten alguna combinación de propiedades comunes, valores o tipos 1008 como se describe (descripción de commonality 1008) por la Categoría 1000. La Categoría 1000 tiene una Relación 1012 con el Item 1002 que denota que el Item 1002 es público y se comparte con la Categoría 1000, sus miembros 1004 y 1006, y cualquier otra categoría, Folder de Item, o Items (que no se muestran) que pueden accesar la Categoría 1000. Sin embargo, no hay Relación del Item 1002 con la Categoría 1000 que no denote que la Categoría 1000 es privada para el Item 1002 y no comparte su información de membresía con el Item 1002. El Item 1004, de lo contrario, tiene una Relación 1024 con la Categoría 1000 que denota que la Categoría 1000 es pública que comparten su información de membresía con el Item 1004. Sin embargo, no hay Relación que se extiende de la Categoría 1000 al Item 1004 que denote que el Item 1004 es privado y no se comparte con la Categoría 1000, por sus otros miembros 1002 y 1006, y cualquier otra Categoría, Folder de Item, o Items (que no se muestran) que pueden accesar a la Categoría 1000. En contraste con sus Relaciones (o ausencia) con los Items 1002 y 1004, la Categoría 1000 tiene una Relación 1016 con el Item 1006 y el Item 1006 tiene una Relación 1026 de vuelta con la Categoría 1000, que denota que el Item 1006 es público y se comparte con la Categoría 1000, sus Items miembros 1002 y 1004, y cualquier otra Categoría, Folder de Item, o Items (que no se muestra) que pueden accesar la Categoría 1000, y que la Categoría mismo es pública y comparte su información de membresía con el Item 1006.

[0173] Finalmente, porque las Categorías y los Folder de Item son en sí Items, y los Items se pueden Relacionar unos con otros, las Categorías se pueden Relacionar con los Folder de Item y viceversa, y las Categorías, Folder de Item, y Items pueden Relacionarse con otras Categorías, Folder de Item, y Item respectivamente en ciertas implementaciones alternativas. Sin embargo, en varias implementaciones, las estructuras de Folderes de Item y/o estructuras de Categorías se prohíben, en el nivel de sistema de interfase hardware/software, de tener ciclos. Donde el Folder de Item y la estructura de Categoría son parecidas a gráficas dirigidas, la implementación que prohíbe los ciclos es parecida a una gráfica dirigida acíclica (DAG) que, por definición matemática en el arte de la teoría gráfica, son gráficas dirigidas donde no hay un camino que comience o termine en el mismo vértice.

6. Extensibilidad

[0174] La plataforma de almacenaje debe tener un conjunto inicial de esquemas 340, como se describe anteriormente. Adicionalmente, sin embargo, en por lo menos unas implementaciones, la plataforma de almacenaje le permite a los consumidores, incluyendo vendedores independientes de software (ISVs), crear nuevos esquemas 344 (por ejemplo nuevos items y tipos de Elementos Anidados). Esta sección está dirigida a los mecanismos para crear tales esquemas extendiendo los tipos de Item y tipos de Elementos Anidados (o simplemente tipos de "Elementos") definidos en el conjunto de esquemas iniciales 340.

[0175] Preferiblemente, las extensiones del set inicial de Items y tipos de Elementos Anidados esta constreñida de la siguiente manera:

- se permite que un ISV introduzca nuevos tipos de Item, por ejemplo, su tipo de BaseItem;
- se permite que un ISV introduzca nuevos tipos de Elementos Anidados, por ejemplo, un subtipo de Base.NestedElement;
- se permite que un ISV introduzca nuevas extensiones, por ejemplo un subtipo de Base.NestedElement pero,

- Un ISV no puede subtipificar ningún tipo (Item, Elemento Anidado o tipos de Extensiones) definidos por el conjunto inicial de esquemas de la plataforma de almacenaje 340.

[0176] Puesto que un tipo de Item o un tipo de Elemento Anidado definido por el conjunto iniciados de esquemas de la plataforma de almacenaje pueden no exactamente coincidir con la necesidad ISV de una aplicación, es necesario permitir que el ISV adapte el tipo. Estos se permite con la noción de Extensiones. Las Extensiones son instancias que se afirman fuertemente pero (a) no pueden existir en forma independiente y (b) tienen que estar pegadas a un Item o un Elemento Anidado.

[0177] Adicionalmente a dirigimos a la necesidad de la extensibilidad de los esquemas, las Extensiones también se pretende que se dirijan al tema del "multi-typing". Desde que, en algunas implementaciones, la plataforma de almacenaje puede no ser compatible con herencias múltiples o subtipos que se traslapan, las aplicaciones pueden utilizar Extensión es como una forma de moldear instancias de traslapo de tipos (por ejemplo, Documento es un documento legal como también un documento seguro).

a) Extensiones de Items

[0178] Para proveer extensibilidad a los Item, el modelo de datos define adicionalmente un tipo abstracto llamado Base.Extension. Este es un tipo raíces para la jerarquía de tipos de extensión. Las aplicaciones pueden subtipificar a Base.Extension para crear un tipo específico de extensión.

[0179] El tipo Base.Extension se define en el esquema Base de la siguiente manera:

<Type Name="Base.Extension" IsAbstract="True">
<Propety Name="ItemID"
Type="the storage platformTypes.uniqueidentified"
Nullable="false"
MultiValued="false"/>

<Property Name="ExtensionID"
Type="the storage platformTypes.uniqueidentified"
Nullable="false"
MultiValued="false"/>
</Type>

[0180] El campo ItemID contiene el ItemID del item con el cual está asociada la extensión. Un Item con este ItemID debe existir. La extensión no se puede crear siete item con el ItemID dado no existe. Cuando el Item se elimina todas las extensiones con el mismo ItemID se elimina. El par (ItemID y ExtensiónID) identifica en forma única a una instancia de extensión.

[0181] La estructura de un tipo de extensiones similar al lado de un tipo de item:

Los tipos de Extensión tienen campos;

- Los campos pueden ser primitivos o tipos de elemento anidados; y
- Los tipos de Extensión pueden ser subtipos

[0182] las siguientes restricciones aplican a los tipos de extensión

- Las extensiones no puede ser fuente u objetivo de las relaciones;
- Las instancias de tipos de extensión no pueden existir independiente de un item; y
- Los tipos de extensión no pueden usarse como tipos de campo en las definiciones de tipo en la plataforma de almacenaje.

[0183] no hay constreñimientoen los tipos de extensión que se pueden asociar con un tipo de Item dado. Se permite que cualquiera tipo de extensión tipo de extensión extienda cualquier tipo de item. Cuando instancias de extensión múltiples se anexan a un item, son independientes tanto en estructura como un comportamiento.

[0184] Las instancias de extensión se guardan y se acceden separadas del item. Todas las instancias de tipos de extensión son accesibles en una vista de extensiones global. Se puede componer una consulta eficiente que devuelva todas las instancias de un tipo de extensión dado sin importar con que tipo de item están asociados. La plataforma de almacenajeAPI suministra un modelo de programación se puede archivar, recuperar y modificar las extensiones en los item.

[0185] Los tipos de extensión pueden ser subtipos de un tipo utilizando el modelo heredado único de la plataforma de almacenaje. Cuando se deriva de un tipo de extensión se crea un nuevo tipo de extensión. La estructura o el comportamiento de un extensión no puede primar por reemplazar la estructura o el comportamiento del tipo del item jerárquico. Similar a los tipos de Item, las instancias de tipo Extensión pueden recuperarse directamente a través de la vista asociada con el tipo de extensión. El ItemID de la extensión indica a que item pertenece tipo de utilizarse por recuperar el correspondiente objeto de item de la vista de Item global. Las extensiones se consideran parte del item por razones de consistencia operacional. El copiar/mover, resguardar/recuperar y otras operaciones comunes en la plataforma de almacenaje define pueden operar en las extensiones como parte de un item.

[0186] Considere el siguiente ejemplo. Un tipo de Contacto se define en la ventana con un conjunto de Tipo.

```
<Type Name="Contact" BaseType="Base.Item" >
<Property Name="Name"
   Type="String"
   Nullable="false"
   MultiValued="false"/>
<Property Name="Address"
   Type="Address"
   Nullable="true"
   MultiValued="false"/>
</Type>
```

[0187] A un diseñador de aplicaciones CRM le gustaría anexar una extensión de aplicaciónCRM a los contactos almacenados en la plataforma de almacenaje. El diseñador del aplicación definiría una extensiónCRM que contendría la estructura de datos adicional que la aplicación puede manipular.

```
<Type Name="CRMExtension" BaseType="Base.Extension" >
<Property Name="CustomerID"
  Type="String"
  Nullable="false"
  MultiValued="false"/>
      ...
</Type>
```

[0188] Un diseñador de aplicación HR podría también querer anexar datos con el Contacto. Estos datos son independientes de los datos de la aplicación CRM. De nuevo el diseñador a puede crear una extensión.

```
<Type Name="HRExtension" EBaseType="Base.Extension" >
<Property Name="EmployeeID"
  Type="String"
  Nullable="false"
  MultiValued="false"/>
      ...
</Type>
```

[0 189] La CRMExtensión y HRExtensión son dos extensiones independientes que se pueden anexar a los items Contacto. Ellos se crean y se recuperan en forma independiente el uno del otro.

[0190] En el ejemplo anterior, los campos y los métodos del tipo CRMExtensión no pueden anular los campos y los métodos de la jerarquía Contacto. Se debía notar que instancias del tipo CRMExtensión pueden anexarse a los tipos de Item distintos de Contacto.

[0191] cuando item Contacto se recupera, su extensión de item no se recupera automáticamente. Dado un item de Contacto, su extensión de item relacionada puede ser recuperada por una consulta en la vista global de extensiones con el mismo ItemID. En

[0192] Todas las extensiones CRMExtensión en el sistema se pueden recuperar a través de la vista de los tipos CRMExtensión , sin importar a que item le pertenece. Todo las extensiones de item de un item comparten el mismo itemid. En el ejemplo anterior, el item Contacto y su CRMExtensión y HRExtensión anexa tienen el mismo ItemID.

[0193] la siguiente tabla resume las similitudes y diferencias entre Item, Extensión y tipos de ElementosAnidados.

Item vs Extensiones de Item vs ElementosAnidados

	Item	Item Extensión	NestedElement
Item ID	Tiene su propio id	Comparte el item id del item	No tiene su propio item id. Elementos anidados son parte del item
Almacenaje	La jerarquía de items se guarda en su propia tabla	La jerarquía de extensiones de items se guarda en su propia tabla	Se guarda con el item
Consulta/búsqueda	Puede consultar las tablas de items	Puede consultar las tablas de extensiones de items	Puede ser consultado únicamente dentro del contexto del item que lo contiene
Consulta/búsqueda alcance	Puede consultar a través de todas las instancias de un tipo de item	Puede consultar a través de todas las instancias de una extensión de tipo de item	Puede buscar únicamente dentro del tipos de elementos anidados de un item (contenedor) individual
Relación semántica	Puede tener Relaciones con un item	No tiene relaciones con Extensiones de items	No tiene relaciones con los Elementos Anidados
Asociación con los items	Puede estar relacionado con otros items a través de Relaciones de Pertenencia,	Generalmente solo se puede relacionar a través de extensiones. La semántica de la	Relacionada con los items a través de los campos. Los elementos anidados son parte del item

	Incrustadas, y suaves.	extensión es similar a la del item incrustado.

b) Extendiendo tipos de Elementos Anidados

[0194] Los tipos de Elementos Anidados no se extienden con el mismo mecanismo de los tipos de Item junta las extensiones de los elementos anidados se guardan y se recuperan con el mismo mecanismo de los campos de tipos de elementos anidados.

[0195] El modelo de datos define una raíz para los tipos de elementos anidados llamado Elementos:

```
<Type Name="Element"
    IsAbstract="True">
<Property Name="ElementID"
  Type="the storage platformTypes.uniqueidentifier"
  Nullable="false"
  MultiValued="false"/>
</Type>
```

[0196] Los tipos de ElementosAnidados heredan de este tipo. El tipo de elemento ElementosAnidados define adicionalmente un campo que es un multi conjunto de Elementos.

```
<Type Name="NestedElement" BaseType="Base.Element"
IsAbstract="True">
<Property Name="Extensions"
  Type="Base.Element"
  Nullable="false"
  MultiValued="true"/>
```

</Type>

[0197] La extensión Elementos Anidados es distinta de la extensión de los items de la siguiente manera:

- Las extensiones de elementos anidados no son tipos de extensión. No pertenecen a la jerarquía de tipos de extensión que está enraizada en el tipo Base.Extensión.
- Las extensiones de elementos anidados se guardan junto con los otros campos del item y no tienen un acceso global -una consulta no se puede componer de tal manera que recupere todas las instancias de un tipo de extensión dado.
- Estas extensiones se guardan de la misma manera que los demás elementos anidados (de los items) se guardan. Como otros conjuntos anidados, la extensión ElementosAnidados se guarda en un UDT. Son accesibles a través del campo Extensiones del tipo de elemento anidados.
- La colección de interfases que se utilizan para acceder a las propiedades multi-valor también se utiliza para acceder conjuntos de tipos de extensión.

[0198] La siguiente tabla en resume y compara los Extensiones de Item y las extensiones de ElementosAnidados.

Extensiones de Item vs Extensiones de ElementosAnidados

	Extensiones de Item	Extensiones de ElementosAnidados
Almacenaje	La jerarquía de extensiones de items se guarda en sus propias tablas	Se guarda como elementos anidados
Consulta/Busqueda	Puede consultar tablas de extensión de items	Generalmente solo se puede consultar dentro del contexto de item que lo contiene
Consulta/Busqueda alcance	Puede consultar a través de todas las instancias de un tipo de extensión de item	Generalmente solo puede consultar dentro de instancias de tipos de elementos anidados de un item(contenedor) individual
Capacidad de programación	Necesita una extensión especial API y una consulta especial en las tablas de extensiones	Extensiones NestedElement son como cualquier otro campo multi-valor de elementos anidados; tipos de elementos anidados normales APIs se usan
Comportamiento	Puede asociar el comportamiento	No se permite el comportamiento (?)
Relaciones semantica	No hay Relaciones con las extensiones de los items	No hay relación con las extensiones NestedElement
Item ID	Comparte el itemid del	No tiene su propio item id.

item	La extensión NestedElement es parte del item

F. MOTOR DE BASE DE DATOS

[0199] Como se mencionó anteriormente, el almacén de datos se implementa en un motor de base de datos. En la presente implementación, el motor de la base de datos comprende un motor de base de datos relacional que implementa un lenguaje de consulta SQL, tal como el motor del Servidor Microsoft SQL, con extensiones relacionales de objetos. Esta sección describe el mapeo del modelo de datos que el almacén de datos implementa con el almacén relacional y nos da información en un API lógico que el cliente de la plataforma de almacenaje consume, de acuerdo con la presente implementación. Se entiende, sin embargo que un mapeo diferente se puede utilizar cuando un motor de base de datos diferente se utiliza. De hecho, adicionalmente a implementar el modelo de datos conceptual de la plataforma de almacenaje en un motor de base de datos relacional, también se puede implementar en otros tipos de bases de datos, por ejemplo bases de datos orientadas a los objetos y bases de datos XLM.

[0200]en Un sistema de base de datos orientada hacia los objetos (OO) provee persistencia y transacciones para el lenguaje de programación de objetos (por ejemplo, C++, Java). La noción, en la plataforma de almacenaje, de un "items" mapea bien con el de un "Objeto" en un sistema orientado hacia los objetos, aunque colecciones incrustadas tendrían que añadirse a los Objetos. Otros conceptos de tipos de plataforma de almacenaje, como tipos de elementos heredados y anidados, también mapean con sistemas de tipos orientados hacia los objetos. Sistemas orientados a los objetos utilizan típicamente la identidad de los objetos; por lo tanto, la identidad de un item se puede mapear con la identidad de un objeto. El comportamiento de los items (operaciones) mapea bien con los métodos de los objetos. Sin embargo, en sistemas orientados hacia los objetos falta capacidad de organización y son muy malos en las búsquedas. También los sistemas orientados hacia los objetos no apoyan

los datos sin estructura o con semi estructura. Para apoyar completamente el modelo de datos de la plataforma de almacenaje aquí descrita, conceptos como relaciones, folderes, y extensiones se tienen que añadir al modelo de datos basado en objetos. Adicionalmente, mecanismos como las promociones, sincronización, notificaciones, y seguridad tendrían que implementares.

[0201] Similar a los sistemas orientados a los objetos, bases de datos XLM, basados en XSD (definición de esquemas XML), son compatibles con un sistema de tipos basado en herencias individuales. El sistema de tipo de item del presente invento se podría mapear al tipo de modelo XSD. El XSD tampoco es compatible con los comportamientos. El XSD para los items tendría que aumentarse con el comportamiento de los items. Una base de datos XML trabaja con documentos individuales XSD y le hace falta la organización y la capacidad de búsquedas amplias. Al igual que con las bases de datos orientadas hacia los objetos, para apoyar el modelo de datos descrito aca, otros conceptos como relaciones, y folderes tendrían que introducirse en dicha base de datos XML; también mecanismos como la sincronización, notificación y seguridad tendrían que implementarse.

[0202] En cuanto a la siguiente subsección, se muestran unas cuantas ilustraciones para facilitar la información general dada: la figura 13 es un diagrama que muestra el mecanismo de notificación. La figura 14 es un diagrama que muestra un ejemplo en donde dos transacciones ambas están insertando un nuevo récord en el mismo Árbol-B. La figura 15 muestra un proceso de detección de cambios de datos. La figura 16 muestra un árbol directorio ejemplar. La figura 17 muestra un ejemplo en donde un folder existente de un sistema de archivos basado en directorios se mueve a la plataforma del almacén de datos.

1. Implementación de una almacén de datos utilizando UDTS

[0203] En la presente implementación, el motor relacional de la base de datos 314 que en una implementación se constituye en el motor del Servidor Microsoft SQL, es compatible con títulos escalares internos. Los tipos escalares internos son "nativos" y "simples". Son nativos en el sentido de que el usuario no puede definir sus propios tipos y son simples en

el sentido de que no pueden encapsular estructuras complejas. Tipos definidos por el usuario (de aquí en adelante: UDTS) suministran un mecanismo para la extensibilidad de los tipos por encima y más allá de los sistemas de tipos nativos escalares permitiéndole al usuario extender el sistema de tipos definiendo tipos complejos estructurados. Un UDT una vez se define por un usuario, se puede usar en cualquier parte del sistema de tipos que un tiempo escalar interno podría usar.

[0204] De acuerdo con un aspecto del presente invento, el esquema de la plataforma de almacenaje está mapeado a clases de UDT en el almacén del motor de la base de datos. Los Items en el almacén de datos se mapean a clases de UDT que se derivan del tipo Base.Item. Como los Items, las Extensiones también se mapean a clases de UDT y hacen uso de la herencia. El tipo raíz de Extensión es Base.Extensión, de donde todos los tipos de Extensión se derivan.

[0205] un UDT es una clase CLR -tiene un estado (por ejemplo campos de datos) y un comportamiento (por ejemplo, rutinas). Los UDT se definen utilizando cualquiera de los lenguajes usados -C#.VB.NET,etc. Los métodos y operadores UDT se pueden invocar en una T-SQL contra una instancia de ese tipo. Un UDT puede ser: el tipo de la columna en una fila, el tipo de un parámetro de una rutinas en T-SQL, el tipo de una variable en T-SQL.

[0206] El mapeo del esquema de la plataforma de almacenaje a clases UDT es bastante directo en los altos niveles. Generalmente, un esquema de plataforma de almacenaje se mapean al espacio de nombre CLR. Un Tipo de plataforma de almacenaje se mapean a una clase CLR. La herencia de la clases CLR es una replica de la herencia del tipo de la plataforma de almacenaje, y una propiedad de la plataforma de almacenaje se mapean a una propiedad clase CLR.

2. El mapeo de Items

[0207] Dado el deseo de poder buscar los Items en forma global, y su compatibilidad en la base de datos relacional de la presente implementación para las herencias y la sustitución de

tipos, una posible implementación para el almacenaje de Items en el almacén de la base de datos sería almacenar todos los items en una sola tabla con una columna de tipo Base.Item. Utilizando la sustitución de tipos, Items de todos los tipos se podrían almacenar, y las búsquedas se podrían filtrar con el tipo de Item y subtipo utilizando el operador Yukon ("es del *(Tipo)*").

[0208] Sin embargo, debido a la preocupación con el exceso asociado a éste enfoque, en la presente implementación, los Items se dividen en tipos de alto nivel, tales como los Items de cada tipo de "familia" que se almacenan en una tabla separada. Bajo este esquema de partición, una tabla se crea para cada tipo de Item heredado directamente de Base.Item. Los tipos que heredan debajo de estos se almacenan en la tabla del tipo de familia apropiada utilizando sustitución de tipos, como se describió más arriba. Únicamente el primer nivel de herencia de Base.Item se trata en forma especial.

[0209] Una tabla "sombra" se usa para guardar copias de propiedades que se pueden buscar globalmente para todos los Items. Esta tabla se puede mantener con el método Update () de la plataforma de almacenaje API, a través de la cual todos los cambios en los datos se hacen. A diferencia de estas tablas de tipos de familia, esto tabla de Items global contiene únicamente las propiedades escalares de alto nivel del Item, no el completo objeto Item UDT. La tabla de items globales permite la navegación al objeto Item almacenado en la tabla del tipo de familia exponiendo un ItemID y un TypeID. El ItemID generalmente identificará en forma única el Item dentro del almacén de datos. El TypeID se puede mapear utilizando meta datos, que no se describen aca, a un nombre del tipo y la vista que contiene el Item. Como encontrar un Item por su ItemID puede ser una operación común, tanto en el contexto de la tabla de Items globales y demás, una función GetItem () se provee para recuperar un objeto de Item dado el ItemID de un Item.

[0210] Para un acceso conveniente y esconder la implementación de los detalles en lo posible, todas las consultas de Items pueden hacerse contra vistas construidas en las tablas de los Items descritos anteriormente. Específicamente, una vista se puede crear para cada tipo de Item contra la tabla del tipo de familia apropiado. Estas vistas de tipos pueden seleccionar todos los Items de un tipo asociado, incluyendo los subtipos. Para conveniencia,

adicional al objeto UDT , las vistas pueden exponer columnas para todos los campos de alto nivel de ese tipo, incluyendo los campos heredados.

3. Mapeo de Extensiones

[0211] Las extensiones son similares a los Items y tienen algunos de los mismos requerimientos. Como el soporte de otra herencia de tipo raíz, las Extensiones están sujetas a muchas de las mismas consideraciones y similitudes en el almacenaje. Por esto, un mapeo a un tipo de familia similar se aplica a las Extensiones, en vez de un enfoque de tabla única. Claro está, en otros implementación se puede utilizar una tabla única. En la presente implementación, una Extension se asocia con exactamente un Item a través de un ItemID, que contiene una ExtensioneID que es única en el contexto del Item. Al igual que con los Items, se puede proveer una función para recuperar una Extensión dada su identidad, que consistiría de un par ItemID y ExtensiónID. Se crea una vista para cada tipo de Extensión, similar a las vistas de los tipos de Item.

4. Mapeo de Elementos Anidados

[0212] Los Elementos Anidados son tipos que se pueden incrustar en un Item, una Extensión, una Relación, u otros Elementos Anidados para formar estructuras anidadas profundas. Es así como los Items y Extensiones y los Elementos Anidados se implementan como un UDT, pero se almacenan dentro de un Item y Extensión. Por lo tanto, los Elementos Anidados no tienen ningún mapeo de almacenaje más allá de los contenedores de su Item y Extensión. En otras palabras, no hay tablas del sistema que directamente almacén las instancias de tipos Elementos Anidados, y no hay vistas dedicadas específicamente a Elementos Anidados.

5. Identidad de Objetos

[0213] Cada entidad en el modelo de datos, por ejemplo cada Item, Extensión y Relación, tiene un valor de llave único. Un Item se identificará en forma única por su ItemID una, Extensión se identificará en forma única con una llave compuesta (ItemId, RelaciónId). ItemId, RelaciónId y ExtensiónId son valores GUID.

6. Nombrando objetos SQL

[0214] Todos los objetos creados en el almacén de datos se pueden guardar con un nombre de un esquema SQL derivado del nombre de esquema de la plataforma de almacenaje. Por ejemplo, la plataforma de almacenaje esquema Base (llamada frecuentemente "Base") puede producir tipos en el esquema SQL "(System.Storage)" tal como "(System.Storage)".Item. Los nombres generados llevan un prefijo de una calificadora para eliminar el nombrar los conflictos. En donde es apropiado, un carácter de exclamación (!) se usa como separador para cada parte lógica del nombre. La tabla a continuación delinea las convenciones utilizadas para nombrar objetos en el almacén de datos. Cada esquema de elemento (Item, Extensión, Relación y Vista), se enumera junto con la convención de nombres decorados utilizadas para acceder a las instancias en el almacén de datos.

Object	Nombre de Decoración	Description	Example
Vista Master de búsqueda de Items	Master!Item	Provee un sumario de items en el dominio actual.	[System.Storage]. [Master!Item]
Vista de búsqueda de Items Tipificados	*ItemType*	Provee todos los datos de propiedad del *item* y sus tipos paternos.	[AcmeCorp.Doc]. [OfficeDoc]

Object	Nombre de Decoración	Description	Example
Vista de búsqueda de Extensión Master	Master!Extension	Provee un sumario de todas las extensiones en el dominio del item actual.	[System.Storage]. [Master!Extension]
Vista de búsqueda de Extensión tipificada	Extension!*extensionType*	Provee todos los datos de propiedad para la extensión.	[AcmeCorp.Doc]. [Extension!StickyNote]
Vista Master de Relaciones	Master!Relationship	Provee un sumario de todas las relaciones en el dominio actual del item.	[System.Storage]. [Master!Relationship]
Vista de Relaciones	Relationship!*relationship Name*	Provee todos los datos asociados a una relación dada.	[AcmeCorp.Doc]. [Relationship!AuthorsFromDocument]
Vista	View!*viewName*	Provee el columnas/tipos basado en la definición de equema *view*	[AcmeCorp.Doc]. [View!DocumentTitles]

7. Nombrado de Columnas

[0215] Cuando se mapea un modelo de objetos a un almacén, la posibilidad de una colisión de nombres puede ocurrir debido a la información adicional que se guarda con un objeto de

la aplicación. Para evitar la colisión de nombres, todas las columnas específicas no-tipo (columnas que no se mapean directamente a un hombre Propiedad en una declaración de tipo) se prefijan con un caracter (_).En la presente implementación, los caracteres (_) no se permiten al comienzo de ninguna propiedad de identificación. Aún más, para unificar los nombres entre el CLR si el almacén de datos, todas las propiedades de los tipos en la plataforma de almacenaje o esquema de elementos (relaciones, etc.) deben tener la primer letra capitalizada.

8. Vistas de búsqueda

[0216] La plataforma de almacenaje suministra vistas para la búsqueda del contenido almacenado. Una vista SQL se suministra para cada Item y tipo de Extensión. Aún más se suministran vistas para apoyar las Relaciones y Vistas (tal como se definen en el modelo de datos). Todas las vistas SQL y tablas contenidas en la plataforma de almacenaje son únicamente de lectura. Los datos se pueden guardar o cambiar utilizando el método Update () de la plataforma de almacenaje API, tal como se describe en más detalle más adelante.

[0217] Cada vista define explícitamente en el esquema de la plataforma de almacenaje (definida por el diseñador del esquema, y no generada automáticamente por la plataforma de almacenaje) se puede acceder a través de la vista llamada SQL [<schema-name>].[View!<view-name>]. por ejemplo, una vista llamada "BookSales" en el esquema "AcmePublisher.Books" se haría accesible utilizando el nombre "[AcmePublisher.Books].[View!BookSales]". Como el formato de salida de una vista está predefinido en base a cada vista (definido por una consulta arbitraria dada por la parte que define la vistas), las columnas se mapean directamente basadas en las definiciones de la vista del esquema.

[0218] Todas las vistas de búsquedas SQL en la plataforma de almacenaje utilizan la siguiente convención para ordenar las columnas:

- Vistas de columnas de "Llaves" lógicas tales como *ItemId, ElementId, RelationshipId,*

- Información de meta datos en el tipo de resultado tal como TypeId.
- Columnas de cambio de seguimiento tales como CreateVersion, UpdateVersion,..
- Columnas específicas de Tipo (s) (propiedades del tipo declarado)
- Vistas específicas de Tipos (vistas de familia) que también contienen columnas de objetos que devuelven el objeto

[0219] Los miembros de todo tipo de familia se pueden buscar utilizando una serie previstas de Item, donde hay una vista por tipo de Item en el almacén de datos. La figura 28 es un diagrama que muestra el concepto de una vista de búsqueda de Item.

a) Item

[0220] Cada vista de búsqueda de Item contiene una fila para cada instancia del Item el tipo específico o de sus subtipos. Por ejemplo, la vista para Documento puede regresar una instancia de Documento, LegalDocumento y Review.Documento. Con este ejemplo, las vistas de Item se pueden conceptualizar cómo se muestran en la figura 29.

(1) vista maestra de búsqueda de un Item

[0221] Cada instancia de un almacén de datos en la plataforma de almacenaje define una vista de Item especial llamada *Master Item View.* Esta vista entregar información resumida década Item en el almacén de datos. La vista muestra una columna por cada propiedad de tipo de Item, una columnas que describe el tipo del Item y varias columnas que se utilizan para dar información de sincronización y cambio de seguimientos. La vista de item master se identifica en el almacén de datos utilizando el nombre "[System.Storage].[Master!Item]".

Columnas	Tipos	Descripción
ItemId	ItemId	La identidad del item en la plataforma de almacenaje

_TypeId	TypeId	El TypeId del Item – identifica el item exacto y se puede usar para recuperar información sobre el tipo utilizando un catalogo de metadatos .
_RootItemId	ItemId	El ItemId del primer ancestro no incrustado que controla el tiempo de vida de ese item.
<global change tracking>	...	Información de cambio de seguimiento global
<Item props>	n/a	One column per Item type property Una columna para cada propiedad de tipo de Item

(2) vistas de búsqueda de Items tipificados

[0222] Cada tipo de Item también tiene una vista de búsqueda. Mientras que es similar a la vista de Item raíz, esta vista también suministra acceso al objeto del item a través de la columna "_ Item". Cada vista de búsqueda búsqueda de items tipificados se identifica en el almacén de datos utilizando el nombre [*schemaName*].[*itemTypeName*].. Por ejemplo, AcmeCorp.Doc].[OfficeDoc].

Column	Type	Description
ItemId	ItemId	La identidad el item en la plataforma de almacenaje
<type change tracking>	...	Información sobre el cambio de seguimiento
<parent props>	<property specific>	Una columna para cada propiedad paterna
<item props>	<property specific>	Una columna para cada propiedad exclusiva de este tipo
_Item	CLR tipo del	O TT-SQL u aquí a a servicio bjecto CLR – tipo

	Item	del Item declarado

b) Extensiones de Items

[0223] Todas las Extensiones de Item en un Almacén WinFS también son accesibles utilizando vistas de búsqueda.

(1) Vista Maestra de Búsqueda de Extensiones

[0224] Cada instancia de un almacén de datos define una vista de Extensión especial llamada el Master Extension View. Esta vista suministra información resumida sobre cada Extensión en el almacén de datos. La vista tiene una columna para cada propiedad de Extensión, una columna que describe el tipo de Extensión y varias columnas que se utilizan para hacer seguimiento a los cambios de seguimiento e información de sincronización. La vista maestra de las extensiones se identifica en el almacén de datos utilizando el nombre "[System.Storage].[Master!Extension]".

Column	Type	Description
ItemId	ItemId	la identidad en la plataforma de almacenaje del item o no con el cual esta extensión se asocia
ExtensionId	ExtensionId (GUID)	Id de esta instancia extensión
_TypeId	TypeId	El TypeId de la Extensión identifica el tipo exacto de la extensión y se puede utilizar para recuperar información sobre la extensión usando el catálogo de meta datos
<seguimiento de cambios globales >	...	Información de seguimiento de cambios globales
<propiedades ext >	<property specific>	Una columna por propiedad de tipo de Extensión

(2) Vistas de Búsqueda de las Extensiones Tipificadas

[0225] Cada tipo de extensión también tiene una vista de búsqueda. Mientras que es similar a la vista maestra de extensiones, esta vista también provee acceso a los objetos del Item a través de -Extensióncolumn. Cada vista de extensión tipificada se identifica en el almacén de datos utilizando el nombre [*schemaName*].[**Extension!***extensionTypeName*]. Por ejemplo [AcmeCorp.Doc].[Extension!OfficeDocExt].

Columna	Tipo	Descripción
ItemId	ItemId	La identidad del item en la plataforma de almacenaje con el cual esta extensión esta asociada
ExtensionId	ExtensionId (GUID)	Id de la instancia de esta extensión
<type change tracking>	...	Información de cambio de vista
<parent props>	<específico a la propiedad>	Una columna por propiedad a paterna
<ext props>	<específico a la propiedad>	Una columna por propiedad exclusiva de este tipo
_Extension	CLR tipo de instancia de Extension	objectO CLR–tipo de Extension declarada

C) Elementos Anidados

[0226] Todos los Elementos Anidados están almacenados dentro de instancias de los Items, Extensiones o Relaciones. Como tal, se encuentran a través de consultar la vista de búsqueda del Item apropiado, Extensión o Relación.

d) Las Relaciones

[0227] Como se discutió anteriormente, las Relaciones forman la unidad fundamental de vínculo entre los Items en el almacén de datos de la plataforma de almacenaje.

(1) La Vista de Búsqueda Maestra de las Relaciones

[0228] Cada almacén de datos suministra un *Master Relationship View.* Esta vista suministra información sobre todos las instancias de una relación en el almacén de datos. La vista maestra de relaciones se identifica en un almacén de datos utilizando el nombre "[System.Storage].[Master!Relationship]".

Columnas	Tipo	De descripción
ItemId	ItemId	Identidad del punto final fuente (ItemId)
RelationshipId	RelationshipId (GUID)	el id de la instancia de la relación
_RelTypeId	RelationshipTypeId	El RelTypeId de la relación-identifica el tipo de relación utilizando el catálogo de meta datos..
<seguimiento de cambios globales>	...	Información de seguimiento de cambios globales
TargetItemReference	ItemReference	Identidad del punto final objetivo
_Relación	Relaciones	Instancias del objeto de Relación para esta instancia

(2) Vistas de Búsqueda de Instancias de una Relación

[0229] Cada Relación declarada también tiene una vista de búsqueda se devuelve todas las instancias de una relación articular con mientras que es similar a la vista maestra de relaciones, estadista también provee columnas nombradas para cada propiedad de los datos

de la relación. Cada vista de búsqueda de instancias de unos relación se identifica en el almacén de datos utilizando un nombre [*schemaName*].[**Relationship**!*relationshipName*]. Por ejemplo [AcmeCorp.Doc].[Relationship!DocumentAuthor].

Columna	Tipo	Descripción
ItemId	ItemId	Identifica el punto final fuente (ItemId)
RelationshipId	RelationshipId (GUID)	el id de la instancia de la relación
<type change tracking>	...	Información de seguimiento de cambios de tipo
TargetItemReference	ItemReference	Identidad del punto final objetivo
<source name>	ItemId	Propiedad nombrada de la identidad del punto final fuente (alias para ItemId)
<target name>	ItemReference o clase derivada	Propiedad nombrada de la identidad del punto final objetivo (alias para TargetItemReference)
<rel property>	<específico a la propiedad>	Una columna por propiedad de la definición de la relación
_Relación	Tipo CLR de instancia de una Relación	objecto CLR–tipo de Relación declarada

e)

9. Actualizaciones

[0230] Todos las vistas en el almacén de datos de la plataforma de almacenaje son únicamente de lectura. Para crear instancias nuevas de un elemento en el modelo de datos

(item, extensión o relación), o para actualizar una instancia existente, se usa el ProcesoOperación o método ProcessUpdategram de la plataforma de almacenaje API. El método de ProcesoOperación es un procedimiento único almacenado y definido por el almacén de datos que consume una "operación" que detalla una acción a ejecutarse. El método ProcessUpdategram es un procedimiento almacenado que toma un conjunto de operaciones ordenadas, conocido como "Updategram", que colectivamente detalla una serie de acciones a ejecutarse.

[0231] El formato de operación es extensible y provee varias operaciones sobre los elementos del esquema. Algunas operaciones incluyen:

1. Operaciones de Items:
    a. CreateItem (crear un nuevo item en el contexto de una relación de incrustación o de pertenencia)
    b. UpdateItem (actualiza un item existente)
2. Operaciones de Relaciones:
    a. CreateRelationship (crea una instancia de una relación de referencia o de pertenencia)
    b. UpdateRelationship (actualiza una instancia de una Relación)
    c. DeleteRelationship (elimina las instancias de una Relación)
3. Operaciones con Extensiones:
    a. CreateExtension (añade una extensión a un Item existente)
    b. UpdateExtension (actualiza una extensión existente)
    c. DeleteExtension (elimina una extensión)

10. Cambio de Seguimiento y Lápidas

[0232] Los servicios de cambio de Seguimiento y lápidas son suministrados por el almacén de datos, como se discute en detalle más abajo. Esta sección suministra un bosquejo de la información de cambio de seguimiento expuesta en un almacén de datos.

a) Cambio de Seguimiento

[0233] Cada vista de búsqueda suministrada por el almacén de datos contiene columnas que se utilizan para dar información sobre cambio de Seguimiento; las columnas son comunes a través de todas las vistas de Item, Extensiones y Relaciones. Las vistas de esquema de la plataforma de almacenaje que definen explícitamente los diseñadores de esquema, no suministra automáticamente información sobre el cambio de Seguimiento, esta información se suministra indirectamente a través de vistas de búsqueda en las cuales la vista en sí se construye.

[0234] Para cada elemento en el almacén de datos, la información de cambios de seguimiento se encuentra disponible en dos sitios -la vista de elemento "máster" y la vista de elementos "tipificados". Por ejemplo, información de cambio de seguimiento en el tipo de Item está disponible de la vista Máster del Item "[System.Storage].[Master!Item]" y de la vista de búsqueda del Item tipificado [AcmeCorp.Document].[documento].

(1) Cambio de Seguimiento en las vista de búsqueda "maestra"

[0235] La información de cambio de vista en la vista de búsqueda maestra suministra información sobre la creación y actualización de versiones de un elemento, información en la que el compañero sync creó el elemento que el compañero sync actualizó últimamente y el elemento y el número de versión de cada compañero para la creación y actualización. Los compañeros en una relación sync (descrita más abajo) se identifican por partner key. Un objeto único UDT llamado _ChangeTrackingInfo del tipo [System.Storage.Store]. ChangeTrackingInfo contiene toda esta información. El tipo se define en el esquema System.Storage . _ChangeTrackingInfo está disponible en todos las vistas de búsqueda global para un Item, Extensión y Relación. La definición del tipo de ChangeTrackingInfo es:

```
<Type Name="ChangeTrackingInfo" BaseType="Base.NestedElement">
    <FieldProperty Name="CreationLocalTS"        Type="SqlTypes.SqlInt64"
    Nullable="False" />
    <FieldProperty Name="CreatingPartnerKey"
    Type="SqlTypes.SqlInt32"        Nullable="False" />
```

```
<FieldProperty Name="CreatingPartnerTS"          Type="SqlTypes.SqlInt64"
Nullable="False" />
<FieldProperty Name="LastUpdateLocalTS"
Type="SqlTypes.SqlInt64"          Nullable="False" />
<FieldProperty Name="LastUpdatingPartnerKey"  Type="SqlTypes.SqlInt32"
Nullable="False" />
<FieldProperty Name="LastUpdatingPartnerTS"   Type="SqlTypes.SqlInt64"
Nullable="False" />
</Type>
```

estas propiedades contienen la siguiente información:

Columnas	**Descripción**
_CreationLocalTS	Creación del sello de tiempo por la máquina local
_CreatingPartnerKey	PartnerKey del compañero que creó esta entidad. Si la entidad a ser creó localmente, éste es la PartnerKey de la máquina local.
_CreatingPartnerTS	Timestamp del tiempo en el cual esta entidad se creó en el compañero correspondiente a_CreatingPartnerKey.
_LastUpdateLocalTS	Timestamp local correspondiente al tiempo de actualización de la máquina local
_LastUpdatingPartnerKey	PartnerKey del compañero que actualizó esta entidad la última vez. Si la última actualización a la entidad se hizo en forma local, éste es el PartnerKey de la máquina local.
_LastUpdatingPartnerTS	Timestamp del tiempo en el cual esta entidad se cotizó en el compañero correspondiente a_LastUpdatingPartnerKey.

(2) Vistas de búsqueda de cambio de Seguimiento "tipificadas"

[0236] Adicionalmente se suministra la misma información que en la vista de búsqueda global, cada vista de búsqueda tipificada provee información adicional que graba el estado de sync de cada elemento en la topología sync.

Columnas	Teclee	Descripción
<seguimiento de cambios globales>	...	Información de seguimiento de cambios globales
_ChangeUnitVersions	MultiSet<ChangeUnitVersion>	Descripción de la versión de números del cambio de unidades dentro de un elemento particular
_ElementSyncMetadata	ElementSyncMetadata	Versión adicional independiente de meta datos acerca de este item que es únicamente de interés para el runtime de Sincronización con
_VersionSyncMetadata	VersionSyncMetadata	Versión específica adicional de meta datos acerca de esta versión que es únicamente de interés para el runtime de Sincronización

b) Lápidas

[0237] El almacén de datos suministra información de lápida para los Items, Extensiones y Relaciones. La vista de lápidas suministra información acerca de entidades vivas y con lápidas (items, extensiones y relaciones) en un determinado sitio. La vista de lápidas de items y extensiones no suministra un acceso al objeto correspondiente, mientras la vista de lápidas de relaciones suministra acceso a las relaciones de un objeto (la relación del objeto es NULL en el caso de una relación con lápida).

(1) Lápidas de Items

[0238] Las lápidas de Items se recuperan del sistema a través de una vista [System.Storage].[Tombstone!Item].

Columnas	Tipo	Descripción
ItemId	ItemId	Identidad del Item
_TypeID	TypeId	Tipo del Item
<Item properties>	...	Propiedades definidas para todos los items
_RootItemId	ItemId	ItemId del primer item no empotrado que contiene este items
_ChangeTrackingInfo	Instancia CLR del tipo ChangeTrackingInfo	Información de cambio de seguimiento para este items
_IsDeleted	BIT	Esta es una bandera que es 0 para items vivos, y 1 para items con lápida.
_DeletionWallclock	UTCDATETIME	La fecha y hora del reloj de pared UTC de acuerdo con el compañero que eliminó el item. Es NULL si el item está vivo.

(2) Lápidas de Extensiones

[0239] Las lápidas de las extensiones se recuperan del sistema utilizando la vista [System.Storage].[Tombstone!Extension]. La información de seguimiento de cambios en la Extensión es similar a la suministrada para los Items con la propiedad adicional ExtensionId.

Columnas	Tipos	Descripción
ItemId	ItemId	Identidad del IItem que es dueño de la Extensión
ExtensionId	ExtensionId	Extensión Id de la Extensión
_TypeID	TypeId	Tipo de la extensión
_ChangeTrackingInfo	CLR instance of type ChangeTrackingInfo	Información de seguimiento de cambios para esta extensión
_IsDeleted	BIT	Esta bandera es 0 para items vivos, y 1 para extensiones con lápida
_DeletionWallclock	UTCDATETIME	La fecha y hora del reloj de pared UTC de acuerdo con el compañero que eliminó el item. Es NULL si la extensión está viva.

(3) Lápidas de Relaciones

[[0240] Las lápidas de relaciones se recuperan del sistema a través de la vista [System.Storage].[Tombstone!Relationship]. La información de las lápidas de las Relaciones es similar a la de las Extensiones. Sin embargo se da información adicional sobre el objetivo ItemRef de la instancia de la relación. Adicionalmente el objeto de la relación también se selecciona.

Columnas	Tipo	Descripción en
ItemId	ItemId	Identidad del IItem que es dueño de

		la Relación (identidad del punto final fuente de la relación)
RelationshipId	RelationshipId	RelationshipId de la relación
_TypeID	TypeId	Tipo de la relación
_ChangeTrackingInfo	Instancia CLR de un tipo ChangeTrackingInfo	Información de seguimiento de cambios para esta relación
_IsDeleted	BIT	Esta bandera es 0 para items vivos, y 1 para extensiones con lápida.
_DeletionWallclock	UTCDATETIME	La fecha y hora del reloj de pared UTC de acuerdo con el compañero que eliminó la relación. Es NULL si la relación está viva.
_Relationship	Instancia CLR de una Relación	Éste es el objeto de relación para una relación viva. Es NULL para una relación con lápida.
TargetItemReference	ItemReference	Identidad del punto final objetivo

(4) Limpieza de Lapidas

[0241] Para prevenir el crecimiento descontrolado de información de lápidas, el almacén de datos suministra una tarea de limpieza de lápidas. Esta tarea determina cuando una información de lápida se puede descartar, la tarea computa un ciclo en la versión crear/actualizar y luego elimina la información de la lápida descartando las versiones anteriores.

11. Ayudas API y Funciones

[0242] El mapeo Base también suministra un número de funciones de ayuda. Estas funciones se suministran para ayudar en operaciones comunes sobre el modelo de datos.

a) Función [System.Storage].GetItem

Regresa un objeto Item dado en ItemId

//

Item GetItem (ItemId ItemId)

b) Function [System.Storage].GetExtension

//regresa un objeto de extensión dado el ItemId y ExtensionId

//

Extensión GetExtension (ItemId ItemId, ExtensionId ExtensionId)

c) Function [System.Storage].GetRelationship

//regresa a un objeto de relación dado un ItemId y RelationshipId

//

Relación GetRelationship (ItemId ItemId, RelationshipId RelationshipId)

12. Metadatos

[0243] Hay dos tipos de meta datos representados en el almacén: instancias de meta datos (el tipo de un Item, etc.) y tipos de meta datos.

a) Esquema de Metadatos

[0244] Esquemas de metadatos se almacenan en el almacén de datos como instancias de tipos de Item del esquema Meta.

b) Instancias de meta datos

[0245] Las instancias de meta datos las usa una aplicación para consultar el tipo de un Item y encontrar las extensiones asociadas con el Item. Dado el ItemID para un Item, una aplicación puede consultar la vista global de items para regresar el tipo de Item y utilizar

este valor para consultar la vista de Meta.Tipo y regresar información en el tipo declarado del Item. Por ejemplo,

```
//regresa metadatos del objeto Item para la instancia de un item
//
SELECT m._Item AS metadataInfoObj
FROM [System.Storage].[Item] i INNER JOIN [Meta].[Type] m ON i._TypeId = m.ItemId
WHERE i.ItemId = ]ItemId
```

G. Seguridad

[0246] En general, todos los objetos que se pueden asegurar disponen esos derechos de acceso utilizando un formato de máscara que se muestra en la figura 26. En este formato, los 16 bits de bajo orden son para derechos de acceso específicos a los objetos, los siguientes 7 bits son para derechos de acceso estándar, que aplican a la mayoría de tipos de objetos, y los cuatro bits de alto orden se usan para especificar derechos de acceso genéricos que cada tipo de objeto pueden mapea a un conjunto estándar de derechos específicos a los objetos. El bit ACCESS_SYSTEM_SECURITY corresponde a los derechos de acceso del objeto SACL.

[0247] En la estructura de la máscara de acceso de la figura 26, derechos específicos a un item se colocan en la sección Object Specific Rights (16 bits de bajo orden). Porque la presente implementación, la plataforma de almacenaje expone dos conjuntos de APIs para administrar la seguridad -Win32 y la plataforma de almacenaje API, los derechos específicos del objeto del sistema de archivos se deben considerar para motivar el diseño de derechos específicos a las plataformas de almacenaje.

[0248] Ningún el modelo de seguridad para la plataforma de almacenaje del presente invento se describe totalmente en la aplicación relacionada y que se incorporó por referencia anteriormente. En este respecto, la figura 27 (parte a, b yc) muestran un una región nueva con protección de seguridad idéntica que se ha sacado de una región de seguridad existente, de acuerdo con una implementación del modelo de seguridad.

H. Notificaciones y Seguimiento de Cambios

[0249] De acuerdo con otros aspectos del presente invento, la plataforma de almacenaje suministra capacidades de notificación que le permite a las aplicaciones seguir los cambios en los datos. Éste rasgo está dirigido más que todo a las aplicaciones que mantienen un estado volátil o ejecutan lógica de negocios en eventos de cambios de datos. Las aplicaciones se registran para notificación de items, extensiones de items y relaciones de items. Las notificaciones se reparten en forma no sincronizada luego de que se hayan hecho cambios en los datos. Las aplicaciones pueden filtrar las notificaciones por item, extensión y relación como también por el tipo de operación.

[0250] De acuerdo con una implementación, la plataforma de almacenaje API322 suministra a dos clases de interfaces para las notificaciones. Primero las aplicaciones se registran para datos simples de cambio de eventos disparados por cambios a un item, extensiones de item correlaciones de items. Segundo, las aplicaciones crean un objeto "observador" que monitorea conjuntos de items, extensiones de items y relaciones entre items. El estado del objeto observador se puede guardar y recrear luego de una falla en el sistema o después de que el sistema haya salido de línea por un período extendido de tiempo. Una notificación única puede reflejar múltiples actualizaciones.

[0251] Detalles adicionales en cuanto a esta funcionalidad se pueden encontrar en la aplicación relacionada que se incorpora cada por referencia.

I. Sincronización

[0252] De acuerdo con otros aspectos del presente invento, la plataforma de almacenaje suministra un servicio de sincronización 330 K (i) permite instancias múltiples de la plataforma de almacenaje (cada una con su propio almacén de datos 302) para sincronizada partes de su contenido de acuerdo a un conjunto flexible de reglas, hizo (ii) suministra una infraestructura para que terceros cinturones en el almacén de datos de la plataforma de almacenaje del presente invento con otros fuentes de datos que implementan protocolos de propiedad.

[0253] La sincronización de una plataforma de almacenaje con otra plataforma de almacenaje ocurre entre un grupo de réplicas participantes. Por ejemplo, en referencia a la figura 3, se puede desear la sincronización entre el almacén de datos 302 y la plataforma de almacenaje 300 con otro almacén de datos remoto 338 bajo el control de otra instancia de la plataforma de almacenaje, que podrían estar corriendo en un sistema de computación distinto. La membresía total de este grupo no necesariamente la conoce una replica dada en un momento dado.

[0254] Réplicas distintas pueden hacer cambios independientes (por ejemplo en forma concurrente). El proceso de sincronización se define haciendo que cada réplicas sea consciente de los cambios hechos en otros réplicas. Esta capacidad de sincronización es inherente al multi máster.

[0255] la capacidad de sincronización del presente invento le permite a las réplicas:

- Determinar de que cambios otra réplica puede ser consciente;
- Solicitar información acerca de los cambios de los cuales estas replica no es consciente;
- Suministrar información acerca de cambios sobre los cuales la otra réplica no es consciente;
- Determinar cuando dos cambios están en conflicto el uno con el otro;
- Aplicar cambios en forma local;

- Suministrar resolución de conflictos a otras réplicas para asegurar la convergencia; y
- Resolver conflictos pasados en políticas específicas para la resolución de conflictos.

1. Sincronización de una plataforma de almacenaje con otra plataforma de almacenaje

[0256] La aplicación primaria del servicio de sincronización 330 de la plataforma de almacenaje del presente invento de sincronizada las instancias múltiples de la plataforma de almacenaje (cada una con su propio almacén de datos). El servicio de sincronización opera en el nivel del esquema de la plataforma de almacenaje (envés de las tablas implícitas en el motor de la base de datos 314).

[257] El servicio de sincronización opera bajo el principio de "net changes". En vez de gravar y enviar operaciones individuales (como ocurre con las réplicas transaccionales), el servicio de sincronización envía el resultado final de esas operaciones, consolidando frecuentemente el resultado de operaciones múltiples en un resultado de cambio único.

[0258] El servicio de sincronización no respeta en general las fronteras de transacción. En otras palabras, si los cambios se hacen en los datos de una plataforma de almacenaje en una transacción única, no hay garantías de que estos cambios se apliquen a las otras réplicas en forma atómica -una puede aparecer sin la otra. La excepción a este principio es que si los cambios se hacen a un mismo Item en la misma transacción, se garantiza que estos cambios se envían y se aplican a las otras réplicas en forma atómica. Por lo tanto, los Items son las unidades de consistencia en un servicio de sincronización.

a) Aplicaciones de Control de Sincronización (Sync)

[0259] Cualquier aplicación se puede conectar al servicio de sincronización e iniciar una operación sync. Tal aplicación suministra todo los parámetros que se necesitan para ejecutar las sincronización (ver la reseña sync más abajo). A estas aplicaciones se les llamara acá Sync Controling Applications (SCAs).

[0260] Cuando se sincronizan instancias de dos plataformas de almacenaje, sync se inicia en un lado por un SCA. El SCA le informa al servicio local de sincronización que se

sincronize con el compañero remoto. Por el otro lado, el servicio de sincronización se despierta con el mensaje enviado por el servicio de sincronización de la máquina que origina. Responde basado en la información de configuración persistente (ver mapeo más abajo) presente en la máquina de destino. El servicio de sincronización puede correr en un horario o en respuesta a distintos eventos. En este caso, el servicio de sincronización implementando el horario se convierte en uno SCA.

[0261] Para permitir la sincronización dos pasos se tienen que tomar. Primero el diseñador del esquema debe anotar el esquema de la plataforma de almacenaje con la semántica sync (designando Change Units como se describe más abajo). Segundo la sincronización se debe configurar en forma apropiada en todas las máquinas que tienen una instancia de la plataforma de almacenaje que va a participar en la sincronización (como se describe más adelante).

b) Esquema de Anotación

[0262] Un concepto fundamental del servicio de sincronización es el de un Change Unit. Un Change Unit es la pieza más pequeña de un esquema al que la plataforma de almacenaje le hace seguimiento. Para cada Change Unit, el servicio de sincronización es capaz de determinar si cambió o no cambió desde la última sync.

[0263] Designar Change Units en el esquema sirve varios propósitos. Primero, determina qué tan inexacto es el servicio de sincronización en el cable. Cuando un cambio se hace dentro de un Change Units, el Change Units completo se envía a otras réplicas, puesto que el servicio de sincronización no saben que parte del Change Units se cambio. Segundo, determina que tan granular es la detección del conflicto. Cuando se hacen dos cambios concurrentes (estos términos se definen en más detalle en secciones subsiguientes) a la misma unidad de cambio, el servicio de sincronización detecta un conflicto; de la otra manera, si un cambio concurrente se hace a diferentes unidades de cambio, entonces ningún conflicto se reporta y los cambios automáticamente se fusionan. Tercero afecta fuertemente la cantidad de metadatos que el sistema mantiene. Muchos de los metadatos del servicio de sincronización se mantienen por cada Change Units; y así, haciendo Change Units más pequeños aumentan la capacidad sobrante de sync.

[0264] Definir los Change Units requiere encontrar un intercambio adecuado. Por esta razón, el servicio de sincronización le permite a los diseñadores de esquema participar en el proceso.

[0265] En una implementación, el servicio de sincronización no apoya Change Units más grandes que un elemento. Sin embargo, si apoya la habilidad para que los diseñadores de esquema especifiquen cambios y unidades más pequeños que un elemento -como, agrupar múltiples atributos de un elemento en un Change Units separado. En ése implementación, esto se logra utilizando la siguiente sintaxis:

```
<Type Name="Appointment" MajorVersion="1" MinorVersion="0"
ExtendsType="Base.Item"                    ExtendsVersion="1">
   <Field Name="MeetingStatus“ Type="the storage platformTypes.uniqueidentifier
Nullable="False"/>
   <Field Name="OrganizerName“ Type="the storage platformTypes.nvarchar(512)"
Nullable="False"/>
   <Field Name="OrganizerEmail“ Type="the storage platformTypes.nvarchar(512)"
TypeMajorVersion="1“                    MultiValued="True"/>
    ...
   <ChangeUnit Name="CU_Status">
       <Field Name="MeetingStatus"/>
   </ChangeUnit>

   <ChangeUnit Name="CU_Organizer"/>
       <Field Name="OrganizerName" />
       <Field Name="OrganizerEmail" />
  </ChangeUnit>
  ...

</Type>
```

C) Configuración Sync

[0266] Un grupo de compañeros de la plataforma de almacenaje que quieran mantener ciertas partes de sus datos en sync son llamados una comunidadsync. Y mientras los miembros de una comunidad quieran estar en sync, no necesariamente representan los datos en exactamente la misma manera; en otras palabras, los compañeros sync pueden transformar los datos que están sincronizando.

[0267] En un escenario colega a colega, es impráctico para los colegas mantener el mapeo de transformaciones para todos sus compañeros. En cambio, el servicio de sincronización retoma el camino de definir "Folderes Comunitarios". Un folderes comunitario es una abstracción que representa a "folderes compartidos" hipotéticos con el que todos los miembros de la comunidad se deben sincronizar.

[0268] Esta noción se ilustra mejor con el siguiente ejemplo. Si Joe quiere mantener el folder Mis Documentos de sus varios computadores en sync, Joe define un folderes comunitario llamado, por ejemplo, JoeDocumentos. Entonces, en cada computador, Joe configura un mapeo entre el folder hipotético JoeDocumento y el folder local Mis Documentos. A partir de este punto, en donde los computadores de Joe se sincronizan el uno con el otro, ellos hablan en términos de documentos en JoeDocumentos, envés de hablar de sus items locales. De esta manera, todos los computadores de Joe se entienden el uno con el otro sin tener que saber quién es el otro -el Folder Comunitario se convierte en la lengua franca de la comunidad sync.

[0269] Configurar el servicio de sincronización consiste de tres pasos: (1) definir el mapeo entre folderes locales y folderes comunitarios; (2) definir una reseña sync que determina que se sincroniza (por ejemplo, con quien sincronizarse y que subconjunto se debe enviar y quien lo debe recibir); y (3) definir un horario en el cual distintas reseñas sync deben correr, o correrlas manualmente.

(1) Folder Comunitario -Mapeos

[0270] Los mapeos del Folder Comunitario se guardan en archivos de configuración XML en las máquinas individuales. Cada mapeo tiene el siguiente esquema:

/mappings/communityFolder

éste elemento nombra el folder comunitario para el cual este mapeo es. El nombre sigue las reglas de sintaxis de los folderes.

/mappings/localFolder

éste elemento nombra el folder local en el cual el mapeo se transforma. El nombre sigue las reglas de sintaxis de los Folderes. El folder debe existir para que el mapeo sea válido. Los items dentro de este folder se consideran para la sincronización de este mapeo.

/mappings/transformations

éste elemento define como transformar los items del folder comunitario al folder local y viceversa. Si está ausente o desocupado, ninguna transformación se ejecuta. En particular, esto significa que ningún ID se mapea. Esta configuración es útil principalmente para crear una memoria temporal del Folder.

/mappings/transformations/mapIDs

éste elemento solicita que un ID local recién generado se asigne a todos los items mapeados del folder comunitario, envés de re usar los IDs comunitarios. El Runtime Sync mantendrá el mapeo de IDs para convertir los items en ambas direcciones.

/mappings/transformations/localRoot

éste elemento solicita que todos los items raíz en el folder comunitario se hagan hijos de la raíz especificada.

/mappings/runAs

éste elemento controla bajo qué autoridad el mapeo solicitado se debe procesar.

/mappings/runAs/sender

la presencia de éste elemento indica que quien envía el mensaje a este mapeo debe estar representado, y las solicitudes procesadoras bajo sus credenciales.

(2) Perfiles

[0271] Un Perfiles Sync. Es un conjunto total de parámetros que se necesitan para arrancar la sincronización. Lo suministra un SCA al SYNC Runtime para iniciar el sync. El perfiles sync de sincronización para plataforma de almacenaje a plataforma de almacenaje contiene la siguiente información:

- Folder local, debe servir como fuente y destino por los cambios;

- El nombre del Folder Remoto con el que se debe sincronizar espacio-este Folder se debe publicar del compañero remoto a través de una mapeo tal como se definió anteriormente;
- Dirección -el servicio de sincronización es compatible con enviar-únicamente, recibir-únicamente, y enviar-recibir sync;
- Filtro Local -se presiona que información local se debe enviar al compañero remoto.
- Expresada como una consulta a la plataforma de almacenaje sobre el folder local;
- Filtro Remoto -selecciona la información remota que va recuperar del compañero remoto- expresada como una consulta a la plataforma de almacenaje sobre el folder comunitario;
- Transformaciones espacio-define como se transforman los items de y para el formato local;
- Seguridad local -especifica si los cambios recuperados de un punto final remoto debe ser aplicados bajo el permiso del punto final remoto (representado) o el usuario que inicia el sync localmente; y.
- Política de resolución de conflictos -enero último caso, se especificaba resolución del conflicto utilizar, como también los parámetros de configuración para él.

[0272] El servicio de sincronización suministra una clase runtime CLR que permiten la construcción simple de SyncProfiles. Los perfiles también se pueden serializar de y para un archivo XML para facilidad en el almacenaje (frecuentemente junto con el horario). Sin embargo, no hay un sitio estándar en la plataforma de almacenaje donde todos los perfiles se guarden; los SCAs son bienvenidos a que construyan perfiles inmediatos sin persistir. Notese que no hay necesidad de tener un mapeo local para iniciar el sync. Toda la información del sync se puede especificar en el perfiles. El mapeo es sin embargo, requerido para responder a la solicitud de sync solicitada e iniciada del lado remoto.

(3) Programación de Tareas

[0273] En una implementación, el servicio de sincronización no suministra su propia infraestructura de programación de tareas. Por el contrario, depende de otro componente para ejecutar ésta tarea -el Windows Scheduler disponible con el sistema operativo Microsoft Windows. El servicio de sincronización incluye una utilidad de comando de línea que actúan sobre un SCA para disparar una sincronización basada en un perfil de sync guardado como un archivo XML. Esta utilidad hace que la configuración del Windows Scheduler sea muy fácil para ejecutar una sincronización ya sea en un horario predefinido o en respuesta a un evento como prender o pagar.

d) Manejo de Conflictos

[0274] El manejo de conflictos en el servicio de sincronización se divide en tres etapas (1) detección del conflicto, que ocurre en el momento de cambiar aplicación -este paso determina si un cambio se puede aplicar en forma segura; (2) resolución y ejecución automática de conflictos -durante esta etapa (que sucede inmediatamente después de que un conflicto se detecta) resolutores automáticos de conflicto se consultan para ver si el conflicto se puede resolver- si no, el conflicto puede opcionalmente quedar en espera; y (3) resolución e inspección de conflictos -esta etapa se ejecuta si algunos conflictos están en espera, y ocurren fuera del contexto de la sesión sync -en este momento, los conflictos guardados se pueden resolver y remover.

(1) Detección de Conflictos

[0275] En la presente implementación, el servicio de sincronización detecta dos tipos de conflictos: basados en conocimiento y basados en constreñimientos.

(a) Conflictos basados en Conocimiento

[0276] Un conflicto basado en conocimiento ocurre cuando dos réplicas hacen cambios a la misma Change Units. Dos cambios se llaman independientes si se hacen sin conocimiento el uno del otro -en otras palabras, la versión del primero no está cubierta por el conocimiento de la 2ª y viceversa. El servicio de sincronización automáticamente detecta estos conflictos basados en el conocimiento de las réplicas tal como se describe arriba.

[0277] A veces ayuda pensar en los conflictos como bifurcaciones en la versión de historia de un Change Unit. Si no hay conflictos en la vida de un Change Unit, su versión de historia es una cadena simple -cada cambio que ocurre luego del anterior. En el caso de un conflicto basado en conocimiento, dos cambios ocurren en paralelo, causando la bifurcación de la cadena y convertirse en un árbol.

(b) Conflictos Basados en Constreñimientos

[0278] Hay casos donde cambios independientes violan la integridad de los constreñimientos cuando se aplican en conjunto. Por ejemplo, dos réplicas creando un archivo con el mismo nombre en el mismo directorio causarían este tipo de conflicto.

[0279] Un conflicto basado en constreñimientos involucra dos cambios independientes (al igual que un conflicto basado en conocimiento), pero ellos no afectan a la misma Change Unit. Más bien, afectan diferentes Change Units pero con un constreñimiento que existe entre los dos.

[0280] El servicio de sincronización detecta una violación de constreñimiento cuando se efectúan cambios de aplicación y anuncia automáticamente conflictos basados en constreñimientos. Resolver conflictos basados en constreñimiento generalmente requiere un código que modifica el cambio de tal manera que no viole el constreñimiento; el servicio de sincronización no suministra un mecanismo general para hacerlo.

(2) Procesamiento de Conflictos

[0281] Cuando se detecta el conflicto, el servicio de sincronización puede tomar una de tres acciones (seleccionada por el iniciador de sync en Perfiles de Sync): (1) rechazar el cambio, enviarlo de vuelta al que lo origina; (2) dejar al conflicto en espera en una lista de espera; o (3) resolver el conflicto automáticamente.

[0282] Si el conflicto se rechaza, el servicio de sincronización actúa como si el conflicto no hubiera llegado a la replica. Un reconocimiento negativo se envía de vuelta al originario. Esta política de resolución es útil en réplicas sin encabezado (tales como archivos de servidor) en donde dejar los conflictos en espera no es posible. En cambio, estas réplicas obligan a las otras a entenderse con el conflicto rechazándolo.

[0283] Los iniciadores de Sync configuran la resolución de conflictos en sus Perfiles de Sync. El servicio de sincronización apoya la combinación de múltiples resoluciones de conflictos ensayando una después de la otra, hasta que una de ellas tiene éxito; y segundo, asociando las resoluciones de conflicto con tipos de conflicto, por ejemplo, dirigiendo conflictos basados en actualización reconocimiento a una resolución, y los demás conflictos a la lista de espera.

(a) Resolución Automática de Conflictos

[0284] El Servicio de sincronización suministra un número de resoluciones por defecto. Esta lista incluye:

- wins locales: hace caso omiso de cambios si entran en conflicto con datos almacenados localmente;
- wins remoto: hace caso omiso de los datos locales si entran en conflicto con cambios que entran;
- wins de última escritura: escoge ya sea al win local o al win remoto para cada unidad de cambio basada en el sello de tiempo del cambio (no 13 que el servicio de sincronización en general no depende de valores del reloj; esta resolución de conflicto es la única excepción a esa reglas);
- Determística: escoge un ganador de tal forma que se garantiza que es igual en todo lo réplicas, y de lo contrario no es significativo -una implementación del servicio de sincronización utiliza comparaciones léxico-gráficas de IDs compañeros para implementar este rasgo.

[0285] Tradicionalmente, el ISV puede instalar sus propias resoluciones de conflicto. Resoluciones de conflicto puede aceptar la configuración de parámetros; estos parámetros deben estar especificados en el SCA de la sección de Resolución de Conflictos del Perfile Sync.

[0286] Cuando una resolución de conflictos maneja un conflicto, devuelve la lista de operaciones que necesite ejecutarce (en vista de que podía haber cambios que entran en conflicto) devuelta al runtime. El servicio de sincronización luego aplica estas operaciones,

habiendo ajustado el conocimiento remoto en forma apropiada para incluir lo que el administrador del conflicto considera.

[0287] Es posible que otro conflicto se detecte mientras se aplica la resolución. En tal caso, todo conflicto se debe resolver antes de que el proceso se reasuma.

[0288] Cuando se piensa en los conflictos como ramas en la versión de la historia de un items, la resolución del conflicto se puede ver como juntas -combinando los ramas para formar un punto único. De esta manera, la resolución del conflicto torna la versión de las historias en un DAG.

(b) Conflictos En Espera

[0289] Una clase muy particular de resolución de conflictos es el conflict Logger. El servicio de sincronización guarda los conflictos en espera como Items del tipo ConflictRecord. Estos recursos se relacionan de nuevo con los items que están en conflicto (al menos de que los items se hayan eliminado). Cada récord de conflictos contiene: el cambio entrante que ha causado el conflicto; el tipo de conflicto: actualización-actualización, actualización-eliminar, eliminar-actualización, insertar-insertar o constreñimiento; y la versión del cambio entrante y el conocimiento de la réplica que envía. Conflictos en espera está disponible para inspección y resolución como se describe a continuación.

(c) Inspección y Resolución de Conflictos

[0290] El Servicios de sincronización suministra una API para que las aplicaciones examinen la lista de conflictos en espera y sugieran resoluciones al conflicto. La API permiten a las aplicaciones enumerar todos los conflictos, o conflictos relacionados a un Item dado. También permite que estas aplicaciones resuelvan conflictos en espera en una de tres maneras: (1)wins remotos -aceptar los cambios en espera y sobreescribir los cambios locales; (2)wins locales -ignorar las partes en conflicto de los cambios en espera; y (3) sugerir un nuevo cambio -donde la aplicación propone la fusión que, en su opinión, resuelve el conflicto. Una vez los conflictos los resuelve la aplicación, el servicio de sincronización lo remueve de la lista en espera.

(d) Convergencia de réplicas y Propagación de Resolución de Conflictos

[0291] En escenarios complejos de sincronización, el mismo conflicto se puede detectar en réplicas múltiples. Si esto ocurre, varias cosas pueden suceder: (1) el conflicto se puede resolver en una réplica, y la resolución se puede enviar a otra; (2) el conflicto se resuelve en ambas réplicas automáticamente; o (3) el conflicto se resuelve en ambas réplicas manualmente (a través de una inspección de conflicto API).

[0292] Para asegurar la convergencia, el servicio de sincronización envía una resolución de conflicto a las otras réplicas. Cuando un cambio resuelve el conflicto que llega de una réplica, el servicio de sincronización automáticamente encuentra un record de conflicto en la lista de espera que se resuelve con esta actualización y los elimina. En este sentido una resolución de conflicto en una réplica está atada a todas las demás réplicas.

[0293] Si distintos ganadores se escogen por diferentes réplicas para el mismo conflicto, el servicio de sincronización aplica el principio de la resolución de conflictos atada si escoge una de las dos resoluciones que ganan sobre las otras automáticamente. El ganador se escoge en una forma determinista que garantiza que se produce el mismo resultado todas las veces (una implementación utiliza comparaciones ID léxicograficas de la réplica).

[0294] Si "nuevos cambios" diferentes se sugieren por distintas réplicas para el mismo conflicto, el servicio de sincronización trata a este nuevo conflicto como un conflicto especial y usa el Conflict Logger para prevenir que no se propague a otras réplicas. Tal situación comúnmente surge con la resolución de conflicto manual.

2. Sincronizando a una plataforma de almacén de datos que no almacena

[0295] De acuerdo con otro aspecto de la plataforma almacenaje del presente invento, la plataforma de almacenaje provee una arquitectura ISVs para implementar Adaptadores Sync se le permiten a la plataforma de almacenaje sincronizarse con sistemas de legado como Microsoft Exchange, AD,Hotmail, etc. los adaptadores Sync se benefician de los muchos Servicios Sync que el servicio de sincronización provee, como se describe más adelante.

[0296] A pesar del nombre, los Adaptadores Sync no necesitan ser implementados como plug-ins en alguna arquitectura de plataforma de almacenaje. Si se desea, un "adaptador sync" puede simplemente ser una aplicación cualquiera que utiliza la interfase runtime del servicio de sincronización para obtener servicios tales como el cambio de numeración y aplicación.

[0297] Para hacerlo más sencillo a otros de configurar y correr una sincronización con un compañero dado, se estimulan a los escritores de Adaptador Sync para que expongan la interfase estándar del Adaptador Sync, que corren la sync dado el Perfil Sync, tal como se describió más arriba. El perfil nos da la información de configuración para el adaptador, algunos de los cuales pasan el control al Sync Runtime para que asuma el control del servicio (por ejemplo, el Folder a sincronizar).

a) El Servicio Sync

[0298] El servicio de sincronización provee un número de servicios sync para escritores de adaptadores. Por el resto de esta sección, es conveniente referirnos a la máquina en la cual la plataforma almacenaje estaba efectuando la sincronización como el "cliente" y la plataforma de no-almacenaje con la que el adaptador está hablando como el "servidor".

(1) Enumeración de Cambios

[0299] Basado en los datos de seguimiento de cambios mantenido por el servicio de sincronización, la Enumeración de Cambios permite al adaptador sync enumerar fácilmente los cambios que han ocurrido en el Folder del almacén de datos desde la última vez que se sincronizo con este compañero.

[0300] La enumeración de cambios basado en el concepto de "ancla" -una estructura opaca que representa información acerca de la última sincronización. El ancla toma la forma de una plataforma de almacenaje de Conocimiento, como se describió en las secciones anteriores. Los adaptadores Sync que utilizan el servicio de la enumeración de cambios caen en dos categorías amplias: aquellos que utilizan "anclas guardadas" vs. aquellos que utilizan "anclas suministradas".

[0301] La distinción esta basada en el sitio donde se almacena a la información de la últimasync -en el cliente, o en el servidor. A veces es más fácil para los adaptadores

almacenar esta información en el cliente –el colega frecuentemente no es capaz de guardar ésta información en forma conveniente. De la otra manera, si clientes múltiples se sincronizan con el mismo colega, almacenar esta información en el cliente es ineficiente en algunos casos o incorrecto -hace que un cliente no se dé cuenta de los cambios que el otro cliente ya le ha enviado al servidor. Si un adaptador quiere utilizar anclas guardadas en el servidor, el adaptador necesita suministrarla de vuelta a la plataforma de almacenaje a la hora de la enumeración de cambios.

[0302] Para que la plataforma de almacenaje mantenga el ancla (ya sea para almacenaje local o remoto), la plataforma de almacenaje necesita conocer los cambios que se aplicaron exitosamente en el servidor. Estos y únicamente estos cambios se pueden incluir en el ancla. Durante la enumeración de cambios, el adaptador Sync utiliza la interfase de reconocimiento para reportar que cambios se aplicaron exitosamente. Al final de la sincronización, el adaptador utilizando las anclas suministradas debe leer la nueva ancla (que incorpora todos los cambios efectuados exitosamente) y enviarla a sus colegas.

[0303] Frecuentemente, los Adaptadores necesitan guardar datos específicos a los adaptadores junto con items que han insertado en el almacén de datos de la plataforma de almacenaje. Ejemplos comunes tales como datos remotos IDs o versiones remotas (timestamps). El servicio de sincronización suministra un mecanismo para almacenar estos datos, Enumeración de Cambios provee un mecanismo para recibir estos datos extras junto con los cambios que se devuelven. Esto elimina la necesidad de que los adaptadores, en la mayoría de casos, vuelvan a consultar la base de datos.

(2) Cambio de Aplicación

[0304] El Cambio de Aplicación le permite al Adaptador Sync aplicar los cambios recibidos de su compañero a la plataforma de almacenaje local. Los adaptadores deben transformar los cambios en el esquema de la plataforma de almacenaje. La figura 24 muestra el proceso mediante el cual las clases en la plataforma de almacenaje API se generan del esquema de la plataforma de almacenaje.

[0305] La función primaria del cambio de aplicación es detectar automáticamente los conflictos. Como es el caso de una sync entre alguna plataforma de almacenaje y otra

plataforma de almacenaje, un conflicto se define como dos cambios superpuestos que se hacen sin el conocimiento el uno del otro. Cuando el adaptador usa Cambios de Aplicación, debe especificar el ancla con respecto al cual la detección de conflictos se ha ejecutado. Cambio de Aplicación manifiesta un conflicto si un cambio local superpuestos del cual el adaptador no tiene conocimiento se detecta. Al igual que con Enumeración de Cambios, el adaptador puede utilizar cualquiera de las anclas almacenadas. Cambio de Aplicación es compatible con el almacenaje de metadatos específicos al adaptador. Estos datos se pueden anexar a los cambios que se están aplicando por el adaptador, y se pueden guardar con el servicio de sincronización. Los datos se pueden regresar en una próxima enumeración de cambios.

(3) Resolución de Conflictos

[0306] El mecanismo de Resolución de Conflictos descritos anteriormente (dejar en espera y resolución automática) están disponibles también a los adaptadores sync. Los adaptadores sync pueden especificar la política para la resolución de conflictos cuando aplican los cambios. Si se especifica, el conflicto puede pasar al administrador de conflictos específico y ser resuelto (si es posible). Los conflictos también se pueden dejar en espera. Es posible que el adaptador detecte un conflicto cuando trata de aplicar cambios locales en el colega. En tal caso, el adaptador todavía puede pasar el conflicto al Runtime Sync para qué lo resuelva de acuerdo con la política. Adicionalmente los Adaptadores Sync pueden solicitar que cualquier conflicto detectado por el servicio de sincronización sea devuelto para procesamiento. Esto es particularmente conveniente en el caso donde el colega es capaz de guardar o resolver los conflictos.

b) Implementación de Adaptadores

[0307] Mientras algunos "adaptadores" son simplemente aplicaciones que utilizan interfases runtime, los adaptadores se estimulan a que implementen una interfase estándar de adaptador. Estas interfases le permiten a las Aplicaciones Sync Controlling a: solicitar que el adaptador ejecute la sincronización de acuerdo con un Perfil Sync dado; cancelar una sincronización en curso; y recibir un reporte del progreso (porcentaje que se completa) en una sync en curso.

3. Seguridad

[0308] El servicio de sincronización briega por introducir lo menos posible en el modelo de seguridad implementado por la plataforma de almacenaje. Más bien que definir nuevos derechos para la sincronización, los derechos existentes se utilizan. Específicamente,

- Cualquiera que pueda leer un item de un almacén de datos puede enumerar los cambios hechos a ese item;
- Cualquiera que pueda escribir un item en un almacén de datos puede aplicar cambios a ese item; y
- Cualquiera que pueda extender un item de un almacén de datos puede asociar metadatos sync con ese item.

[0309] El servicio de sincronización no mantiene información segura de derechos de autor. Cuando un cambio se hace en la replica A por un usuario U y envía a la replica B, el hecho de que el cambio originalmente se hizo en A (o por U) se pierde. Si B envía éste cambio a la replica C, esto se hace bajo la autoridad de B, no la de A. esto conlleva la siguiente limitación: si no se confía que la replica haga los cambios a su propio item, no puede enviar los cambios hechos por otros.

[0310] Cuando el servicio de sincronización se inicia, lo hace con una Aplicación Sync Controlling. El servicio de sincronización toma la identidad del SCA y ejecuta todas las operaciones (tanto locales como remotas) bajo esa identidad. Para ilustrar, observen que el usuario U no puede hacer que el servicio local de sincronización recupere cambios de una plataforma de almacenaje remota para items a los cuales el usuario U no tiene acceso de lectura.

4. Manejabilidad

[0311] Monitorear una comunidad distribuida de réplicas es un problema complejo. El servicio de sincronización puede utilizar un algoritmo "sweep" para recoger y distribuir información acerca del estatus de las réplicas. La propiedad a del algoritmo sweep asegura que la información de todas las réplicas configuradas se recoge eventualmente y que las réplicas que fallan (no responde) se detectan.

[0312] Este monitoreo de información a través de toda la comunidad se hace posible en cada réplica. Las herramientas de monitoreo se pueden correr en una réplica cualquiera arbitraria para examinar la información de monitoreo y tomar decisiones administrativas. Cualquier cambio de configuración se debe hacer directamente en las réplicas afectadas.

J. LA INTEROPERABILIDAD DEL SISTEMA DE ARCHIVO TRADICIONAL

[0313] Cómo se mencionó anteriormente, se pretende que la plataforma de almacenaje del presente invento sea, en algunas implementaciones, parte integral del sistema de interfase de hardware/software del sistema de computación. Por ejemplo, la plataforma de almacenaje del presente invento puede estar implementada como una parte integral del sistema operativo, tal como la familia de sistemas operativos Microsoft Windows. En esa capacidad, la plataforma de almacenaje API se convierte en parte del sistema operativo API a través del cual en los programas de aplicación interactúan con el sistema operativo. Así, la plataforma de almacenaje se convierte en el medio a través del cual los programas de aplicación guardan información en el sistema operativo, y el modelo de datos basado en items de la plataforma de almacenaje por lo tanto reemplaza el sistema de archivos tradicional de dicho sistema operativo. Por ejemplo, tal como se implementa en la familia de sistemas operativos de Microsoft Windows, la plataforma de almacenaje puede reemplazar a el sistema de archivos NTFS en el sistema operativo. En el presente, los programas de aplicación accesan el servicio del sistema de archivos NTFS a través del API Win32 expuesto por la familia de sistemas operativos Windows.

[0314] Hay que reconocer, sin embargo, que reemplazar completamente el sistema de archivos NTFS con la plataforma de almacenaje del presente invento requeriría la recodificación de los programas de aplicación existentes basados en Win32 y que esa recodificación puede ser indeseable, sería benéfico para la plataforma de almacenaje del presente invento suministrar alguna interoperabilidad con los sistemas de archivo existentes, tal como el NTFS. En una implementacion del presente invento, por lo tanto, la plataforma de almacenaje permite que los programas de aplicación que dependen del

modelo de programación Win32 tengan acceso al contenido de ambos, el almacén de datos en la plataforma de almacenaje como también al sistema de archivos tradicional NTFS. Para este propósito, la plataforma de almacenaje utiliza una convención de nombres que es un súper set de la convención de nombres Win32 para facilitar la interoperabilidad. Aún más, la plataforma de almacenaje apoya la búsqueda de archivos y directorios almacenados en la plataforma de almacenaje a través del API Win32.

[0315] detalles adicionales en cuanto esta funcionalidad se pueden encontrar en las aplicaciones relacionadas que se han incorporado aca por referencia.

K. API DE LA PLATAFORMA DE ALMACENAJE

[0316] La Plataforma de almacenaje comprende una API que permite a los programas de aplicación a acceder a los rasgos y capacidades de la plataforma de almacenaje discutidas anteriormente y acceder a los items guardados en el almacén de datos esta sección describe una implementación de la plataforma de almacenaje API de la plataforma de almacenaje del presente invento. Detalles de esta funcionalidad se pueden encontrar en las aplicaciones relacionadas y que se incorporan aca por referencia, con algo de esta información resumida más adelante por conveniencia.

[0317] En referencia a la figura 18, un folder de Contención es un items que contiene relaciones de pertenencia con otros Items es el equivalente del concepto de un folder en un sistema de archivos. Cada Items esta "contenido" dentro de por lo menos un folder de contención.

[0318] La figura 19 muestra la arquitectura básica de la plataforma de almacenaje API, de acuerdo con la presente implementación. La plataforma de almacenaje API utiliza un SQLClient 1900 para hablar con el almacén de datos local 302 y también puede utilizar un SQLClient 1900 para hablar con el almacén de datos remota (por ejemplo, el almacén de datos 340). El almacén local 302 también puede hablar con el almacén de datos remoto 340 utilizando ya sea DQP (Distributed Query Processor) o través del servicio de sincronización de la plataforma de almacenaje ("Sync") descrito más arriba. La plataforma de almacenaje API 322 también actúa como un puente API para almacenar la notificación de datos,

pasando la suscripción de las aplicaciones al motor de notificación 322 y guiando a las notificaciones a la aplicación (por ejemplo aplicación 350a, 350b, o 350c), como también se describió más arriba. En una implementación, la plataforma de almacenaje API 322 también puede definir una arquitectura limitada "proveedor" de tal manera que puede acceder datos en Microsoft Exchange y en AD.

[0319] La figura 20 representa esquemáticamente varios componentes de la plataforma de almacenaje API. La plataforma de almacenaje API consiste de los siguientes componentes: (1) clases de datos 2002, que representan elementos en la plataforma de almacenaje y tipos de item, (2) un marco runtime 2004, que maneja la persistencia de objetos y provee apoyo por las clases 2006; y (3) herramientas 2008, que se usan para generar clases CLR del esquema de la plataforma de almacenaje.

[0320] la jerarquía de clases que resulta de un esquema dado directamente refleja la jerarquía de tipos en ese esquema. Como un ejemplo, considere el tipo de Item definido en el esquema Contactos como se muestra en la figura 21A y figura 21B.

[0321]la figura 22 ilustra un marco runtime operando. El marco runtime opera en la siguiente manera:

1. Una aplicación 350 a, 350b o 350c se adhieren a un item tendrá plataforma de almacenaje.
2. El marco 2004 crea un objeto ItemContext 2202 que corresponde al item adherido y lo regresa al aplicación.
3. La aplicación envía una Find en el ItemContext para recoger una colección de Items; una colección de vuelta es conceptualmente un objeto gráfico 2204 (debido a la relación).
4. Los cambios de aplicación, eliminaciones, datos que se inserta.
5. La aplicación guardan los cambios llamando al método Update ().

[0322] es la figura 23 muestra la ejecución de una operación "FindAll".

[0323] la figura 24 muestra el proceso mediante el cual las clases en la plataforma de almacenaje API se generan del Esquema de la plataforma de almacenaje.

[0324] La figura 25 muestra un esquema en el cual el Archivo API está basado. La plataforma de almacenaje API incluye un namespace para tratar con los objetos archivos.

Este namespace se llama SystemStorage.Files. Los datos miembros de la clases en SystemStorage.Files directamente refleja la información almacenada en el almacén de la plataforma de almacenaje; esta información se "promueve" del sistema de archivos José puede crear en forma nativa utilizando el API Win32. El namespace del SystemStorage.Files tiene dos clases: FileItem y DirectoryItem. Los miembros de estos clases y métodos por lo tanto se pueden encontrar rápidamente mirando en el diagrama de esquema de la figura 25. FileItem y DirectoryItem son únicamente de lectura por la plataforma de almacenaje API. Para modificarlos, uno tiene que precisar la API Win32 o clases en el Sistema.IO.

[0325] En cuanto a la interfase de programación del API (o sencillamente, interfase) se puede ver como un mecanismo, proceso, protocolo para permitir que uno con más segmentos de código se podrá comunicar obtener acceso a la funcionalidad de uno con más segmentos de código. En forma alternativa, una interfase de programación se puede ver como un mecanismo(s), método(s), función(es), módulo(s), objeto(s) etc. del componente de un sistema capaz de comunicarce a uno o más mecanismo(s), método(s), función(es), módulo(s), objeto(s) etc. entre otros componentes. el termino "segmento de código" en el párrafo anterior tiene la intención de incluir una o más instrucciones por líneas de código, e incluye, por ejemplo, módulos de código, objetos, subrutinas, función es, y demás, sin importar qué terminología se aplique o si el segmento de código estar recopilado en forma separada, o si el segmento de código es un objeto fuente, intermedio u objetivo, así lo segmentos de código se utilicen en un sistema runtime o proceso, o si están localizados en la misma o diferente máquina o distribuidos a través de múltiples máquinas, o si la funcionalidad representada por lo segmentos de código está implementada en su totalidad en el software, en el hardware, o en una combinación de hardware y software.

[0326] normalmente, una interfase de programación se puede ver en forma genérica, como se muestra en la figura 30A o 30B. La figura 30A muestra en una interfase Interfase 1 como un conducto a través del cual el primer y segundo código de segmentos se comunican. La figura 30B muestra una interfase que comprende los objetos I1 y I2 (que pueden ser conocer parte del primero y segundo segmentos de código), que le permiten al primer y segundo segmento de código de un sistema comunicarse a través del medio M. En la figura

30B uno poder considerar los objetos de interfase I1 yI2 más el medio M comprende en la interfase. Aunque en la figura 30A y 30B muestra en un un flujo bidireccional interfases a cada lado del flujo, en ciertas implementaciones puede únicamente tener información del flujo en una dirección (o sin información del flujo como se describe más abajo) o puede únicamente tener un objeto de interfase en un lado. Como ejemplo y no como limitante, términos tales como interfase de programación de aplicaciones (API),. Entrada, método, función, subrutina, llamada de procedimiento remota, e interfase de componente de modelo de objeto (COM), están todos comprendidos dentro de toda definición de interfase de programación.

[0327] Aspectos de esta interfase de programación pueden incluir métodos mediante los cuales el primer segmento de código tranamite información (donde "información" se utiliza en el sentido más amplio e incluye datos, comandos, peticiones, etc.) al segundo segmento de código; el método mediante el cual el segundo segmento de código recuperar la información; y la estructura, secuencia, sintaxis, organización, esquema, horario y contenido de la información. En este respecto, el medio de transporte en sí puede no ser importante a la operación de la interfase, así el medio esté alambrado o será inalámbrico, o una combinación de ambos, siempre y cuando la información se transporte en una manera esté definida por la interfase. En ciertas situaciones, la información puede no pasar en una o en ambas direcciones en el sentido convencional, pues la transferencia de información puede ser a través de otro mecanismo (por ejemplo, información que se coloca en un buffer, archivo, etc. separada del flujo de información entre segmentos de códigos) o no existente, como cuando un segmento de código simplemente accede a una funcionalidad ejecutada por un segundo segmento de código. Cualquiera o todos estos aspectos pueden ser importantes en una situación dada, por ejemplo, dependiendo de si el segmento de código es parte del sistema en una configuración acoplada en forma ligera o apretada, y por lo tanto están listos se debe considerar ilustrativa y no limitante.

[0328] Esta noción de una interfase de programación es conocida a aquellos diestros en el arte y está clara con la descripción detallada que se da del invento. Hay, sin embargo, otras formas de implementar la interfase de programación, y al menos de que se excluya expresamente, ésta s también están comprendidas dentro de las reivindicaciones que se

elevan al final de esta especificación. Estas otras maneras pueden parecer más sofisticadas y complejas en la vista simplista de las figuras 30A y 30B, pero sin embargo ellas ejecutan funciones similares para lograr el mismo resultado general. Ahora describiremos en forma breve algunas implementaciones ilustrativas alternativas de la interfase de programación.

[0329] Factorización: una comunicación de un segmento de código a otro puede lograrse indirectamente rompiendo la comunicación en comunicaciones múltiples discretas. Esto se muestra esquemáticamente en la figura 31A y 31B. Como se muestra, algunas interfases se pueden describir en términos de conjuntos de funcionalidad divisibles. Así, la funcionalidad de la interfase de la figura 30A y 30B puede considerarse que obtenga el mismo resultado, así como uno puede matemáticamente proveer 24, o 2 veces 2 veces 3 veces 2. De la misma manera, como se ilustra en la figura 31A, la función provista por la interfase Interfase 1 se puede factorizarr para convertir la comunicación de la interfase en múltiples interfases Interfase 1A, Interfase 1B, Interfase 1C, etc. mientras se consigue el mismo resultado. Como se muestra en la figura 31B, la función provista por la interfase I1 se puede subdividir en múltiples interfases I1a. I1b, I1c, etc. mientras se logra el mismo resultado.

En forma similar, la interfase I2 del segundo segmento de código que recibe la información del primer segmento de código puede ser factorizada en múltiples interfases IC, IB, IC, etc. Cuando se factoriza, el número de interfases se incluye con el primer segmento de código y no necesita corresponder al número de interfases incluido con el segundo segmento del código. En ambos casos de la figura 31A y 30B, el espíritu funcional del sistema de interfases de interfase 1 y I1 se mantienen igual a las figuras 30A y 30B, respectivamente. La factorización de las interfases puede seguir propiedades matemáticas asociativas, conmutativas y otras tales que la factorizacion puede ser difícil de reconocer. Por ejemplo, el ordenamiento de operaciones puede no ser importante, y consecuentemente, una función que una interfase lleva a cabo puede ejecutarse antes de llegar la interfase, por otra parte del código o interfase, o ejecutada por un componente del sistema separado. Aún más, uno de destreza ordinaria en el arte de la programación puede apreciar que hay una variedad de maneras de llamar distintas funciones que logran el mismo objetivo.

[0330] Redefinición: en algunos casos, es posible hacer caso omiso, añadir o redefinir ciertos aspectos (por ejemplo, parámetros) de una interfase de programación mientras se

logra el resultado buscado. Esto se muestra en la figura 32 finA y 32B. Por ejemplo, en la interfase Interfase 1 de las figuras 30ª se incluye una función llamada Square (entrada, precisión, salida), es una llamada que incluye tres parámetros, entrada, precisión y salida, y que emana del primer segmento de código al segundo segmento del código. Si el parámetro de precisión del centro no importa en un escenario dado, como se muestra en la figura 32A, podría perfectamente ser ignorado e incluso reemplazado con un parámetro sin significado (en esta situación). Uno podría también añadir un parámetro adicional sin importancia. En cualquiera de los casos, la funcionalidad de square se puede lograr, siempre y cuando una salida se devuelva luego de que la entrada es squared por el segundo segmento del código. La precisión puede muy bien ser un parámetro significativo a algunos sistemas de computación; sin embargo, una vez que se reconoce que la precisión no es necesaria para los propósitos estrechos de cálculo del square, se puede remplazar o ignorar. Por ejemplo, en vez de pasar un valor de precisión válido, un valor sin sentido tal como la fecha de nacimiento se puede pasar sin que afecte en forma adversa el resultado. En forma similar, como se muestra en la figura 32B, la interfase I1 se reemplaza por la interfase I1[0, para redefinir, ignorar o añadir parámetros a la interfase. Similarmente, la interfase I2 se puede redefinirse como la interfase I2[0, redefinir para ignorar los parámetros innecesarios, o parámetros que se pueden procesar en algún otro sitio. Acá es que en algunos casos una interfase de programación puede incluir aspectos, tales como parámetros, que por alguna razón no se necesitan, entonces se pueden ignorar o redefinir, o procesar en algún otro lado para otros propósitos.

[0331] Codificación en línea: también es posible fusionar algunas o todas las funcionalidades de los módulos de código separados, de tal manera que la "interfase" entre ellos cambia de forma. Por ejemplo, la funcionalidad de la figura 30A y 30B se puede convertir en la funcionalidad de la figura 33A y 33B respectivamente. En la figura 33A, el primer y segundo segmento del código de la figura 30A se fusionan en un módulo que los contiene a ambos. En este caso, los segmentos de código pueden todavía comunicarse entre ellos pero la interfase se puede adaptar en una forma que es más adecuada al módulo único. Así, por ejemplo, comandos formales como Call y Return pueden no ser ya necesarios, pero procesos similares o respuestas concernientes a la interfase Interfase 1 pueden todavía estar

vigentes. En forma similar se muestra en la figura 33B, parte (o todo) de la interfase I2 1000 y de la figura 30B que se pueden describir en línea dentro de la interfase I1 para formar la interfase I1[0. Como se muestra, la interfase I2 se divide en I2a y I2b, y la porción de interfase I2a ha sido codificada en línea con la interfase I1 para formar a la interfase I1[0. Para poner un ejemplo concreto, considere que la interfase I1 de la figura 30B ejecutó una función llamada square (entrada, salida), que la interfase I2 recibe, y luego de procesar el valor recibido con una entrada (elevarlo al cuadrado) por el segundo segmento de código, pasa de vuelta el resultado elevado al cuadrado con la salida. En tal caso, el procesamiento ejecutado por el segundo segmento de código (elevando la entrada al cuadrado) puede ejecutarse por el primer segmento de código sin hacer una llamada a la interfase.

[0332] Divorcio: una comunicación de un segmento de código a otro se puede lograr indirectamente rompiendo la comunicación en múltiples comunicaciones discretas. Esto se muestra esquemáticamente en las figuras 34A y 34B. Como se muestren las figuras 34A una o más piezas del middleware (Interfase de Divorcio (s), puesto que divorcian funcionalidaes y/o funciones de interfase de una interfase original) se suministran para convertir las comunicaciones en la primera interfase, Interfase 1, para conformarlas a una interfase diferente, en éste caso la interfase Interfase2A, Interfase2B, Interfase2C. Esto se puede hacer, por ejemplo, donde hay una base de aplicaciones instalada diseñada para comunicarse, por ejemplo, con un sistema operativo de acuerdo con un protocolo de Interfase1, pero luego el sistema operativo cambia a utilizar una interfase diferente, en éste caso las interfases Interfase2A, Interfase2B y Interfase2C. El punto es que la interfase original utilizada por el segundo segmento de código cambia de tal manera que ya no es compatible con la interfase utilizada por el primer segmento de código, y por lo tanto un intermediario se utiliza para hacer que la vieja y nueva interfases sean compatibles. En forma similar, como se muestra en la figuras 34B, un tercer segmento de código se puede introducir con la interfaseDI1 divorcio para recibir las comunicaciones de la interfaseI1 y con la interfaseDI2 divorcio para transmitir la funcionalidad de la interfase a, por ejemplo, las interfasesI2A yI2B, rediseñadas para trabajar con DI2, pero para dar el mismo resultado funcional. En forma similar, DI1 y DI2 pueden trabajar juntos para traducir la

funcionalidad de las interfases I1 y I2 de la figura 30B a un sistema operativo nuevo mientras proveen el mismo resultado funcional.

[0333] Reescribir: otra posible variante es re escribir dinámicamente el código para reemplazar la funcionalidad de la interfase con algo más pero que logre el mismo resultado general. Por ejemplo, puede haber un sistema en donde un segmento de código presentado en un lenguaje intermedio (por ejemplo, Microsoft IL, Java BiteCode, etc.) se suministran a un compilador Just-in-time (JIT) o intérprete en un ambiente de ejecución (como el que provee Net framework, o el Java runtime, u otro tipo de ambientes runtime). El compilador JIT se puede escribir de tal manera que convierta dinámicamente las comunicaciones del primer segmento de código al segundo segmento de código, por ejemplo, para conformarlos a una interfase diferente como lo puede requerir el segundo segmento de código (ya sea del original o una segundo segmento de código distinto). Esto se muestra en la figura 35A y 35B. Como se puede ver en la figura 35A, este enfoque es similar al escenario Divorcio que se describió más arriba. Se puede hacer, por ejemplo, donde una base de aplicaciones está instalada y diseñada para comunicarse con un sistema operativo de acuerdo con el protocolo de la Interfase 1, pero luego el sistema operativo cambia a utilizar una interfase diferente. El Compilador JIT podría ser usado para conformar la comunicación al vuelo de las base de aplicaciones instalada a la nueva interfase del sistema operativo. Como se muestra en la figura 35B, éste enfoque de re escribir dinámicamente la interfase (s) se puede aplicar para qué altere la interfase dinámicamente.

[0334] Se debe notar que el escenario descrito anteriormente para lograr el mismo resultado o similar que con una interfase a través de una implementación alternativa también se puede combinar de varias maneras, en forma serial y/o paralela, o con otros códigos que intervengan. De esta manera, la implementación alternativa que se presenta arriba no son mutuamente exclusivas y se pueden mezclar, coordinar y combinar para producir el mismo escenario o equivalente al escenario genérico presentado en la figura 30A y 30B. También se nota que, como con la mayoría de construcciones de programación, a y otras formas similares de lograr la misma funcionalidad o similar de una interfase que puede no estar descrita, pero que sin embargo están representadas por el espíritu y el alcance del invento,

por ejemplo se nota que por lo menos parte de la funcionalidad representada por, y los resultados ventajosos que se logran con una interfase subyacente al valor de una interfase.

III. EXTENSIONES Y HERENCIAS

[0335] Un concepto fundamental del presente invento es la utilización de Items que, hasta cierto punto, modelan objetos de aplicaciones del mundo real con estructuras complejas, comportamientos, y operaciones descritas por un esquema y regidas por un sistema de interfase de hardware/software. Para poder suministrar una funcionalidad rica en sub tipificar, en varias implementaciones del presente invento, un sistema de interfase de hardware/software (que por conveniencia, simplemente vamos a llamar "WinFS") puede suministrar un mecanismo mediante el cual los Items (y tipos de Items) se pueden extender utilizando "Extensiones". Las Extensiones proveen una estructura de datos adicional (Propiedades, Relaciones, etc.) a una estructura existente de tipos de Items.

[0336] Cómo se discutió previamente (y particularmente en las secciones II.E.6 (a) y II.F.3, mientras que la intención de la plataforma de almacenaje es la de proveer un conjunto inicial de esquemas, en una implementación la plataforma de almacenaje le permite a los clientes, incluyendo vendedores independientes de software (ISVs), crear nuevos esquemas (por ejemplo, un nuevo Item y tipos de Elementos Anidados). Como un tipo de Items o tipo de Elemento Anidados definido por el esquema inicial de la plataforma de almacenaje puede no exactamente corresponder con el ISV necesitado por la aplicación, es necesario permitir que el ISV adapte el tipo. Esto se permite con la noción de Extensiones. Las Extensiones son instancias fuertemente tipificadas pero (a) no pueden existir en forma independiente y (b) deben estar adheridas a un Item o Elemento Anidado. También, adicionalmente a dirigirse a la necesidad de la Extensibilidad del esquema, las Extensiones también tienen el propósito de dirigirse al tema de la "multi-tipificación". Como, en algunas implementaciones la plataforma de almacenaje puede no apoyar las herencias múltiples o los subtipos que se sobreponen, la aplicación puede utilizar las Extensiones como una

forma de modelar las instancias de tipos que se sobreponen (por ejemplo un Documento puede ser un "documento legal" como también un "documento seguro").

A. SISTEMA TIPO

[0337] En varias implementaciones del presente invento, el sistema tipo de WinFS suministra un mecanismo para definir estructuras de datos. El sistema tipo se utiliza para representar datos almacenados en WinFS. Un tipo WinFS se declara un esquema WinFS. Un esquema WinFS define un namespace que sirve como una agrupación lógica para un conjunto de tipos y otros elementos del esquema WinFS. Los Esquemas WinFS se pueden declarar utilizando un Lenguaje de Definición de Esquemas WinFS (SDL) que puede utilizar en un formato XML. Lo que sigue es un ejemplo de una declaración de un esquema posible:

<Schema Namespace="System.Storage" >

Type definitions

</Schema>

[0338] Los Esquemas WinFS también sirven como una unidad para establecer las versiones. El WinFS define varios esquemas de sistema que amarran el sistema. Estos son el esquema namespace System.Storage que contienen la declaración del tipo de los tipos raíz en el sistema, y el System.Storage . El esquema namespace WinFS que declara todos los tipos escalares primitivos en el sistema.

[0339] El sistema tipo de WinFS declara una serie de tipos escalar simples. Estos tipos se utilizan como el bloque más primitivo de construcción para los demás tipos en el sistema tipo WinFS. Estos tipos se declaran en el esquema namespace System.Storage . WinFS. La siguiente tabla define el conjunto de tipos primitivos.

WinFS Type	Tipo SQL Administrado	Tipo CLR	Descripción
String	SqlString	String	Largo de la variable. Unicode datos con el

			largo máximo de largo of 2^31 caracteres. Largo puede estar fijo en 1 – 4000 caracteers o sin constricción utilizando la clave "max"
Binary	SqlBinary	Byte[]	Largo de la variable binario datos con el largo máximo de largo de 2^32 bytes. Largo puede estar fijo en 1 – 8000 o sin constricción utilizando la clave "max" .
Boolean	SqlBoolean	Boolean	Valor Null-able Boolean
Byte	SqlByte	Byte	Un bite único no registrado
Int16	SqlInt16	Int16	Datos integrales de -2^15 (-32,768) through 2^15 - 1 (32,767).
Int32	SqlInt32	Int32	Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647).
Int64	SqlInt64	Int64	Integrales (números enteros) datos de -2^63 (-9223372036854775808) hasta 2^63-1 (9223372036854775807).
Sencillo	SqlSingle	Single	Dato de numero de precision flotante de -1.79E + 308 hasta1.79E + 308
Doble	SqlDouble	Double	Dato de numero de precision flotante de -3.40E + 38 through 3.40E + 38
Decimal	SqlDecimal	Decimal	SQLDecimal tiene un valor de rango superior a CLR tipo Decimal. En el almacén la precision siempre es de 28 digitos y la escala es 0.
FechaHora	SqlDateTime	DateTime	Datos de fecha y hora desde Enero 1, 1753,

			hastaDeciembre 31, 9999, con una exactitud de 3 milesimes de segundo, o 3.33 milisegundos.
Guid	SqlGuid	Guid	Un identificador global unico (GUID).
Xml	SqlXmlReader	XmlReader	Para el Hito B, WinFS.Xml será maperado a "String" type. True XML datatype se espera compatibilidad en el Hito C.
Stream	TBD	TBD	Un tipo de datos binario que utiliza un almacenaje de resguardo de flujo de archivos para un acceso mas eficiente. Este tipo será apoyado en el Hito C.

[0340] Una enumeración WinFS en es un tipo escalar que declara un conjunto de constantes nombradas llamadas lista de valores. Un tipo enumeración se puede usar en cualquier sitio donde un tipo escalar se puede usar. Aquí hay un ejemplo para una declaración enumeración:

```
<Enumeration Name="Gender" >
        <Value Name="Male" />
        <Value Name="Female" />
      </Enumeration>
```

Los valores de enumeración están basados en cero. En el ejemplo anterior Gender.Male representa un valor cero y Gender.Fem representa un valor 1.

[0341] Un tipo complejo se define con un nombre y un conjunto de propiedades. Una Propiedad es un miembro del campo tipo, y se define con un nombre y un tipo. El tipo de Propiedad puede ser un escalar (incluyendo al tipo enumeración) u otro tipo complejo. Un tipo WinFS que se puede utilizar como un tipo de Propiedad se llama un tipo anidado. Una instancia del tipo anidado sólo puede existir como un valor de Propiedad de un tipo complejo WinFS -la instancia es anidada dentro de una instancia del tipo complejo. Un tipo anidado se declara utilizando el elemento de esquema NestedType. Aquí hay unos ejemplos de un tipo de declaración válida:

```
<NestedType Name="Address" BaseType="System.Storage.NestedObject">
                <Property Name="Street" Type="WinFS.String" Size="256"
               Nullable="false" />
                <Property Name="City" Type=" WinFS.String" Size="256"
               Nullable="false" />
                <Property Name="State" Type=" WinFS.String" Size="256"
               Nullable="false" />
                <Property Name="Country" Type="WinFSString" Size="256"
               Nullable="false" />
               </NestedType>
               <ItemType Name="Person" BaseType="System.Storage.Item">
                <Property Name="Name" Type="WinFS.String" Size="256"
               Nullable="false" />
                <Property Name="Age" Type="WinFS.Int32"
               Nullable="false" Default="1"/>
                <Property Name="Picture" Type="WinFS.Binary" Size="max"/>
                <Property Name="Addresses" Type="MultiSet"
MultiSetOfType="Address"/>
               </ItemType>
```

[0342] Para propiedades del tipo String y Binary un atributo Size se debe especificar. Éste atributo especifica el tamaño máximo del valor contenido en Propiedad. Propiedad puede opcionalmente declarar un constrefiimiento de nulabilidad utilizando el atributo Nullable. Un valor "falso" para éste atributo indica que la aplicación debe suministrar un valor cuando crea una instancia de este tipo. Otro atributo opcional Propiedad es el atributo por defecto que especifica el valor por defecto para Propiedad. En este caso será asignado a Propiedad en el momento de la creación si la aplicación no lo suministró.

[0343] La Address Propiedad en el ejemplo anterior es del tipo Multiset. Una Propiedad del tipo Multiset también es llamada una Propiedad multi-valor. En el ejemplo Multiset contiene un conjunto de instancias del tipo Address. Un Multiset es similar a una corrección. Puede contener cero o más instancias del tipo complejo. El tipo de instancias en

Multiset debe ser un tipo complejo anidado. El tipoMultiset no es compatible con instancias de tipo escalar WinFS (incluyendo los tipos de enumeración). Una Propiedad del tipo Multiset no puede ser anulable y no puede tener un valor por defecto.

[0344] WinFS apoya la herencia sencilla de tipos. Todos los tipos en WinFS deben heredar uno y sólo un tipo WinFS. Al tipo heredado se le refieren como tipo derivado y el tipo del cual este tipo fue derivado se le refieren como tipo base. El tipo base es el atributo BaseType del elemento de declaración tipo WinFS. Supóngase que el tipo A se deriva del tipo base B que a su turno se deriva del tipo C. El tipo C es un tipo ancestro de los tipos a A y B. Una instancia de datos almacenados en WinFS es siempre una instancia de tipo sencillo. Podemos sin embargo tratar la instancia de datos como una instancia de un conjunto de tipos que contienen al tipo de todos sus tipos ancestros. Para una instancia de datos es una instancia de dicho conjunto de tipos, llamamos al tipo que no es un ancestro de ningún otro tipo en el conjunto el tipo más derivado. En general, nos vamos a referir al tipo más derivado de un elemento tipo sencillo como su tipo. El tipo derivado hereda todas las propiedades declaradas en su tipo base. El tipo derivado puede declarar nuevas propiedades pero no puede anular las propiedades definidas en el tipo base. Una Propiedad declarada en el tipo derivado no debe usar el mismo nombre de la Propiedad del tipo base.

[0345] La ventaja principal de la herencia en el modelo de datos viene de la capacidad de sustitución en los tipos heredados. Considere el siguiente ejemplo:

```
<NestedType Name="Name" BaseType="System.Storage.NestedObject" >
                <Property Name="FirstName" Type="WinFS.String" />
                <Property Name="LastName" Type="WinFS.String" />
               </Nestedype>
               <NestedType Name="NameWithMiddleInitial" BaseType="Name" >
                <Property Name="MiddleInitial" Type="WinFS.String" />
               </NestedType>
               <NestedType Name="Person" BaseType="System.Storage.Item" >
                <Property Name="RealName" Type="Name" />
                <Property Name="OtherNames" Type="MultiSet"
MultiSetOfType="Name" />
```

</NestedType>

En el ejemplo de arriba, el tipo Persona tiene una Propiedad RealName del tipo Nombre y Propiedad OtherName que es un conjunto del tipo Nombre. Normalmente se requeriría que la Propiedad RealName tuviera una sola instancia cuyo tipo es Nombre. Sin embargo, con la herencia, se permite que instancias de valor único sean el valor de RealName siempre y cuando el tipo Nombre es uno de los tipos más derivados ancestros de ese elemento. Por lo tanto, una instancia de NameWithMiddleInitial será permitida como valor de Propiedad RealName.

[0346] La misma regla se extiende al conjunto de propiedades. La propiedad OtherNames contiene un conjunto de elementos. Para cada instancia del tipo sencillo que es un miembro de ese conjunto, el tipo más derivado de esa instancia debe tener Nombre como uno de sus ancestros. De tal manera, que algunas instancias en el conjunto OtherNames puede ser instancias del tipo Nombre mientras que otros pueden ser instancias del tipo NameWithMiddleInitial.

[0347] Las herencias también permiten una consulta conveniente en el sentido de que es posible en el sistema WinFS encontrar instancias de cierto tipo. Cuando se buscan las instancias de un tipo, el motor de búsqueda también regresará todas las instancias que su tipo más derivado son descendientes de este tipo. Sin embargo, estas operaciones sólo son compatibles con tipos Item, Extensión, y Relación (no tipos Propiedad). Para tipos anidados (a.k.a, Elementos Anidados, Propiedades, o tipos de propiedades complejas), la operación sólo está apoyada por instancias contenidas dentro de una Propiedad multiset única.

B. TIPOS DE FAMILIA

[0348] En resumen el sistema WinFS define cuatro tipos de familias distintas:

- Tipos de Elementos Anidados (a,k.a. Tipos anidados o tipos propiedad)
- Tipos de item
- Tipos de Relaciones
- Tipos de Extensiones

Cada tipo de familia tiene un conjunto de propiedades diferente y de uso en el sistema de tipo WinFS. El esquema namespace System.Storage declara cuatro tipos que sirven como tipos raíz para cada tipo de familia. La siguiente sección describe los tipos de familia en detalle.

1. Tipos de Elementos Anidados.

[0349] A diferencia de otras familias tipo WinFS, los tipos anidados se pueden usar como tipos de propiedades de tipos complejos WinFS. Instancias de un tipo anidado sólo pueden estar anidados dentro de una instancia de otro tipo complejo. Sin embargo instancias del tipo anidado no pueden ser consultadas en forma global -eso es, las aplicaciones no pueden componer una consulta sencilla que regrese todas las instancias de un tipo anidado dado en el almacén WinFS.

2. Tipos de Item

[0350] Un item WinFS es una instancia de un tipo cuyo ancestros es del tipo System.Storage.Item. Este tipo es un tipo complejo que es la raíz del tipo de familia del Item. System.Storage .Item que declara una Propiedad del nombre ItemId del tipo Guid. Ésta es una Propiedad de un Item que sirve como llave primaria del item. El valor de ésta Propiedad se garantiza que es único para los items en un almacén WinFS dado. Esta Propiedad no es anulable, y debe ser asignada por la aplicación cuando crea una instancia de un tipo de Item. La Propiedad ItemId también es inmutable, nunca puede cambiar y no se puede re usar.

[0351] El motor de consulta puede regresar instancias de un tipo de Item dado en el almacén WinFS. Ésta consulta puede regresar todos los instancias del tipo de todos los tipos del tipo descendiente. Luego describiremos el papel central que los Items tienen en la semántica del sistema operacional WinFS.

3. Tipos de Relaciones

[0352] Los tipos de Relaciones permiten que Relaciones existan entre Items. Tipos de Relaciones WinFS describen una Relación binaria donde un Item se designa como la fuente y el otro Item se designa como el objetivo. Una Relaciones es una instancia de un tipo cuyo

ancestros es el tipo Relación System.Storage. Este tipo es una raíz de la jerarquía del tipo Relación. El tipo de Relación System.Storage declara las siguientes propiedades:

- SourceItemId – el ItemId del Item que es fuente de la instancias Relación
- RelationshipId – 1 identificador único de Relación relativo al Item fuente; él (SourceItemId, RelationshipId) forma la llave primaria para tipos de Relación WinFS
- TargetItemId – el ItemId del objetivo de la Relación;
- Mode – 1 de 3 posibles modalidades de instancias de Relación: pertenencia, incrustada o referencia
- Name – contiene el nombre de la Relación para las Relaciones de pertenencia
- IsHidden – un atributo Booleano que la aplicación puede utilizar opcionalmente para filtrar Relaciones que no necesitan mostrarse en pantalla

El SourceItemId, RelationshipId, TargetItemId y valor de Mode Property son inmutables. Están asignados al tiempo de creación de instancias de Relación y no pueden ser cambiadas.

[0353] Un tipo de Relación se declara como un tipo complejo con los constrefiimiento siguientes:

Especificación de puntos finales fuente y objetivo: cada punto final especifica un hombre y el tipo del Item se refiere

Permite modalidades de las instancias de un tipo de Relación: una instancia de Relación no pueden tener un valor para Mode Property que no esté permitido en la declaración de Relación

[0354] Este es un ejemplo de una declaración de Relación:

```
<RelationshipType Name="DocumentAuthor"
BaseType="System.Storage.Relationship"
AllowsHolding="true"
AllowsEmbedding="false"
AllowsReference="true" >
 <Source Name="Document" Type="Core.Document"/>
 <Target Name="Author" Type="Core.Contact" />
```

<Property Name="Role" Type="WinFS.String" />
<Property Name="DisplayName" Type="WinFS.String" />
</RelationshipType>

La Relación DocumentAuthor esta declarada con restricciones sobre las instancias en la modalidad de Pertenencia o Referencia. Esto significa que una instancia de una relación DocumentAuthor puede tener instancias con valor Mode="Reference" o Mode="Holding". Instancias con valor Mode="Embedding" no se permiten.

[0355] La Relación declara un punto final fuente llamado "Documentos" del tipo de Item "CoreDocument" si hay un punto final objetivo del tipo "CoreContacts". La Relación también declara dos propiedades adicionales. Las instancias de Relación se guardan y se acceden en forma separada del Item. Todos los tipos de instancias de Relación son accesibles a través de una vista de Extensión global. Una consulta se puede componer para que reporte todas las instancias de un tipo dado de Relación.

[0356] Dado un Item todas las Relaciones para las cuales ese Item es una fuente se pueden enumerar en la propiedad SourceItemId de la Relación. En forma similar para un item dado todas las Relaciónes en el mismo almacén para el cual el Item es un objetivo se pueden enumerar utilizando la propiedad TargetItemId de la Relación.

a) Semántica de las Relaciónes

[0357] La siguiente sección describe la semántica de distintas modalidades de instancias de Relación:

[0358] Relaciones de Pertenencia: las Relaciónes de Pertenencia se utilizan para modelar la administración de referencias contables de los Items objeto. Un Item puede ser el punto final fuente para cero o más Relaciónes con un Item. Un Item que no es un Item incrustado puede ser el objeto de una y más Relaciónes de pertenencia. El Item objeto dede estar en el mismo almacén que la instancia de la Relación.

[0359] Las Relaciones de Pertenencia cohercen una administración de por vida del punto final objeto. La creación de una instancia de Relación de pertenencia y el Item al que apunta es una operación atómica. Relaciónes de pertenencia adicionales se pueden crear de tal manera que apuntan al mismo Item. Cuando la última instancia de una Relación de

pertenencia con un Item dado como punto final objeto se elimina el Item objeto también se elimina.

[0360] Los tipos de los Items de punto final se especifican en la declaración de la Relación y serán obligados cuando una instancia de la Relación se cree. Los tipos de Items de punto final no se pueden cambiar luego de que la Relación se establece.

[0361] Las Relaciónes de Pertenencias juegan un rol importante en la formación del Item namespace WinFS. Todas las Relaciónes de pertenencia participan en la declaración namespace. La Propiedad "Nombres" en la declaración de la Relación define el nombre del Item objeto relativo al Item fuente. Esté nombre relativo es único para todas las Relaciónes de pertenencia que se derivan de un Item dado y no pueden ser nulas. La lista ordenada de estos nombres relativos comenzando por el Item raíces a un Item dado forma el nombre completo del Item.

[0362] Las Relaciónes de pertenencias forman una gráfica dirigida acíclica (DAG). Cuando una Relación de pertenencia se crea el sistema se asegura que un ciclo no se cree y así se asegura que el namespace Item WinFS forma un DAG. Para más información sobre el namespace WinFS y paths de Items referirse a las especificaciones del "namespace WinFS".

[0363] Relaciones de Incrustación: las Relaciones de Incrustación modelan el concepto de control exclusivo del tiempo de vida del Item objeto. Ellas habilitan el concepto de Items compuestos. La creación de instancias de Relaciones de Incrustación y el Item al que apuntan es una operación atómica. Un Item puede ser la fuente de cero o más Relaciones de Incrustación. Sin embargo, un Item puede ser el objetivo de una y sólo una Relaciones de Incrustación. Un Item que es el objetivo de una Relaciones de Incrustación no puede ser el objetivo de una Relación de pertenencia. El Item objeto debe estar en el mismo almacén en donde estaba la instancia de la Relación.

[0364] El tipo de Items de punto final especificados en la declaración de la Relación se verán forzados cuando una instancia de la Relación se crea. Los tipos de los Items de punto final no se pueden cambiar luego de que la Relación se ha establecido. Las Relaciones de Incrustación no participan en el namespace WinFS. El valor de Name Property de una Relacion de Incrustación debe ser nulo.

[0365] Relaciones de Referencia: las Relaciones de Referencia no controlan el tiempo de vida del Item al que hacen referencia. Las Relaciones de Referencia no garantizan la existencia del objetivo, ni garantizan el tipo del objetivo como se especifica en la declaración de la Relación. Esto significa que la referencia de la Relación puede estar en el aire. También, la Relación de referencia puede hacer referencia a Items en otros almacenes WinFS.

[0366] En una referencia WinFS las Relaciones se utilizarán para moldear, de por vida, la mayoría de Relaciones administrativas entre los Items. Puesto que la existencia del objetivo no se obliga, la Relación de referencia es conveniente para moldear Relaciones sueltas. La Relación de referencia se puede utilizar para apuntar a Items en otros almacenes WinFS incluyendo almacenes en otras máquinas. Las Relaciones de incrustación no participan en el namespace WinFS. El valor de NameProperty de una Relación de incrustación de ser nulo.

b) Reglas de las Relaciones y Constreñimientos

[0367] Las siguientes reglas adicionales y de constreñimiento aplican a las Relaciónes:

- Un Item debe ser objetivo de (exactamente una Relación de incrustación) o (una o más Relaciónes de pertenencia). La única excepción es el Item raíz. Un Item puede ser un objetivo de cero o más Relaciónes de referencia.
- Un item que es objetivo de una Relación de incrustación no puede ser fuente de una Relación de pertenencia. Puede ser fuente de una Relación de referencia.
- Un Item no puede ser fuente de una Relación de pertenencia si sea promovido de un archivo. Puede ser fuente de Relaciónes de incrustación y si de referencias.
- Un Item que se ha promovido de un archivo no puede ser objetivo de una Relación de incrustación.

[0368] Cuando una Relación tipo A se deriva de una Relación base tipo B las siguientes reglas aplican:

- La Relación tipoA puede restringir aún más los tipos de puntos finales. Los tipos de puntos finales deben ser subtipos del punto final correspondiente en la

Relación base tipo B. Si el punto final se restringe aún más un nuevo nombre para el punto final se declara. Si el punto final no se restringe la especificación del punto final es opcional.

- La Relación tipoA puede restringir aún más la modalidad de instancias declaradas en la Relación base. El conjunto restringido de modalidades de instancias debe ser un subconjunto del conjunto tipo base de tipos de instancias permitidos.
- Los nombres de los puntos finales se tratan como nombres de Propiedad: no pueden ser lo mismo que el nombre de una Propiedad o el nombre de un punto final de su tipo o tipo base.
- Los elementos Fuente y Objetivo son opcionales si el tipo del punto final correspondiente no tiene restricciones adicionales de la Relación derivada

Aquí hay un ejemplo de una declaración de un tipo de Relación que deriva de la Relación DocumentAuthor que se definió anteriormente:

```
<RelationshipType Name="LegalDocumentAuthor"
BaseType="Core.DocumentAuthor"
AllowsHolding="false"
AllowsEmbedding="false"
AllowsReference="true" >
 <Source Name="LegalDocument" Type="Legal.Document "/>
 <Property Name="CaseNumber" Type="WinFS.String" />
</RelationshipType>
```

La Relación LegalDocumentAuthor restringe aún más del punto final fuente pero no del punto final objetivo. El punto final fuente desde el tipo Legal.Document que se deriva de Core.Document. El punto final objetivo no se restringe más en este caso, por lo tonto el elemento objetivo se omite. En la relación]restringe adicionalmente la modalidad de instancias permitidas. No permite la modalidad Pertenencia y la única modalidad permitidas es la de Referencia.

4. Tipos de Extensiones

[0369] Una Extensión WinFS es una instancia de un tipo cuyo ancestro es del tipo System.Storage.Extension. Este tipo es un tipo complejo que es la raíz de la familia del tipo Extensión.

- System.Storage.Extension define dos propiedades:
- ItemId –0 ItemId of del Item con el cual la Extensión está asociada
- ExtensionId –un identificador único para la Extensión relativa a ItemId. El par (ItemId, ExtensionId) identifican en forma única una instancia de Extensión.

[0370] Las siguientes restricciones aplican para tipos de Extensión:

- Las instancias de tipos de Extensión no pueden existir independientes de un Item. Una instancia de tipo Item con el mismo ItemId que el ItemId de la Extensión debe existir en el almacén antes que la instancia del tipo Extensión se cree. La Extensión no se puede crear si el Item con el ItemId no existe. Cuando el Item se elimina todas las Extensiones con el mismo ItemId se eliminan.
- Máximo una instancia de un tipo de Extensión más derivado se puede asociar con un Item individual
- Las Extensiones no pueden ser fuente y objetivo de las Relaciónes.

[0371] No hay constreñimientos en los tipos de Extensión que se pueden asociar con un tipo de Item dado. Se permite que cualquier tipo de Extensión se extienda a cualquier tipo de Item. Cuando instancias múltiples de distintos tipos de Extensión se adhieren a un Item, son independientes el uno del otro tanto en estructura como en comportamiento. Las instancias de Extensión se guardan y se acceden en forma separada del Item. Todas las instancias de tipos de Extensión se pueden acceder en una vista global de Extensiones. Una consulta se puede componer para que reporte todas las instancias de un tipo de Extensión dado sin importar con que tipo de Item están asociados. El ItemID de la Extensión indica a que Item pertenecen y se pueden utilizar para recuperar el objeto de Item correspondiente de la vista de Items global. También, dado un Item todas las instancias de Extensión asociadas con el Item se pueden enumerar utilizando la Propiedad de la Extensión ItemId.

C. FUNCIONALIDAD MEJORADA

[0372] En varias implementaciones del presente invento, un sistema de interfase de hardware/software utiliza Extensiónes y Herencias para formalizar las relaciones entre varios Items y de esta manera mejorar la habilidad para consultar una pluralidad de Items.

1. Herencias

[0373] La figura 36 muestra una serie de Items interrelacionado -subconjunto de sus Relaciones. El Document Item 3602 y el Contact Item 3604 están directamente relacionados por una relación designada 3606 que en este caso, es una "author relationship"—esto es, el Contact 3604 del "autor" del Document 3602. En este ejemplo, el Picture Item 3622, el Music Item 3624, y el Special Item 3626 todos heredan del Document Item 3604 porque cada tipo de Item es un subtipo de Document Item type. En forma similar, el Person Item 3642 y el Organization Item 3644 heredan de Contact Item 3604. En varias implementaciones del presente invento, estos Items que heredan (Picture 3622, Music 3624, Special 3626, Person 3642, y Organization 3644) no sólo heredan las Propiedades del Item paterno respectivo (Document 3602 y Contact 3604), pero también heredan las Relaciones designadas entre estos dos Items paternos. Por ejemplo, Picture 3622 hereda una relación 3662 con Contact 3604, una relación 3664 con Person 3642 y una relación 3666 con Organization 3644. Un conjunto similar de Relaciones también se heredan por cada uno de los otros Items que se muestran.

[0374] Sin embargo, es importante notar que la herencia de relaciones no es automática y no ocurre en todos los contextos. Por ejemplo, atributos que describen cuando un tipo puede heredar (por ejemplo, controles de herencia) en sí no se heredan. Los parámetros de densidad se mantienen y se regulan por la sistema de interfase de hardware/software.

2. Extensiones

[0375] La figura 37A muestra las debilidades de la sub tipificacion estándar de un Item por razones específicas de la aplicación. En esta figura, un Contacto es accesible a cuatro aplicaciones, APP1, APP2, APPX, y APPY. APP1 y APP2 accesan Contact, pero cada una

de APPX y APPY necesitan un un objeto de contacto extendido (añadir campo adicional) y así derivar Contacto' y Contacto", cada uno de los cuales hereda de Contacto. Sin embargo, el problema es que ahora hay tres instancias diferentes del Item Contacto básico -uno en Contacto, otro en Contacto', y otro en Contacto".

[0376] Una solución parcial a este problema, que se ilustra en la figura 37B, es extender las propiedades de Contacto para incluir los campos necesarios para una aplicación que los requiera. En éste caso, Contacto se extiende para incluir los Campos Adicionales requeridos por APPX. Sin embargo, extender directamente los campos de un Item tal como Contacto sólo se puede hacer una vez, y así APPY no puede utilizar este método.

[0377] En una implementación del presente invento, una solución más amplia es extender Contacto con una Extensión que es distinta y separada de Contacto en sí mismo como se muestra en la figura 37C. De esta manera, APPX puede extender a Contacto para incluir su Campo Adicional APPX mientras APPY también puede en forma separada extender a Contacto para incluir sus Campos Adicionales APPY. Estas Extensiones se pueden buscar y consultar en sí mismas, y así estas Extensiones permiten una forma de multi-tipificacion para el sistema de interfase de hardware&software.

IV. CONCLUSIÓN

[0378] Como lo anterior muestra, el presente invento está dirigido a plataformas de almacenaje para organizar, buscar, y compartir datos. La plataforma de almacenaje del presente invento extiende y amplía el concepto de almacenaje de datos más allá del sistema de archivos existente y sistemas de bases de datos, y está diseñada para almacenar todo tipo de datos, incluyendo estructurados, no estructurados, o semi-estructurados, tales como datos relacionales (tabular), XML, y nuevas formas llamados Items. A través de su fundamento común y datos esquematizados, la plataforma de almacenaje del presente invento permite un diseño de aplicaciones más eficientes para el consumidor, trabajadores con conocimiento y empresas. Ofrece una interfase de programación de aplicaciones rica y extensible que no sólo hace las capacidades inherentes en el modelo de datos disponible,

pero también abraza y extiende a los sistemas de archivos existentes y métodos de acceso a las bases de datos. Se entiende que se pueden hacer cambios a las implementaciones descritas anteriormente sin alejarnos del concepto fundamental del invento. De esta manera, el presente invento no se limita a la implementación particular descrita, pero pretende cubrir todas las modificaciones que se hagan dentro del espíritu y el alcance del invento tal como se define en las reivindicaciones anexas.

[0379] Cómo es aparente de lo antedicho, todo o parte de los varios sistemas, métodos, y aspectos del presente invento se pueden implementar en la forma de un código de programa (por ejemplo instrucciones). Este código de programa se puede almacenar en un medio legible por computador, como magnético, eléctrico, medios de almacenaje ópticos, incluyendo y sin limitaciónes discos floppy, CD-ROM, CD-RW, DVD-ROM, DVD-RAM, cientas magnéticas, memoria flash, discos duros o cualquier otro medio de almacenaje legible por una máquina, donde, cuando un código de programa se carga y es ejecutado por una máquina, como un computador, o un servidor, la máquina se convierte en un aparato en el cual se pueden practicar el invento. El presente invento también se pueden implementar en la forma de un código de programa que se trasmite a través de un medio de trasmisión, como cables eléctricos, fibras ópticas, redes, incluyendo Internet e intranet, o por cualquier otra vía o forma de trasmisión, donde, cuando el código de programa se reside y se carga y se ejecuta por la máquina, como un computador, la máquina se convierte en un aparato para la práctica de este invento cuando se implementa en un procesador de propósito general, el código de programa se combina con el procesador para suministrar un aparato único que opera análogo a circuitos lógicos específicos.

Lo que se reivindica es:

1. Un método para extender un Item, dicho item está constituido por una unidad de información discreta que se puede almacenar y manipular por un sistema de interfase de hardware/software, dicho método comprende la utilización de instancias fuertemente tipificadas (una "Extensión") para extender dicho Item, dicha Extensión constituyendo una unidad de información discreta que se puede almacenar y manipular por dicho sistema de interfase de hardware/software.

2. El método que se reivindica en 1 donde dicha Extensión se adhiere a dicho Item.

3. El método que se reivindica en 1 donde dicha Extensión no puede existir independiente de dicho Item, de tal manera que si dicho Item deja de existir, dicha Extensión también deja de existir.

4. El método que se reivindica en 1 donde dicho Items se extiende con una pluralidad de Extensiones.

5. El método que se reivindica en 4 donde dicha pluralidad de Extensiones se usan para moldear instancias de tipo que se sobreponen.

6. Un método para extender una Propiedad, dicha Propiedad constituye un tipo de propiedad compleja que se puede manipular por un sistema de interfase de hardware/software, dicho método comprende la utilización de instancias fuertemente tipificadas (una "Extensión") para extender dicha Propiedad, dicha extensión constituye una unidad de información discreta que se puede almacenar y manipular por dicho sistema de interfase de hardware/software y que está asociada con dicha Propiedad.

7.El método que se reivindica en 6 donde dicha Extensión se adhiere a dicha Propiedad.

8. El método que se reivindica en 6 donde dicha Extensión no puede existir independiente de dicha Propiedad, de tal manera que si dicha Propiedad deja de existir, dicha Extensión también deja de existir.

9. El método que se reivindica en 6 donde dicha Propiedad se extiende con una pluralidad de Extensiones.

10. El método que se reivindica en 9 donde donde dicha pluralidad de Extensiones se usa para moldear instancias de tipos que se sobreponen.

11. Un método para un sistema de interfase de hardware/software para organizar y consultar eficientemente una pluralidad de Items, dicho Item constituye una unidad de información discreta que se puede almacenar y manipular por un sistema de interfase de hardware/software, dicha pluralidad de Items comprende una primera Relación que relaciona un primer Item con un segundo Item, dicho método comprende, para la institución del tercer Item, dicho tercer Item siendo un subtipo de instancias de dicho primer Item, dicho tercer Item automáticamente hereda de dicho Item una Relación con dicho segundo Item.

12. El método que se reivindica en 11 donde, para una instancia del cuarto Item, dicho cuarto Item siendo un subtipo de instancias del segundo Item, dicho cuarto Item automáticamente hereda de dicho segundo Item una Relación con dicho primer Item.

13. El método que se reivindica en 12 donde dicho cuarto Item hereda automáticamente de dicho segundo Item una Relación con dicho tercer Item.

14. El método que se reivindica en 11 donde, para cada una de una primera pluralidad de instancias sub tipificadas de dicho primer Item cada una de esta primera pluralidad de instancias sub tipificadas automáticamente heredan de dicho primer Item una Relación con dicho segundo Item.

15. El método que se reivindica en 14 donde, por cada una de una segunda pluralidad de instancias sub tipificadas de dicho segundo Item dicha segunda pluralidad de instancias sub tipificadas automáticamente hereda de dicho segundo Item una Relación con dicho primer Item.

16. El método que se reivindica en 15 donde cada uno de esta primera pluralidad de instancias sub tipificadas automáticamente hereda de dicho primer Item una Relación con cada uno de dicha segunda pluralidad de instancias sub tipificadas.

17. El método que se reivindica en 16 donde cada uno de esta segunda pluralidad de instancias sub tipificadas automáticamente heredan de dicho segundo Item una Relación con cada uno de dicha primera pluralidad de instancias sub tipificadas.

18. Una sistema de interfase de hardware/software para manipular una pluralidad de Items, donde un Item constituye una unidad de información discreta que se puede almacenar y manipular por un sistema de interfase de hardware/software, dicho sistema comprende, un subsistema para extender Items con instancias fuertemente tipificadas (una "Extensión"), dicha Extensión constituye una unidad de información discreta que se puede almacenar y manipular por un sistema de interfase de hardware/software.

19. El sistema de la reivindicación 18 donde dicha Extensión va adherida a dicho Item.

20. El sistema de la reivindicación 18 donde dicha Extensión no puede existir independiente de dicho Item, de tal manera que si el Item deja de existir, dicha Extensión también deja de existir.

21. El sistema de la reivindicación 18 donde dicho Item se extiende con una pluralidad de Extensiónes.

22. Un sistema de interfase de hardware/software para manipular una pluralidad de Propiedades, dichas Propiedades constituyen un tipo de propiedad complejo que se puede manipular por un sistema de interfase de hardware/software, dicho sistema comprende un subsistema para extender una Propiedad con una instancia fuertemente tipificadas (una "extensión"), dicha Extensión constituida por una unidad de información discreta que se puede almacenar y manipular por un sistema de interfase de hardware/software.

23. El sistema de la reivindicación 22 donde dicha Extensión va adherida a dicha Propiedad.

24. El sistema de la reivindicación 22 donde dicha Extensión no puede existir independiente de dicha Propiedad, de tal manera que si dicha Propiedad deja de existir, dicha Extensión también deja de existir.

25. El sistema de reivindicación 22 donde dicha Propiedad se extiende con una pluralidad de Extensiónes.

26. Un sistema de interfase de hardware/software para manipular una pluralidad de items, donde un Item constituye una unidad de información discreta que se puede almacenar y manipular por dicho sistema de interfase de hardware/software, dicho sistema comprende un subsistema para organizar y consultar eficientemente una pluralidad de Items, dicha pluralidad de Items comprende una primera Relación que relaciona un primer Item y el segundo Item, donde dicho subsistema:

Para la instancia de un tercer Item, dicho tercer Item siendo una instancia sub tipificada de dicho primer Item, automáticamente establece una relación entre dicho tercer Itemy y dicho segundo Item;

En la instancia de un cuarto Item, dicho cuarto Item siendo una instancia sub tipificada de dicho segundo Item, automáticamente establece una relación entre dicho cuarto Item y dicho primer Item; y

Automáticamente establece una relación entre dicho cuarto Item y dicho primer Item.

27. El sistema de la reivindicación 26 donde, por cada primera pluralidad de un subtipo de instancias de dicho primer Item, por cada segunda pluralidad de instancias sub tipificadas de dicho segundo Item, dicho subsistema:

Automáticamente establece una relación para cada una de la primera pluralidad de instancias sub tipificadas con dicho segundo Item;

Automáticamente establece una relación para cada una de dicha segunda pluralidad de instancias sub tipificadas con dicho primer Item; y

Automáticamente establece una relación para cada una de dicha pluralidad de instancias sub tipificadas con cada uno de dicha segunda pluralidad de instancias sub tipificadas.

28. Un sistema de interfase de hardware/software para manipular una pluralidad de Items, donde un Item constituye una unidad de información discreta almacénable que se puede manipular por dicho sistema de interfase de hardware/software, dicho sistema comprende un subsistema para extender una Item con una instancia fuertemente tipificada (una "Extensión"), dicha Extensión constituye una unidad de información discreta almacénable que se puede manipular por dicho sistema de interfase de hardware/software.

29. El sistema de la reivindicación 28 donde dicha Extensión se adhiere a dicho Item.

30. El sistema de la reivindicación 28 donde dicha Extensión no puede existir independiente de dicho Item, de tal manera que si dicho Item deja de existir, dicha Extensión también deja de existir.

31. El sistema de la reivindicación 28 donde dicho Item se extiende con una pluralidad de Extensiónes.

32. Un sistema de interfase de hardware/software para manipular una pluralidad de propiedades, dichas Propiedades constituyen un tipo complejo de propiedad que se puede manipular por un sistema de interfase de hardware/software, dicho sistema comprende un subsistema para extender una Propiedad con una instancia fuertemente tipificada (una "Extensión"), dicha Extensión constituye una unidad de información discreta que se puede manipular y almacenar por dicho sistema de interfase de hardware/software.

33. El sistema de la reivindicación 32 donde dicha Extensión se adhiere a dicha Propiedad.

34. El sistema de la reivindicación 32 donde dicha Extensión no puede existir independiente de dicha Propiedad, de tal manera que si dicha Propiedad deja de existir, dicha Extensión también deja de existir.

35. El sistema de la reivindicación 32 donde dicha Propiedad se extiende con una pluralidad de Extensiones.

36. Un sistema de interfase de hardware/software para manipular una pluralidad de Items, donde un Item constituye una unidad de información discreta almacénable que se puede manipular por dicho sistema de interfase de hardware/software, dicho sistema comprende un subsistema para organizar y consultar eficientemente dicha pluralidad de Items, dicha pluralidad de Items comprende una primera Relación que relaciona un primer Item con un segundo Item, donde dicho subsistema:

Para la instancia de un tercer Item, dicho tercer Item siendo una instancia sub tipificada de dicho primer Item, automáticamente establece una relación entre dicho tercer Item y dicho segundo Item;

Para la instancia de un cuarto Item, dicho cuarto Item siendo un subtipo de instancia de dicho segundo Item, automáticamente establece una relación entre dicho cuarto Itemy dicho primer Item; y

Automáticamente establece una relación entre dicho cuarto Item y dicho primer Item

37. El sistema de la reivindicación 36 donde, para cada pluralidad de instancias sub tipificadas de dicho primer Item, y por cada segunda pluralidad de instancias sub tipificadas de dicho segundo Item, dicho subsistema:

Automáticamente establece una relación para cada una de dicha primera pluralidad de instancias sub tipificadas con dicho segundo Item;

Automáticamente establece una relación para cada una de dicha segunda pluralidad de instancias sub tipificadas con dicho primer Item; y

Automáticamente establece una relación para cada una de dicha primera pluralidad de instancias sub tipificadas con cada una de dicha segunda pluralidad de instancias sub tipificadas.

38. Un medio legible por computador que consta de instrucciones legibles por computador para extender un Item, donde un Item constituye una unidad de información discreta

almacénable que se puede manipular por dicho sistema de interfase de hardware/software, dicha instrucción legible por computador comprende instrucciones para la utilización de instancias fuertemente tipificadas (una "Extensión") para extender dicho Item, dicha Extensión constituye una unidad de información discreta que se puede almacenar y manipular por dicho sistema de interfase de hardware/software donde dicha Extensión está adherida a dicho Item y donde dicha Extensión también deja de existir cuando dicho Item deja de existir.

39. Un medio legible por computador que comprende instrucciones legibles por computador para extender una Propiedad, dicha Propiedad constituye un tipo de propiedad acompleja que se puede manipular por un sistema de interfase de hardware/software, dicho es instrucciones legibles por computador comprende en instrucciones por la utilización de instancias fuertemente tipificada es (una "Extensión") para extender dicha propiedad, dicha Extensión constituye una unidad de información discreta que se puede almacenar y manipular por dicho sistema de interfase de hardware/software donde dicha Extensión se adhiere a dicha Propiedad y donde dicha Extensión también deja de existir cuando dicha Propiedad deja de existir.

40. Un medio legible por computador que comprende instrucciones legibles por computador para organizar y consultar en forma eficiente una pluralidad de Items, dichos Items constituyen una unidad de información discreta que se puede manipular por un sistema de interfase de hardware/software, dicha instrucción legible por computador comprende instrucciones para:

crear instancia del primer Item, un segundo Item, y una primera Relación que relaciona el primer Item con el segundo Item;

crear instancia del tercer Item, dicho tercer Item siendo un subtipo de instancias de dicho primer Item, y automáticamente estableciendo una Relación heredada entre dicho tercer Item y dicho segundo Item.

41. El medio legible por computador de la reivindicación 40 comprende adicionalmente instrucciones para una instancia de un cuarto Item, dicho cuarto Item siendo curso tipo de instancias de dicho segundo Item; y

Automáticamente establécela una Relación heredada entre dicho cuarto Item y dicho primer Item.

42. El medio legible por computador de la reivindicación 41 comprende adicionalmente instrucciones para automáticamente establecer una Relación heredada entre dicho tercer Item y dicho cuarto Item.

RESUMEN

Al modelar los objetos del mundo real en aplicación con estructuras complejas, comportamientos, y operaciones descritas por un esquema que está implementado por un sistema de interfase de hardware/software, varias implementaciones del presente invento proveen una funcionalidad rica de sub tipificación extendiendo Items (y tipos de Item) utilizando "Extensiones" que proveen una estructura de datos adicional (Propiedades, Relaciones, etc.) a estructuras de tipos de Items existentes. Las Extensiones son instancias fuertemente tipificada que no pueden existir independientes y deben estar adheridas a un Item o a un Elemento Anidado. Se pretende que las Extensiones también se refieran al tema de "multi-tipificación" permitiendo que tipos de instancias se sobrepongan (por ejemplo, un Documento puede ser "un documento legal" como también un "documento seguro").

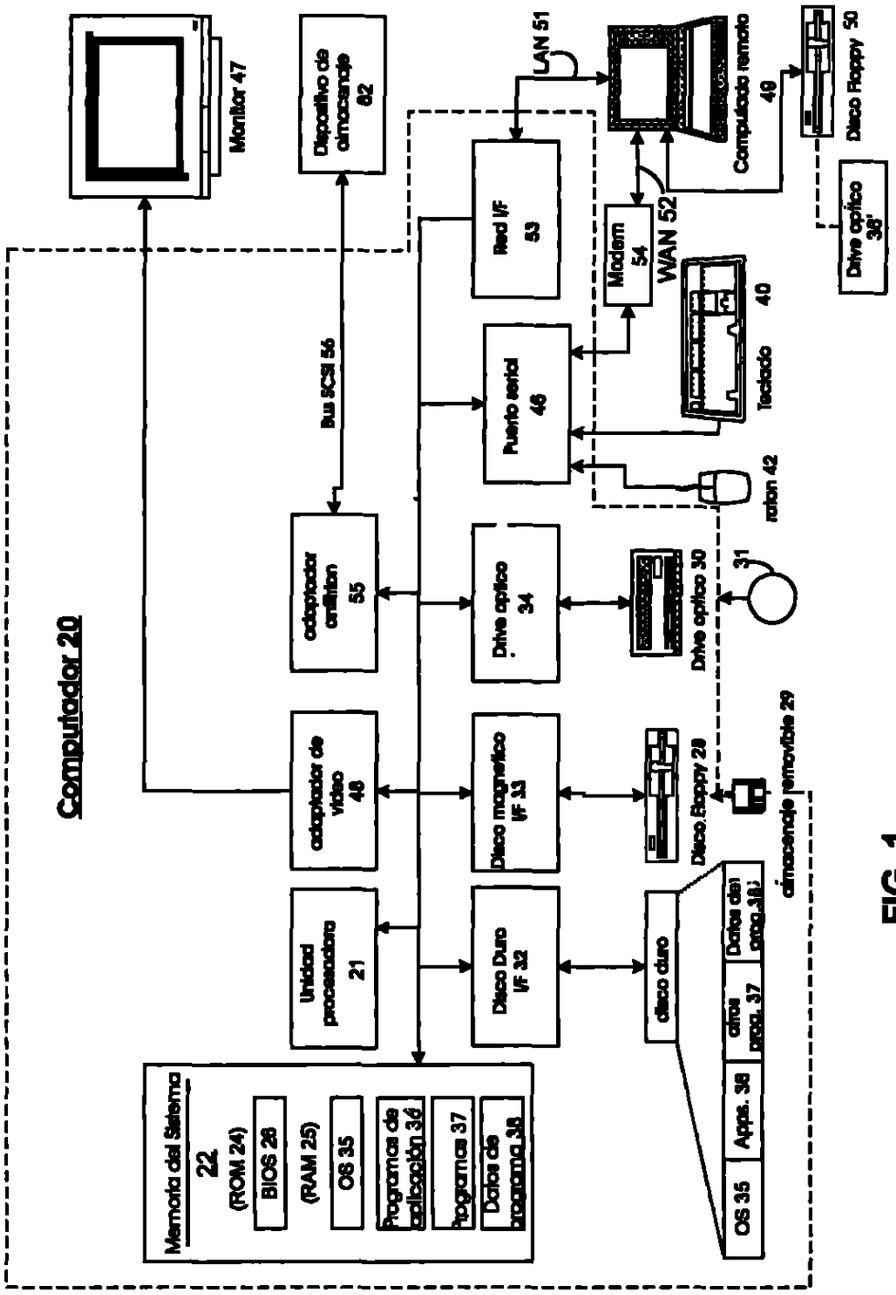

Computador 20
Monitor 47
Dispositivo de almacenaje 62
LAN 51
Computador remoto 49
Disco Floppy 50
Red I/F 53
Modem 54
WAN 52
Drive óptico 39'
Bus SCSI 56
Puerto serial 46
Teclado 40
ration 42
adaptador ardition 55
Drive óptico 34
Drive óptico 30
31
adaptador de video 48
Disco magnético I/F 33
Disco Floppy 28
Unidad procesadora 21
Disco Duro I/F 32
disco duro
almacenaje removible 29
Memoria del Sistema 22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
Programas de aplicación 36
Programas 37
Data de programa 38
OS 35
Apps. 38
data prog. 37
Data del prog. 38

FIG. 1

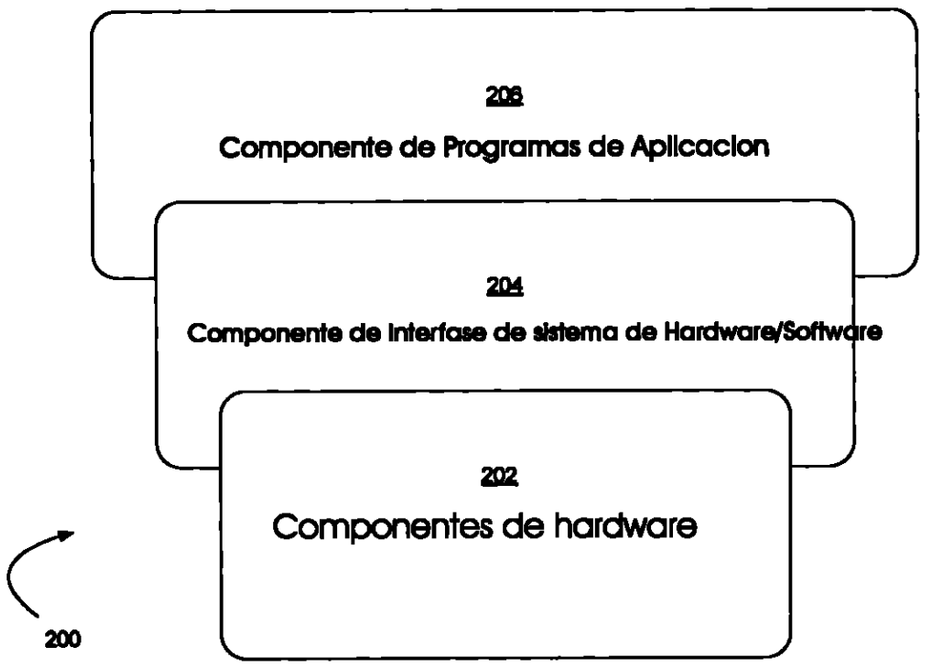

206
Componente de Programas de Aplicacion
204
Componente de Interfase de sistema de Hardware/Software
202
Componentes de hardware
200

FIG. 2

FIG. 2A

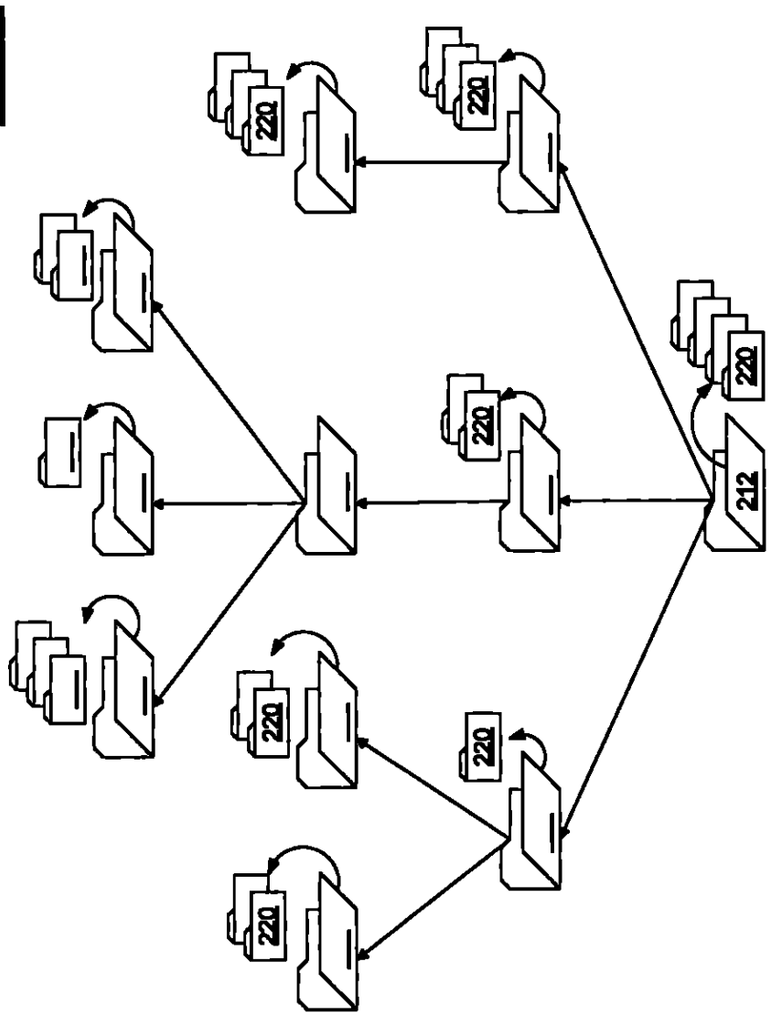


Fig. 3

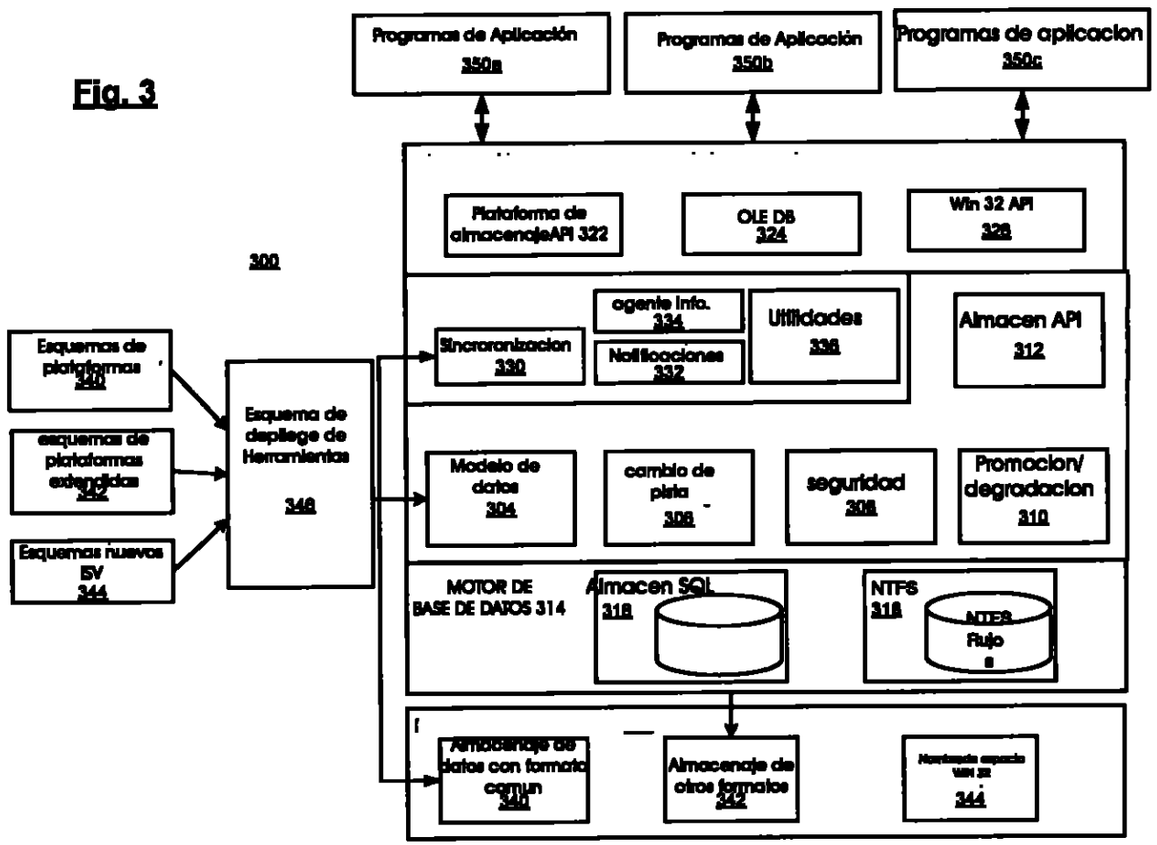

Programas de Aplicación 350a
Programas de Aplicación 350b
Programas de aplicacion 350c
Plataforma de almacenajeAPI 322
OLE DB 324
Win 32 API 328
300
agente info. 334
Utilidades 336
Almacen API 312
Sincronizacion 330
Notificaciones 332
Esquemas de plataformas 340
esquemas de plataformas extendidas 342
Esquemas nuevos ISV 344
Esquema de despliege de Herramientas 346
Modelo de datos 304
cambio de pista 306
seguridad 308
Promocion/ degradacion 310
MOTOR DE BASE DE DATOS 314
Almacen SQL 316
NTFS 316
NTFS Flujo
Almacenaje de datos con formato comun 340
Almacenaje de otros formatos 342
344

FIG. 4

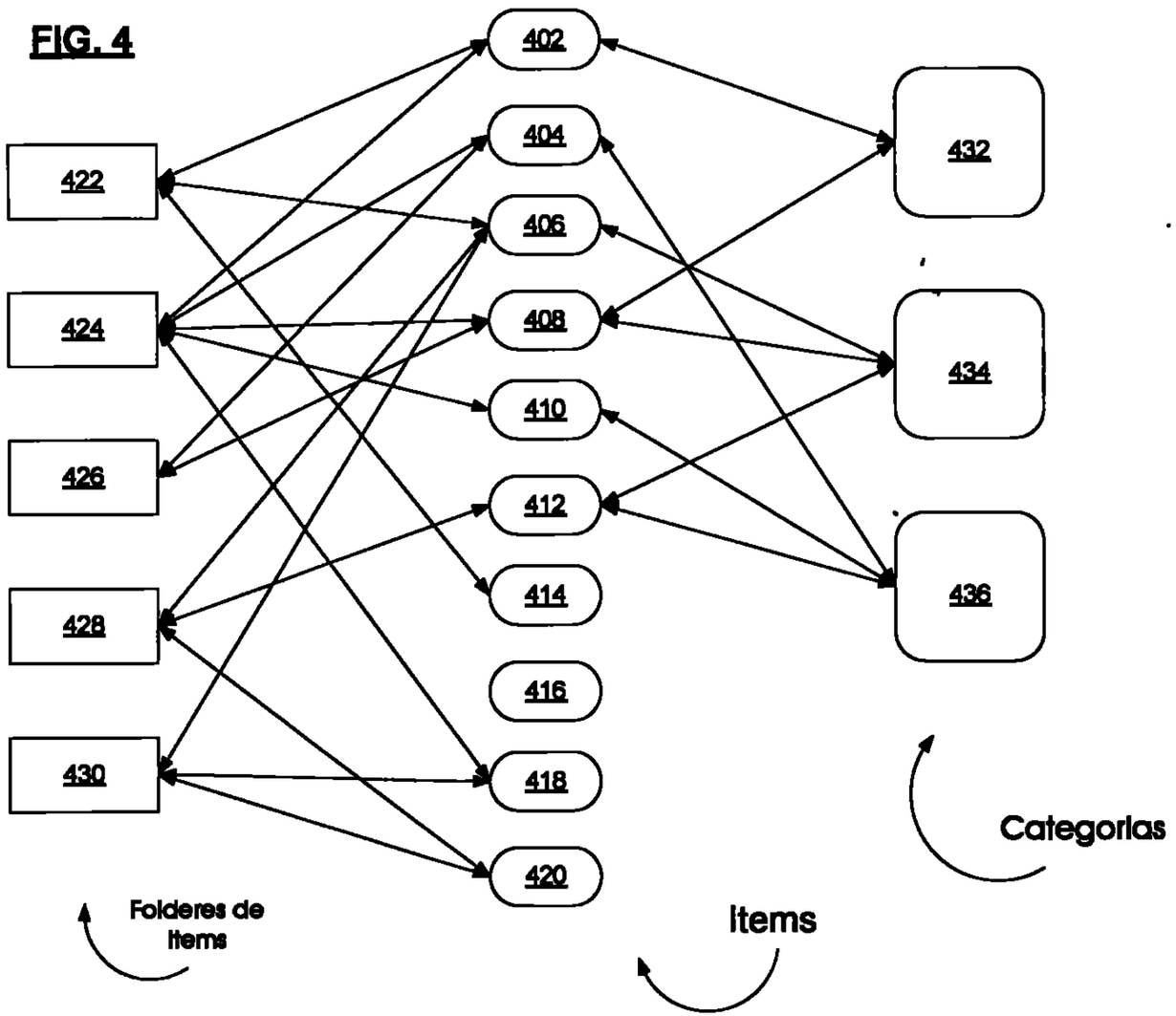

402
404
406
408
410
412
414
416
418
420
422
424
426
428
430
432
434
436
Folderes de Items
Items
Categorias

FIG. 5A

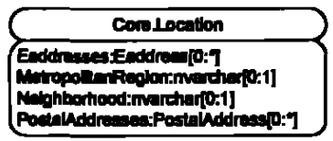


FIG. 5B

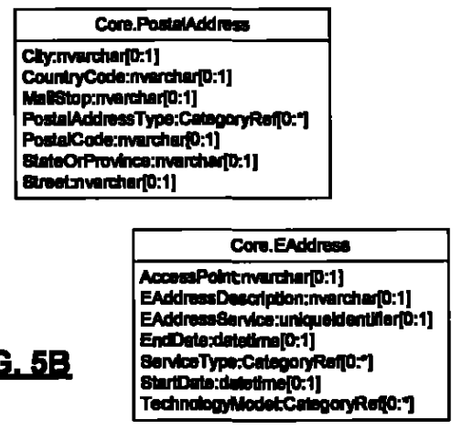


FIG. 5C

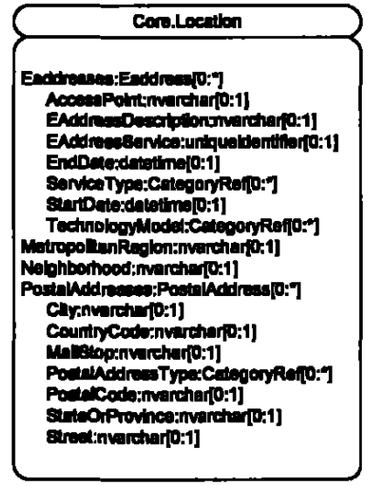
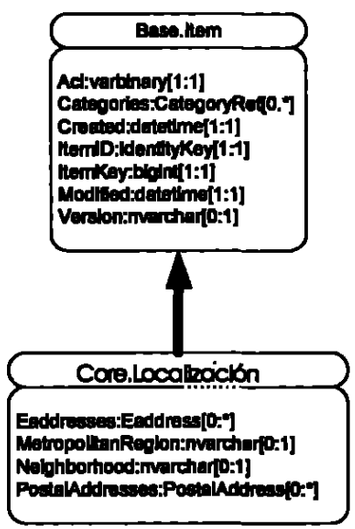


FIG. 6A


Base.Item
Acl:varbinary[1:1]
Categories:CategoryRef[0.*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Core.Localización
Eaddresses:Eaddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
PostalAddresses:PostalAddress[0:*]

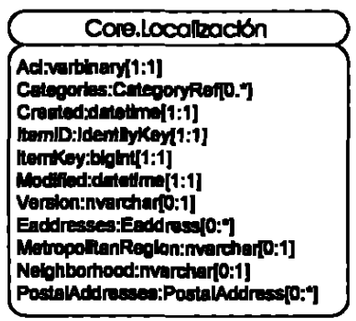

Core.Localización
Acl:varbinary[1:1]
Categories:CategoryRef[0.*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Eaddresses:Eaddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
PostalAddresses:PostalAddress[0:*]

FIG. 6B

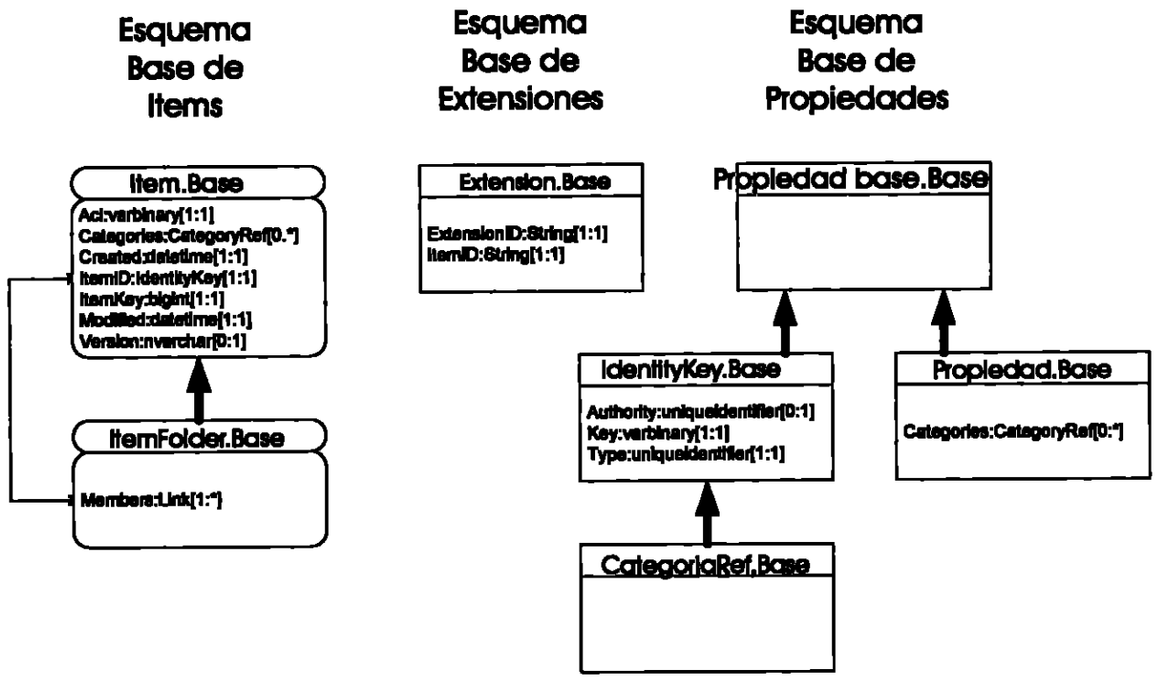

Esquema Base de Items
Esquema Base de Extensiones
Esquema Base de Propiedades
Item.Base
Acl:varbinary[1:1]
Categories:CategoryRef[0:*]
Created:datetime[1:1]
ItemID:identityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
ItemFolder.Base
Members:Link[1:*]
Extension.Base
ExtensionID:String[1:1]
ItemID:String[1:1]
Propiedad base.Base
IdentityKey.Base
Authority:uniqueidentifier[0:1]
Key:varbinary[1:1]
Type:uniqueidentifier[1:1]
Propiedad.Base
Categories:CategoryRef[0:*]
CategoriaRef.Base

FIG. 7

FIG. 8A

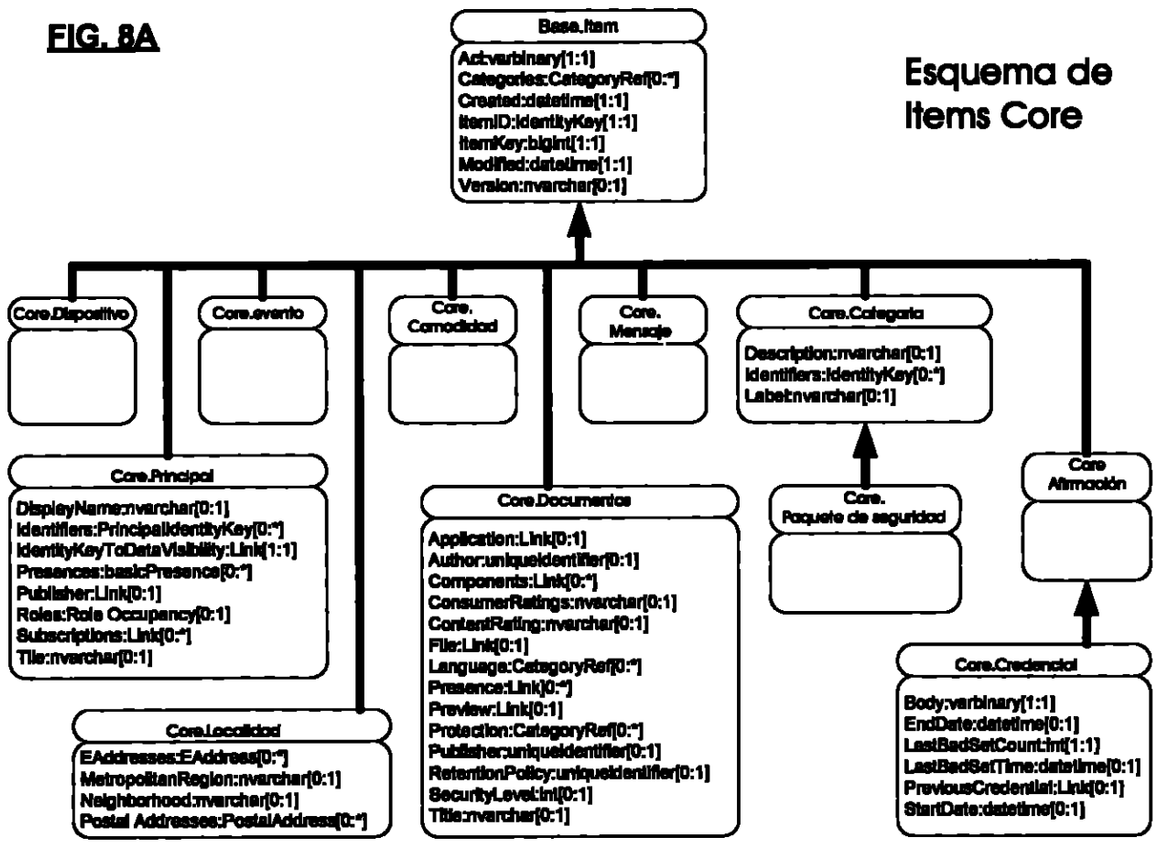
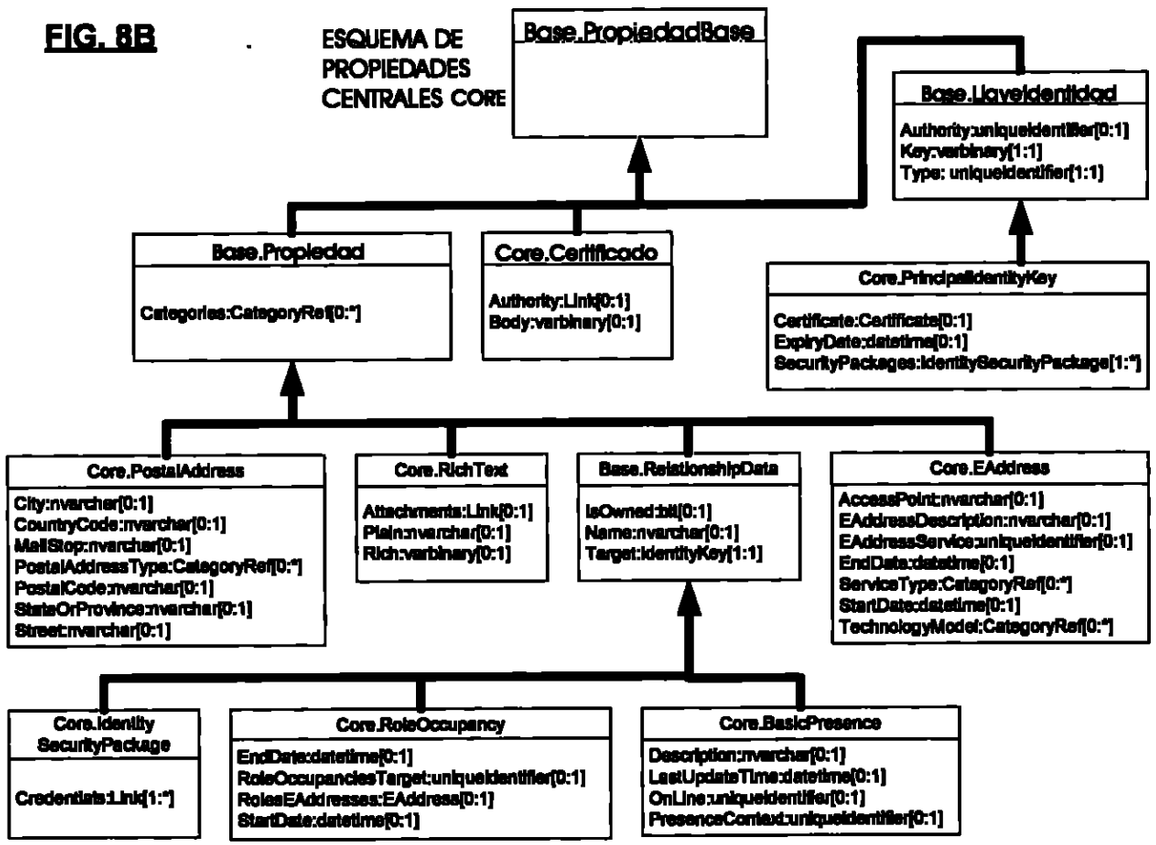

Base.Item
Act:varbinary[1:1]
Categories:CategoryRef[0:*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Esquema de Items Core
Core.Dispositivo
Core.evento
Core. Comodidad
Core. Mensaje
Core.Categoria
Description:nvarchar[0:1]
Identifiers:IdentityKey[0:*]
Label:nvarchar[0:1]
Core.Principal
DisplayName:nvarchar[0:1]
Identifiers:PrincipalIdentityKey[0:*]
IdentityKeyToDataVisibility:Link[1:1]
Presences:basicPresence[0:*]
Publisher:Link[0:1]
Roles:Role Occupancy[0:1]
Subscriptions:Link[0:*]
Tile:nvarchar[0:1]
Core.Documentos
Application:Link[0:1]
Author:uniqueIdentifier[0:1]
Components:Link[0:*]
ConsumerRatings:nvarchar[0:1]
ContentRating:nvarchar[0:1]
File:Link[0:1]
Language:CategoryRef[0:*]
Presence:Link[0:*]
Preview:Link[0:1]
Protection:CategoryRef[0:*]
Publisher:uniqueIdentifier[0:1]
RetentionPolicy:uniqueIdentifier[0:1]
SecurityLevel:int[0:1]
Title:nvarchar[0:1]
Core. Paquete de seguridad
Core Afirmación
Core.Localidad
EAddresses:EAddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
Postal Addresses:PostalAddress[0:*]
Core.Credencial
Body:varbinary[1:1]
EndDate:datetime[0:1]
LastBadSetCount:int[1:1]
LastBadSetTime:datetime[0:1]
PreviousCredential:Link[0:1]
StartDate:datetime[0:1]

FIG. 8B

ESQUEMA DE PROPIEDADES CENTRALES CORE



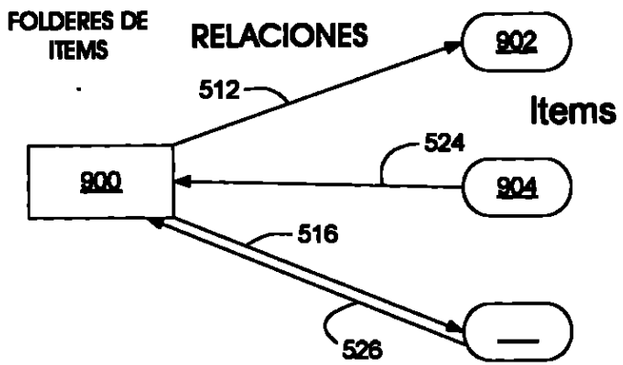
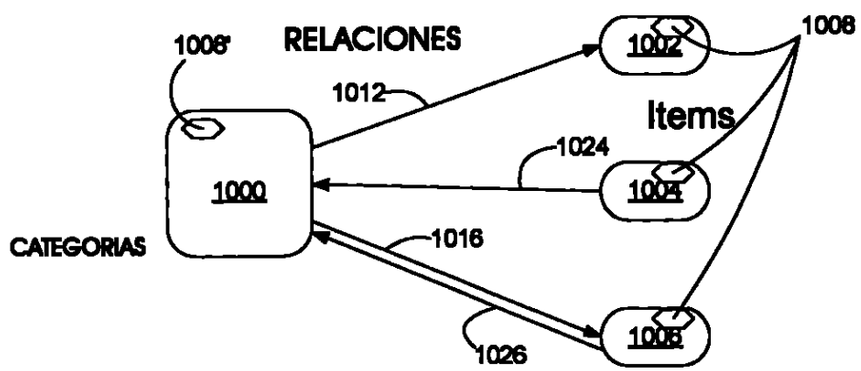

FOLDERES DE ITEMS
RELACIONES
902
512
Items
524
900
904
516
526

FIG. 9

FIG. 10


1008'
RELACIONES
1002
1008
1012
Items
1024
1000
1004
CATEGORIAS
1016
1006
1026

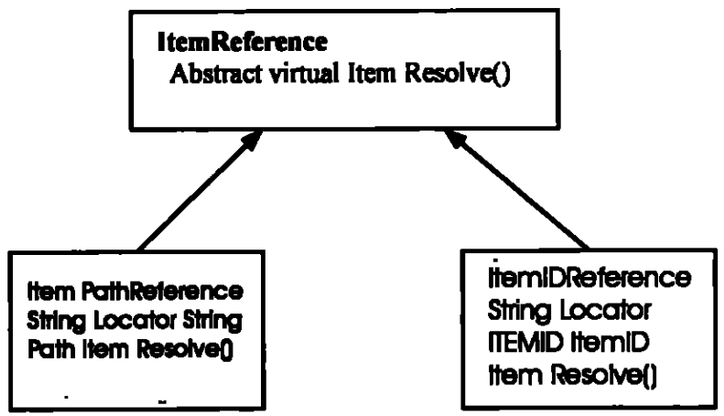

ItemReference
Abstract virtual Item Resolve()
Item PathReference
String Locator String
Path Item Resolve()
ItemIDReference
String Locator
ITEMID ItemID
Item Resolve()

FIG. 11

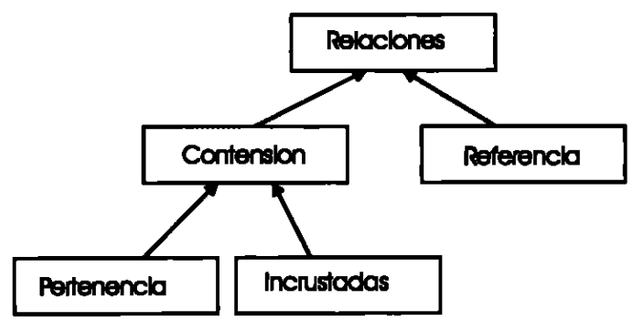

Relaciones
Contension
Referencia
Pertenencia
Incrustadas

FIG. 12

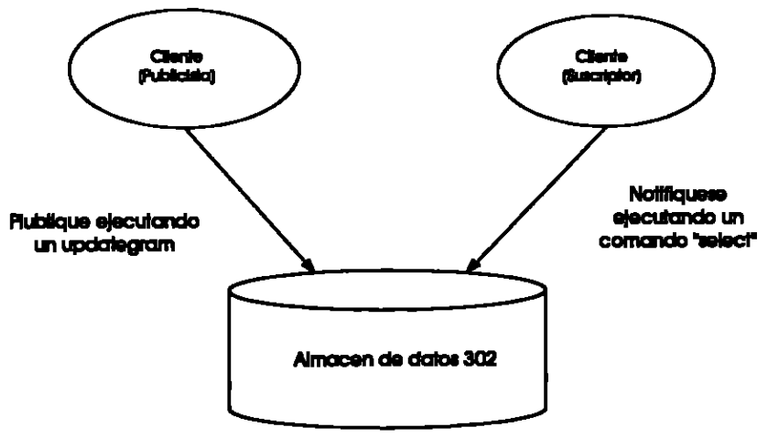

Cliente
(Publicista)
Cliente
(Suscriptor)
Publique ejecutando
un updatelegram
Notifiquese
ejecutando un
comando "select"
Almacen de datos 302

FIG. 13

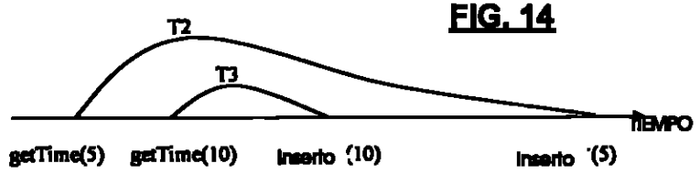

T2
T3
TIEMPO
getTime(5)
getTime(10)
Inserto (10)
Inserto (5)

FIG. 14

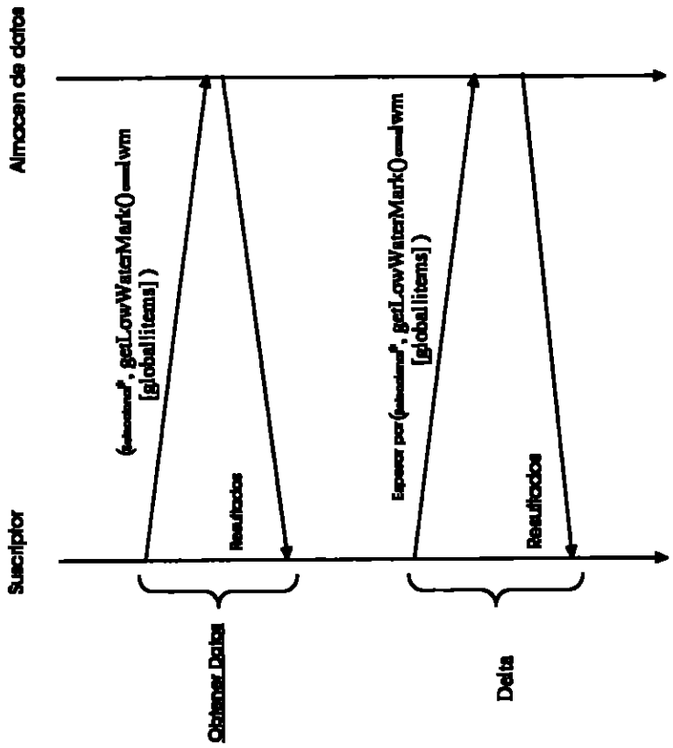

Almacen de datos
Suscriptor
Resultados
Resultados
Obtener Delta
Delta

FIG. 15

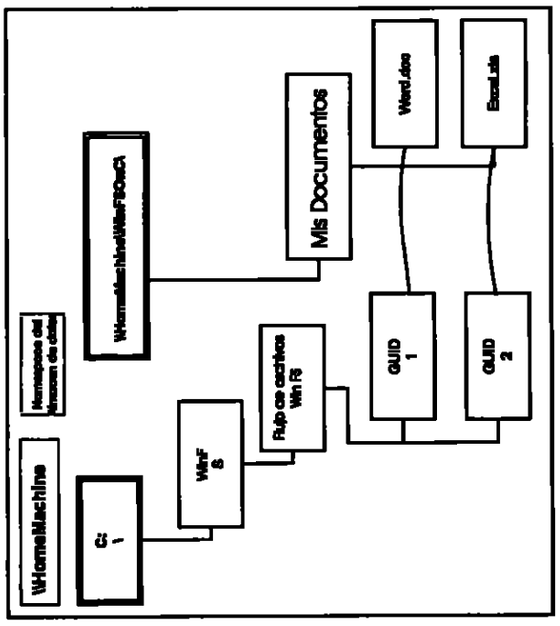

Namespace Local
WHomeMachine
C: \
Windows\ System32
Mis Documentos
Word.doc
Excel.xls

FIG. 16

Namespace del Amazon de datos
WHomeMachine
C: \
WinF S
Ruta de archivos Win FS
GUID 1
GUID 2
WHomeMachine\WinFSRoot\
Mis Documentos
Word.doc
Excel.xls

FIG. 17

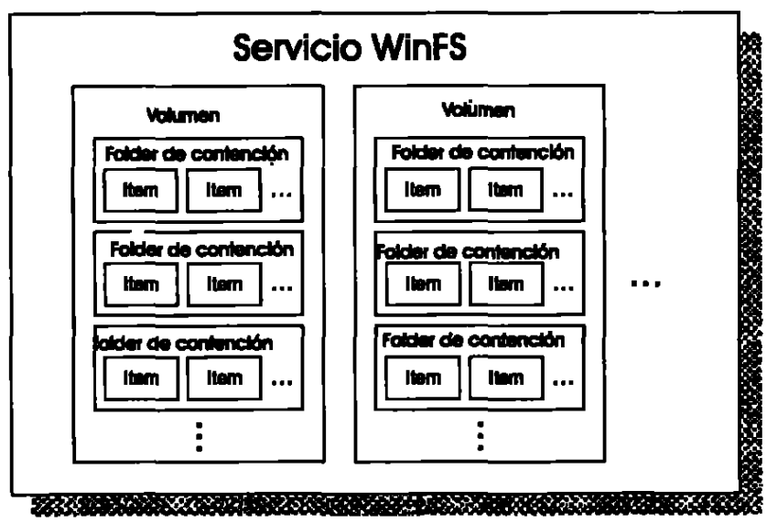

Servicio WinFS
Volumen
Folder de contención
Item
Item
...
Folder de contención
Item
Item
...
Folder de contención
Item
Item
...
Volumen
Folder de contención
Item
Item
...
Folder de contención
Item
Item
...
Folder de contención
Item
Item
...
...

FIG. 18

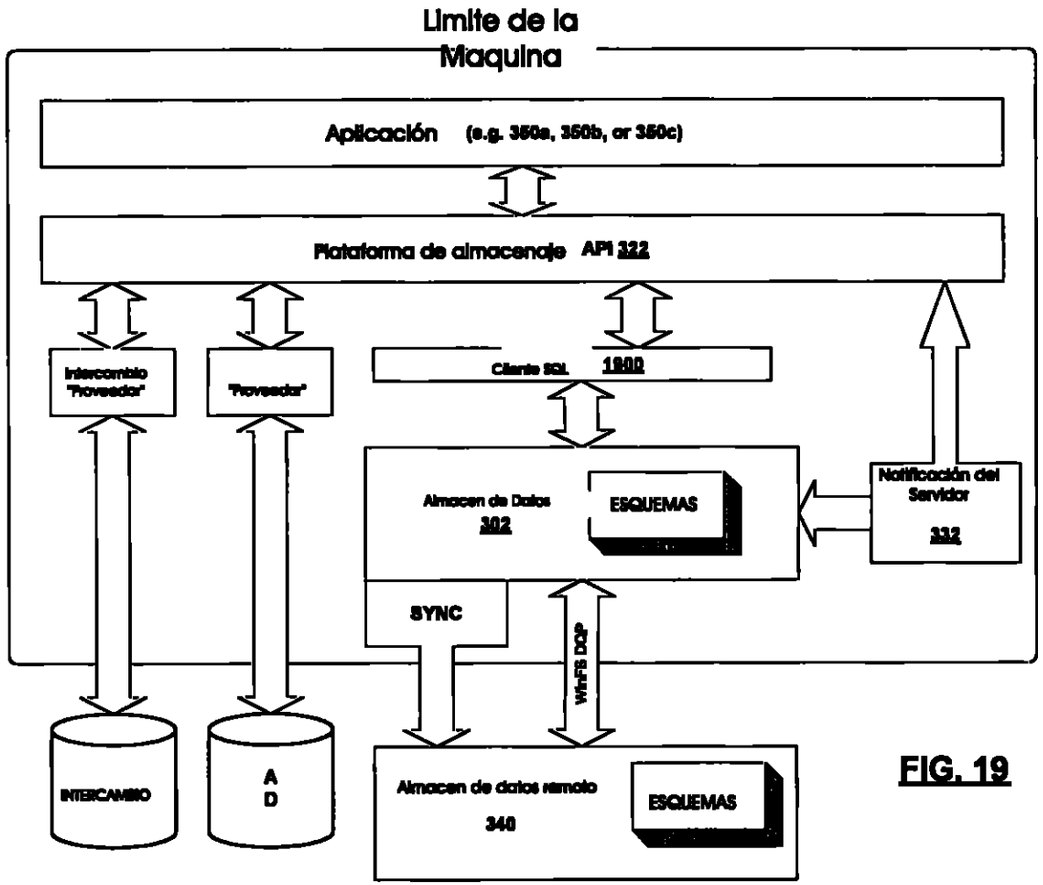

Límite de la Máquina
Aplicación (e.g. 350a, 350b, or 350c)
Plataforma de almacenaje API 322
Intercambio "Proveedor"
"Proveedor"
Cliente SQL 1900
Almacen de Datos 302
ESQUEMAS
Notificación del Servidor 332
SYNC
WinFS DQP
INTERCAMBIO
A D
Almacen de datos remoto 340
ESQUEMAS

FIG. 19

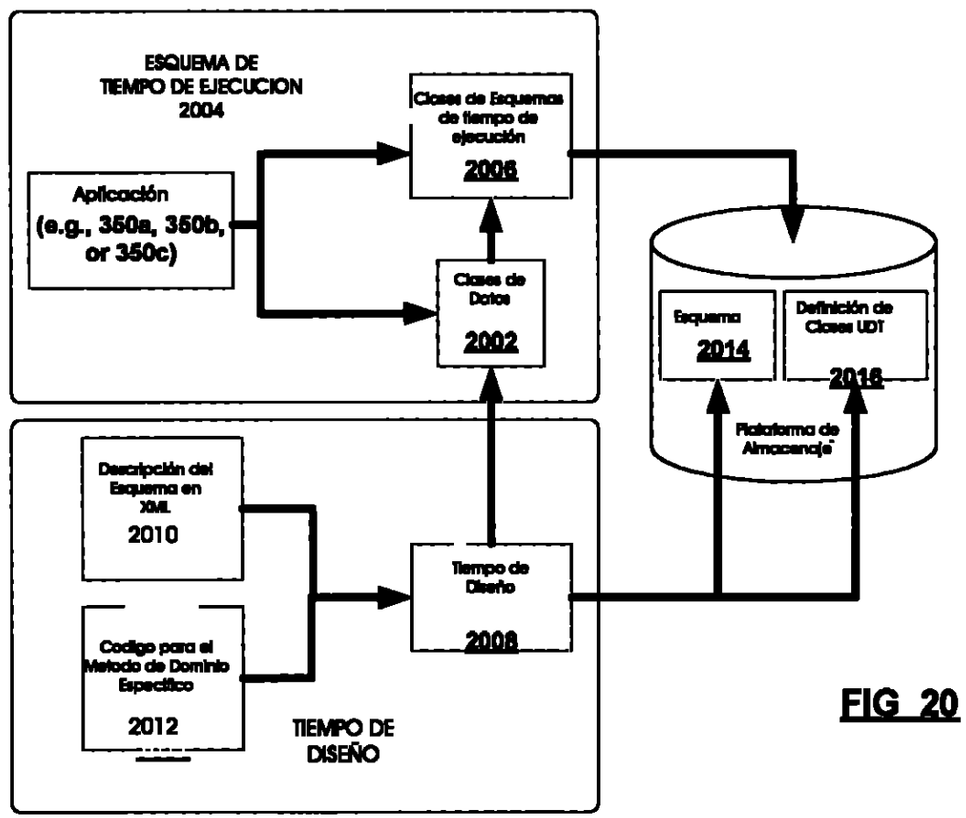

ESQUEMA DE
TIEMPO DE EJECUCION
2004
Aplicación
(e.g., 350a, 350b,
or 350c)
Clases de Esquemas de tiempo de ejecución
2006
Clases de Datos
2002
Esquema
2014
Definición de Clases UDT
2016
Plataforma de Almacenaje
Descripción del Esquema en XML
2010
Codigo para el Metodo de Dominio Especifico
2012
Tiempo de Diseño
2008
TIEMPO DE DISEÑO

FIG 20

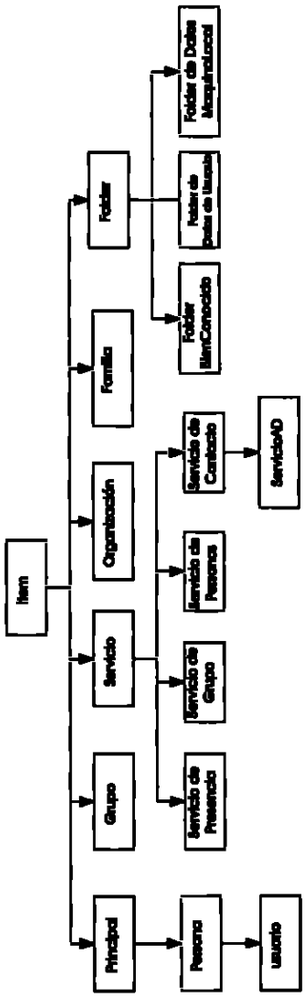

Item
Principal
Persona
Usuario
Grupo
Servicio
Servicio de Presencia
Servicio de Grupo
Organización
Servicio de Persona
Servicio de Contacto
ServicioAD
Familia
Folder
Folder de Datos Manejados Local
Folder de datos de Usuario
Folder IMenConocido

FIG. 21A

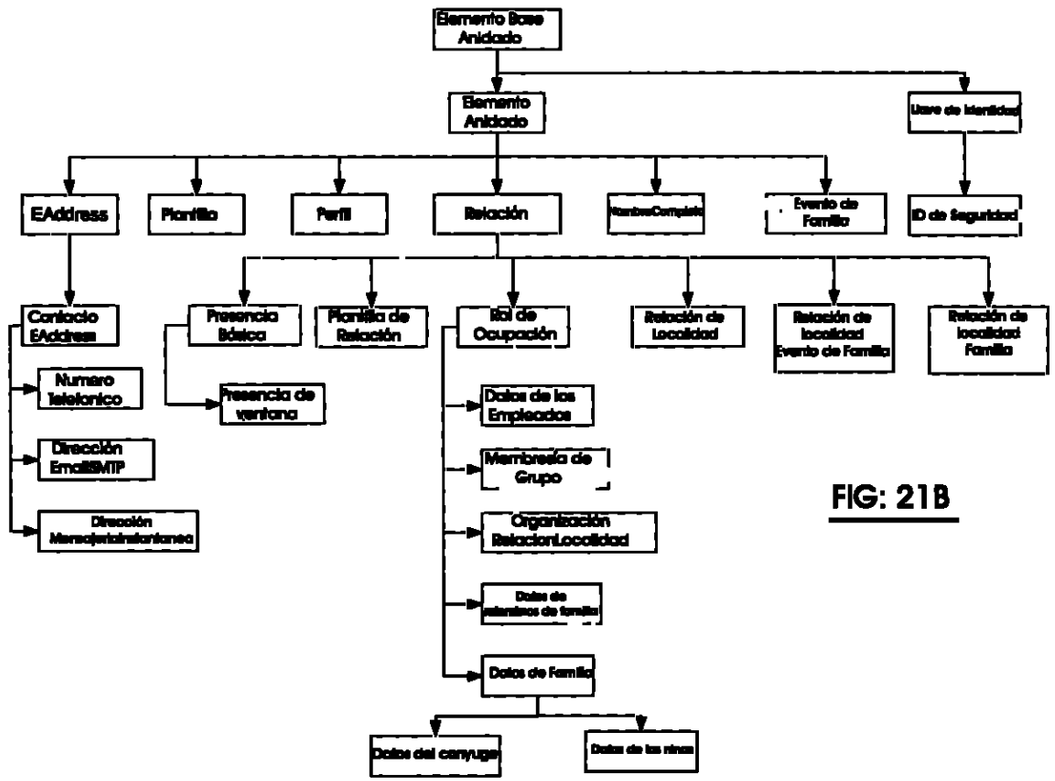

Elemento Base Anidado
Elemento Anidado
Usare de Identidad
EAddress
Plantilla
Perfil
Relación
Evento de Familia
ID de Seguridad
Contacto EAddress
Presencia Básica
Plantilla de Relación
Rol de Ocupación
Relación de Localidad
Relación de localidad Evento de Familia
Relación de localidad Familia
Numero Telefonico
Presencia de ventana
Dirección EmailSMTP
Datos de los Empleados
Membresía de Grupo
Organización RelacionLocalidad
Datos de Familia
Datos del cenyuge

FIG: 21B

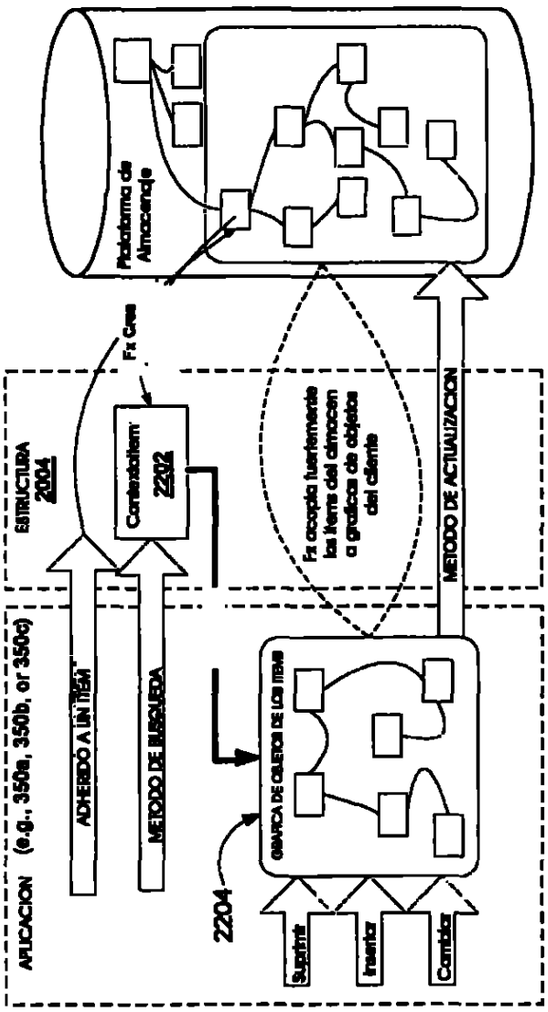

Plataforma de Almacenaje
Fx Cree
ESTRUCTURA 2004
Contenedorltem 2202
Fx acopia fuertemente las llamadas del almacen a graficos de objetos del cliente
METODO DE ACTUALIZACION
APLICACION (e.g., 350a, 350b, or 350c)
ADHERIDO A UN ITEM
METODO DE BUSQUEDA
GRAFICA DE OBJETOS DE LOS ITEMS
2204
Suprimir
Insertar
Cambiar

FIG. 22

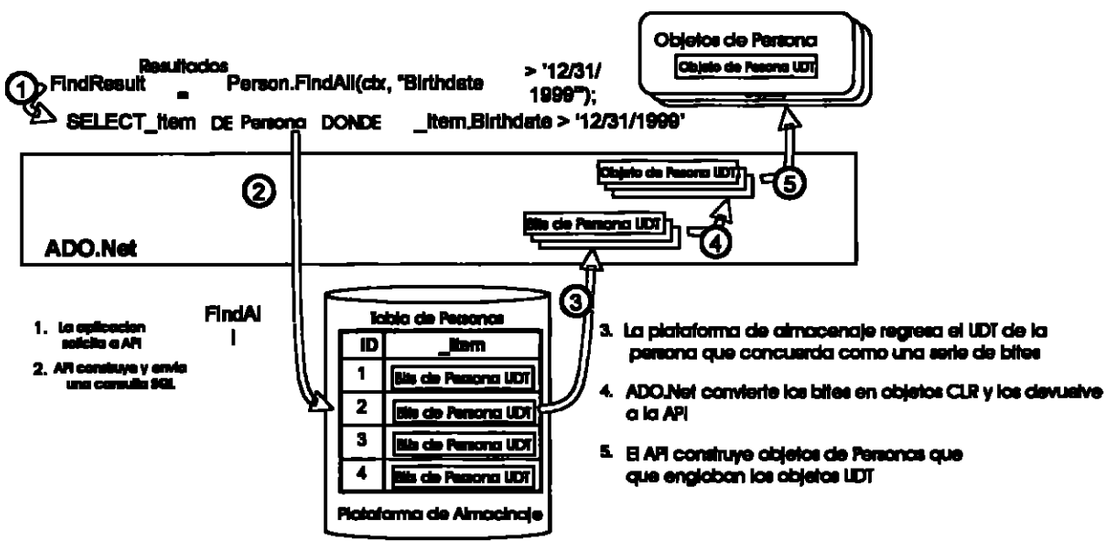

Objetos de Persona
Objeto de Persona UDT
Resultados
FindResult = Person.FindAll(ctx, "Birthdate > '12/31/1999");
SELECT_Item DE Persona DONDE _Item.Birthdate > '12/31/1999'
ADO.Net
Objeto de Persona UDT
Bits de Persona UDT
1. La aplicacion solicita a API
2. API construye y envia una consulta SQL
FindAll
Tabla de Persona
ID
_Item
1
2
3
4
Bits de Persona UDT
Plataforma de Almacenaje
3. La plataforma de almacenaje regresa el UDT de la persona que concuerda como una serie de bites
4. ADO.Net convierte los bites en objetos CLR y los devuelve a la API
5. El API construye objetos de Personas que que engloban los objetos UDT

FIG. 23

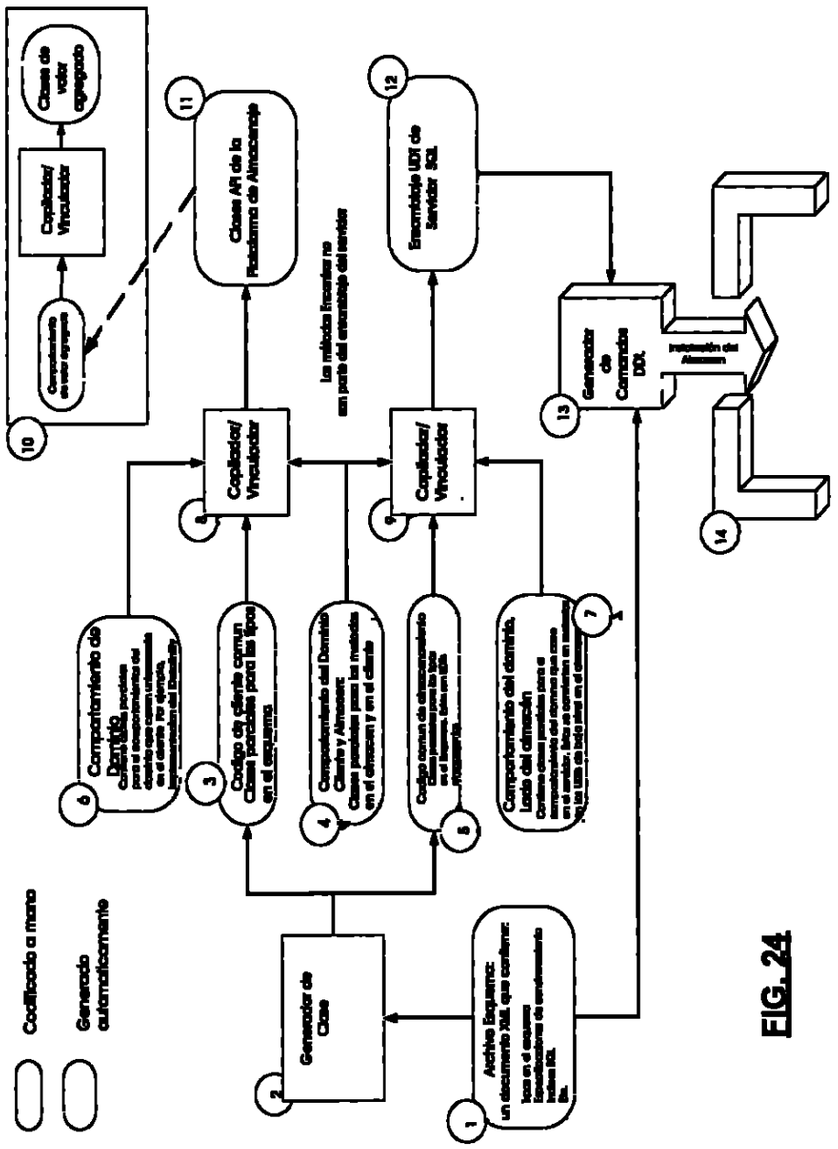
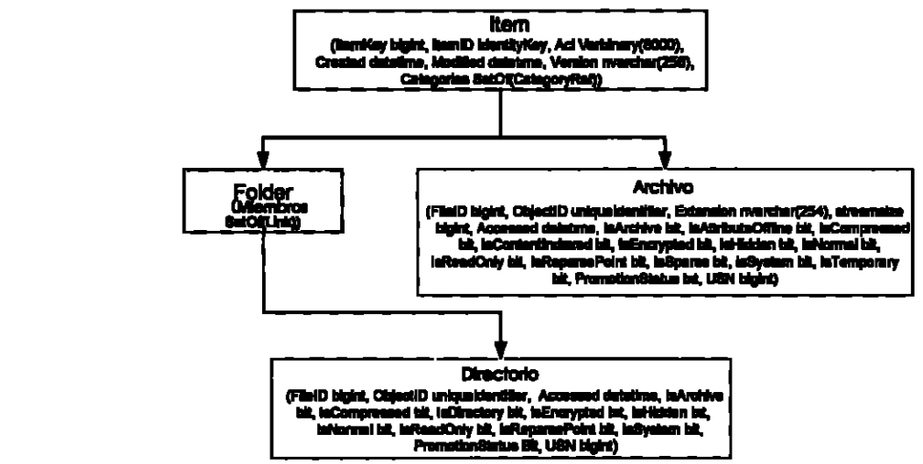

Codificado a mano
Generado automáticamente
Archivo Esquema:
Generador de Clase
Código de cliente común
Clases parciales para las tipos
en el esquema
Compartimiento del Dominio
Clases parciales para los métodos
en el dominio y en el cliente
Código común de almacenamiento
Clases parciales para los tipos
en el lienzo, bien en
Compartimiento del dominio.
Lado del dominio
Copiador/
Vinculador
Copiador/
Vinculador
Clase AR de la
Plataforma de Almacenaje
Ensamblaje UDI de
Servidor SQL
Generador
de
Comandos
UDI
Clase de
valor
agregado
Copiador/
Vinculador
Los métodos Ensamble no
son parte del ensamblaje del servidor

FIG. 24



Nota: Los cuadros grises son tipos definidos en el Esquema Basico, pero se muestran aca para mayor claridad

Items definidos en el Esquema de Archivos

FIG. 25



GR —> Generic_Read
GW —> Generic_Write
GE —> Generic_Execute
GA —> Generic_ALL
AS —> Right to access SACL

FIG. 26

128

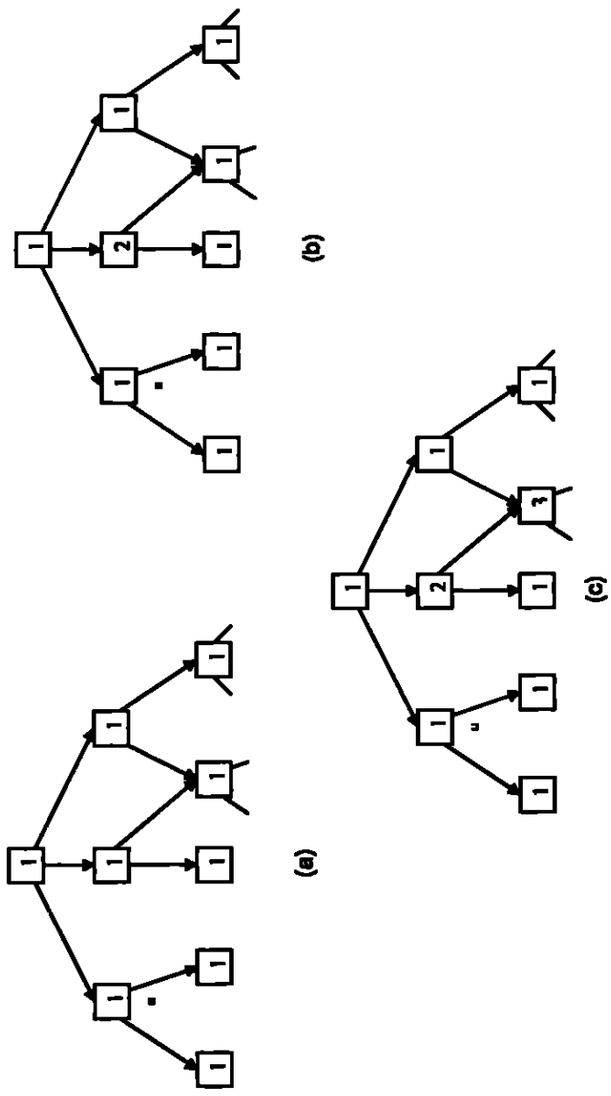


FIG. 27

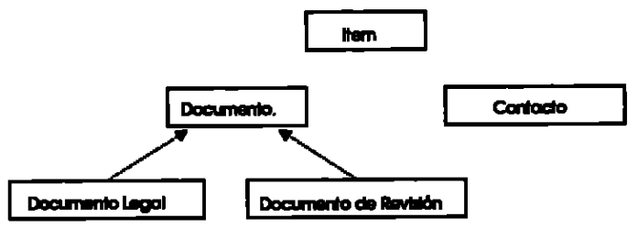

Item
Documento.
Contacto
Documento Legal
Documento de Revisión

FIG. 28

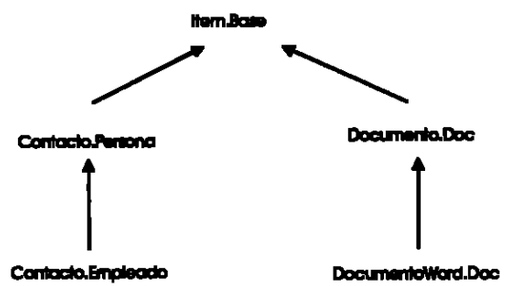

Item.Base
Contacto.Persona
Documento.Doc
Contacto.Empleado
DocumentoWord.Doc

FIG. 29

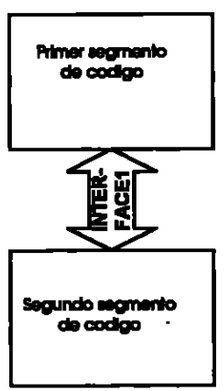


FIG. 30A

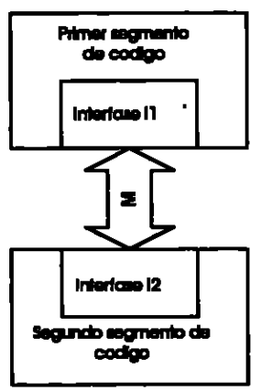


FIG. 30B

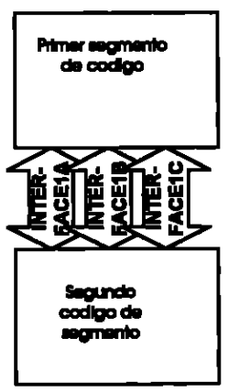


FIG. 31A

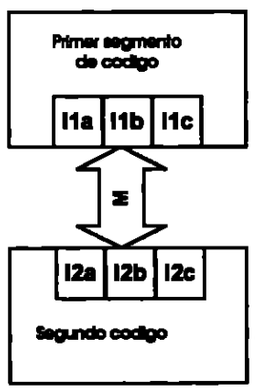


FIG. 31B

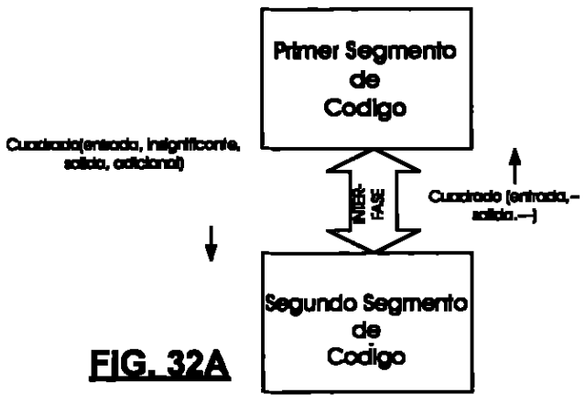


FIG. 32A

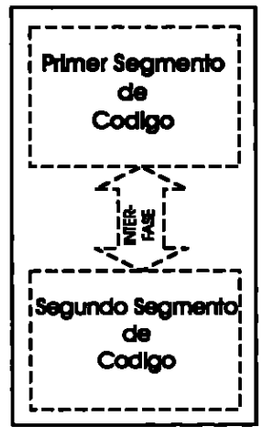


FIG. 33A

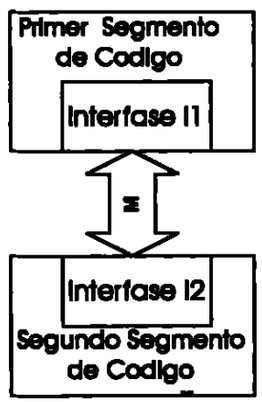


FIG. 32B

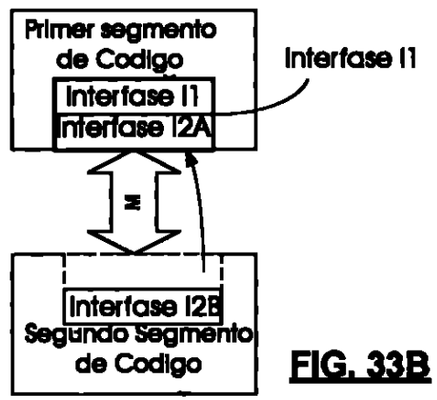


FIG. 33B

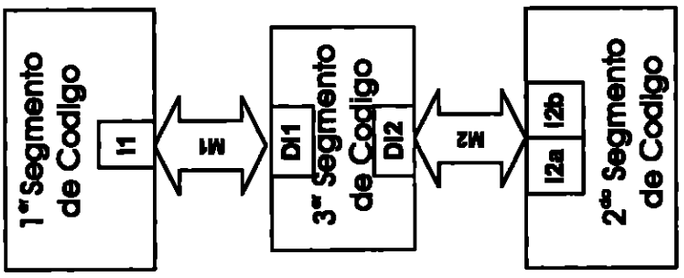


FIG. 34B

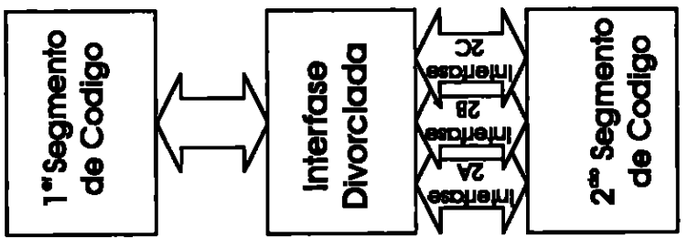


FIG. 34A

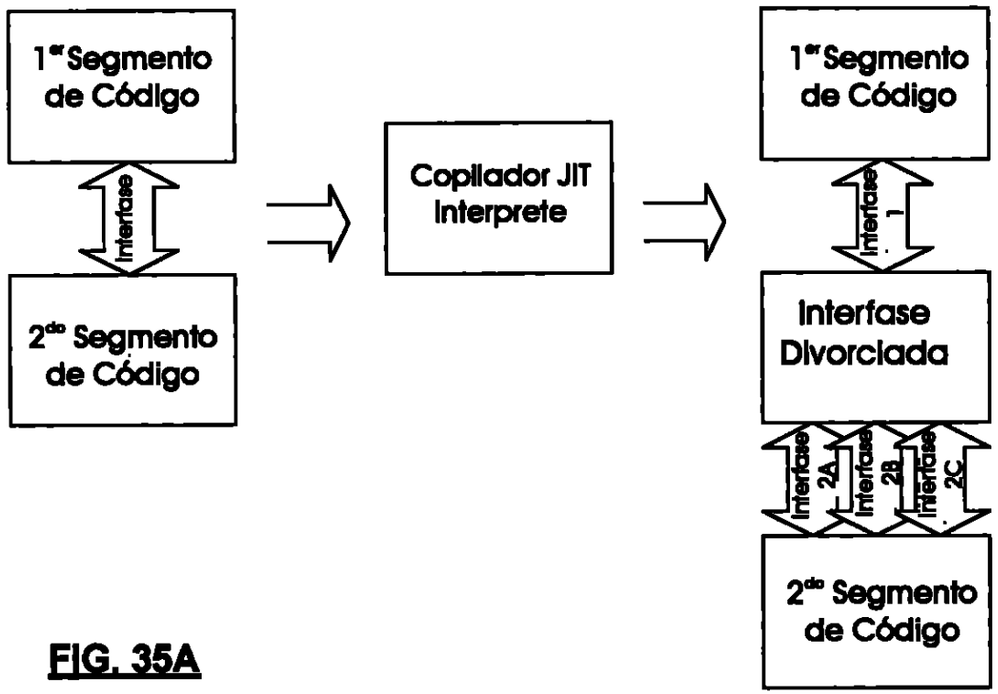

1er Segmento de Código
Interfase
2do Segmento de Código
Copilador JIT Interprete
1er Segmento de Código
Interfase 1
Interfase Divorciada
Interfase 2A
Interfase 2B
Interfase 2C
2do Segmento de Código

FIG. 35A

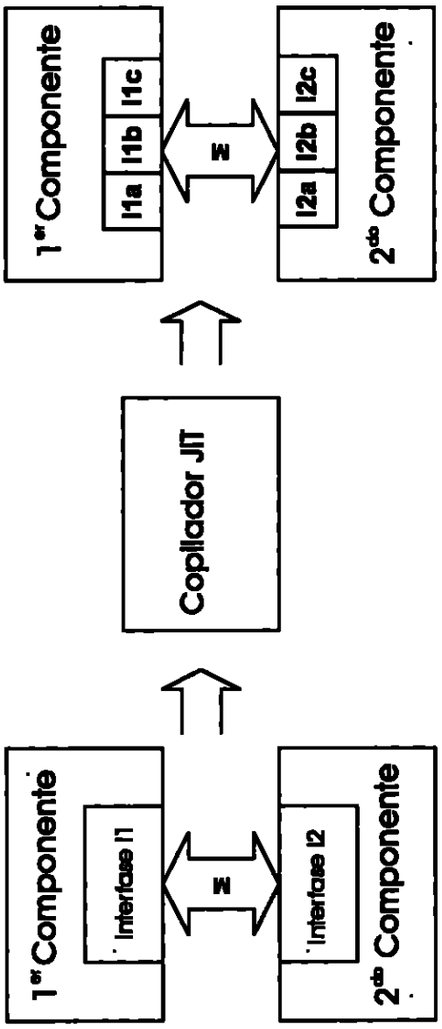

1er Componente
Interface I1
2do Componente
Interface I2
Copilador JIT
1er Componente
I1a
I1b
I1c
2do Componente
I2a
I2b
I2c

FIG. 35B

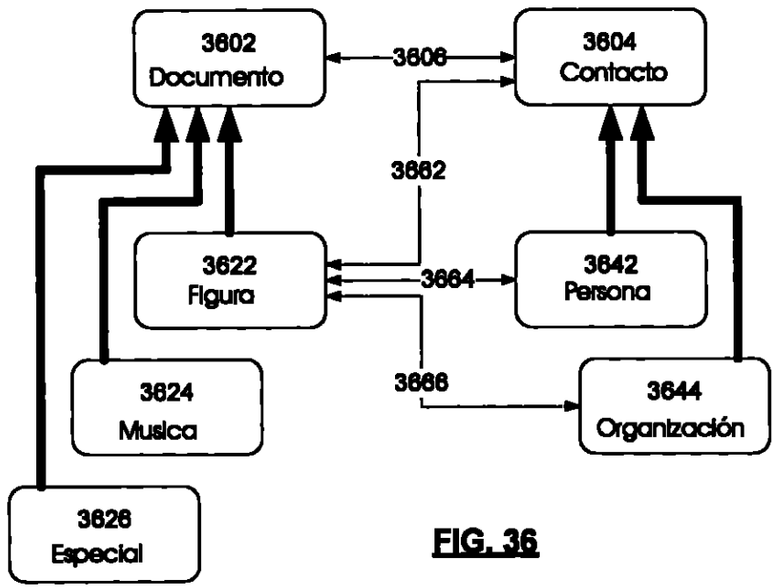

3602
Documento
3606
3604
Contacto
3662
3622
Figura
3664
3642
Persona
3624
Musica
3666
3644
Organización
3626
Especial

FIG. 36


Llave
Herencia de Item
Designada
Relación

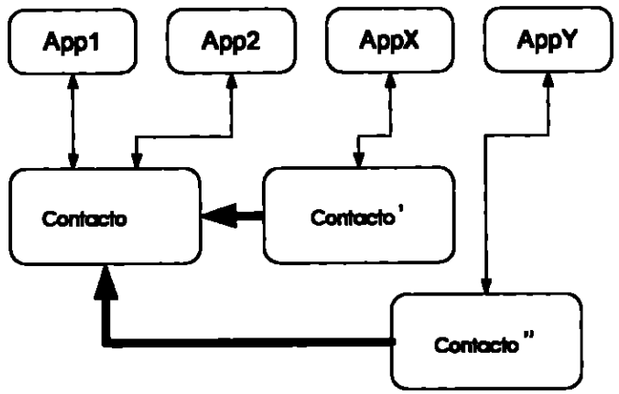


FIG. 37A

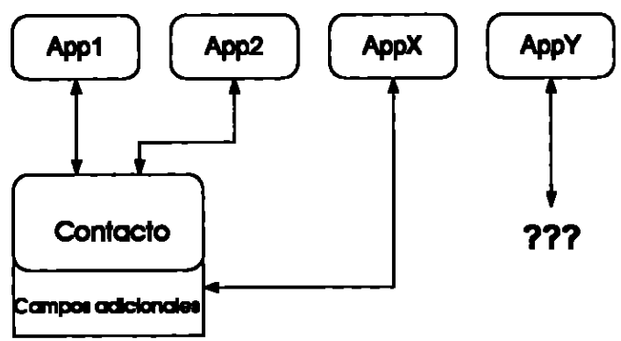


FIG. 37B

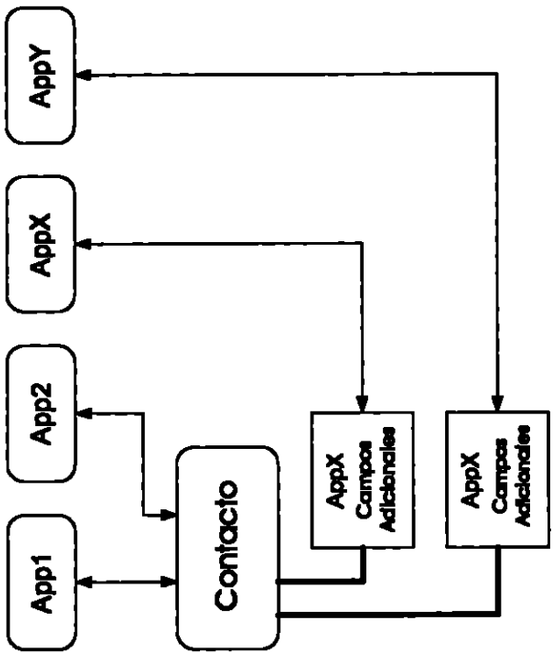

AppY
AppX
App2
App1
Contacto
AppX Campos Adicionales
AppX Campos Adicionales

FIG. 37C

PCT Request (5)

PCT

REQUEST

The undersigned requests that the present international application be processed according to the Patent Cooperation Treaty.

For receiving Office use only

International Application No.

International Filing Date

Name of receiving Office and "PCT International Application"

189

Applicant's or agent's file reference *(if desired) (12 characters maximum)* MSFT-4419

Box No. I TITLE OF INVENTION

SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM

Box No. II APPLICANT ☐ This person is also inventor

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

Telephone No.

Facsimile No.

Teleprinter No.

Applicant's registration No. with the Office

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☒ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box

Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

DEMIROSKI, Bekim
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
☐ applicant only
☒ applicant and inventor
☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* of nationality: MK

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on a continuation sheet.

Box No. IV AGENT OR COMMON REPRESENTATIVE; OR ADDRESS FOR CORRESPONDENCE

The person identified below is hereby/has been appointed to act on behalf of the applicant(s) before the competent International Authorities as: ☒ agent ☐ common representative

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country.)*

ROCCI, Steven J.; KNIGHT, Richard W.
WOODCOCK WASHBURN LLP
One Liberty Place, 46th Floor
Philadelphia, Pennsylvania 19103
UNITED STATES OF AMERICA

Telephone No. 215-568-3100

Facsimile No. 215-568-3439

Teleprinter No.

Agent's registration No. with the Office 30,489; 42,751

☒ Address for correspondence: Mark this check-box where no agent or common representative is/has been appointed and the space above is used instead to indicate a special address to which correspondence should be sent.

10

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S) 190

If none of the following sub-boxes is used, this sheet should not be included in the request.

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

WHITNEY, Robert T.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

THOMPSON, J. Patrick
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: GB

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

NORI, Anil K.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

AGARWAL, Sameet
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: IN

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on another continuation sheet.

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

If none of the following sub-boxes is used, this sheet should not be included in the request.

PR1

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

CELIS, Pedro
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** US

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

CAMPBELL, David G.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** US

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

TEREK, F. Soner
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** TR

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

CAMERON, Kim
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** CA

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on another continuation sheet.

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

If none of the following sub-boxes is used, this sheet should not be included in the request.

192

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

SMITH, Walter R.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

SHAKIB, Darren A.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

BALLOU, Nathaniel H.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

ACHARYA, Srinivasmurthy P.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office:

State *(that is, country)* of nationality: US

State *(that is, country)* of residence: US

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on another continuation sheet.

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S) 193

If none of the following sub-boxes is used, this sheet should not be included in the request.

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

RAMAN, Balan Sethu
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** US

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☒ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

SPIRO, Peter M.
c/o MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
United States of America

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:** US

State *(that is, country)* **of residence:** US

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☒ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
- ☐ applicant only
- ☐ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:**

State *(that is, country)* **of residence:**

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☐ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
- ☐ applicant only
- ☐ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

Applicant's registration No. with the Office

State *(that is, country)* **of nationality:**

State *(that is, country)* **of residence:**

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☐ the United States of America only
- ☐ the States indicated in the Supplemental Box

☐ Further applicants and/or (further) inventors are indicated on another continuation sheet.

24

194

Box No. V DESIGNATIONS

The filing of this request constitutes under Rule 4.9(a), the designation of all Contracting States bound by the PCT on the international filing date, for the grant of every kind of protection available and, where applicable, for the grant of both regional and national patents.

However,

☐ DE Germany is not designated for any kind of national protection

☐ KR Republic of Korea is not designated for any kind of national protection

☐ RU Russian Federation is not designated for any kind of national protection

(The check-boxes above may be used to exclude (irrevocably) the designations concerned in order to avoid the ceasing of the effect, under the national law, of an earlier national application from which priority is claimed. See the Notes to Box No. V as to the consequences of such national law provisions in these and certain other States.)

Box No. VI PRIORITY CLAIM

The priority of the following earlier application(s) is hereby claimed:

Filing date of earlier application *(day/month/year)*	Number of earlier application	Where earlier application is:		
		national application: country or Member of WTO	regional application:* regional Office	international application: receiving Office
item (1) 21 August 2003 21.08.03	10/646,580	US		
item (2) 21 August 2003 21.08.03	PCT/US03/26144			US
item (3) 24 October 2003 24.10.03	10/693,574	US		

☐ Further priority claims are indicated in the Supplemental Box.

The receiving Office is requested to prepare and transmit to the International Bureau a certified copy of the earlier application(s) *(only if the earlier application was filed with the Office which for the purposes of this international application is the receiving Office)* identified above as:

☒ all items ☐ item (1) ☐ item (2) ☐ item (3) ☐ other, see Supplemental Box

** Where the earlier application is an ARIPO application, indicate at least one country party to the Paris Convention for the Protection of Industrial Property or one Member of the World Trade Organization for which that earlier application was filed (Rule 4.10(b)(ii)):* .....

Box No. VII INTERNATIONAL SEARCHING AUTHORITY

Choice of International Searching Authority (ISA) *(if two or more International Searching Authorities are competent to carry out the international search, indicate the Authority chosen; the two-letter code may be used)*:

ISA / US

Request to use results of earlier search; reference to that search *(if an earlier search has been carried out by or requested from the International Searching Authority)*:

Date *(day/month/year)*	Number	Country *(for regional Office)*
21 August 2003 & 24 October 2003	10/646,580 & 10/693,57	US

Box No. VIII DECLARATIONS

The following declarations are contained in Boxes Nos. VIII (i) to (v) *(mark the applicable check-boxes below and indicate in the right column the number of each type of declaration)*:		Number of declarations
☐ Box No. VIII (i)	Declaration as to the identity of the inventor :	
☐ Box No. VIII (ii)	Declaration as to the applicant's entitlement, as at the international filing date, to apply for and be granted a patent :	
☐ Box No. VIII (iii)	Declaration as to the applicant's entitlement, as at the international filing date, to claim the priority of the earlier application :	
☐ Box No. VIII (iv)	Declaration of inventorship (only for the purposes of the designation of the United States of America) :	
☐ Box No. VIII (v)	Declaration as to non-prejudicial disclosures or exceptions to lack of novelty :	

195

Box No. IX CHECK LIST; LANGUAGE OF FILING

This international application contains:

(a) **in paper form, the following number of sheets:**

request (including declaration sheets)	:	7
description (excluding sequence listing and/or tables related thereto)	:	110
claims	:	7
abstract	:	1
drawings	:	35
Sub-total number of sheets	:	160
sequence listing	:	
tables related thereto	:	
(for both, actual number of sheets if filed in paper form, whether or not also filed in computer readable form; see (c) below)		
Total number of sheets	:	160

(b) ☐ **only in computer readable form (Section 801(a)(i))**
- (i) ☐ sequence listing
- (ii) ☐ tables related thereto

(c) ☐ **also in computer readable form (Section 801(a)(ii))**
- (i) ☐ sequence listing
- (ii) ☐ tables related thereto

Type and number of carriers (diskette, CD-ROM, CD-R or other) on which are contained the
- ☐ sequence listing: .....
- ☐ tables related thereto: .....

(additional copies to be indicated under items 9(ii) and/or 10(ii), in right column)

This international application is accompanied by the following item(s) *(mark the applicable check-boxes below and indicate in right column the number of each item)*: — **Number of items**

1. ☒ fee calculation sheet : 1
2. ☐ original separate power of attorney :
3. ☐ original general power of attorney :
4. ☒ copy of general power of attorney; reference number, if any: ..... : 1
5. ☐ statement explaining lack of signature :
6. ☐ priority document(s) identified in Box No. VI as item(s): ..... :
7. ☐ translation of international application into *(language)*: ..... :
8. ☐ separate indications concerning deposited microorganism or other biological material :
9. ☐ sequence listing in computer readable form *(indicate type and number of carriers)*
   - (i) ☐ copy submitted for the purposes of international search under Rule 13*ter* only (and not as part of the international application) :
   - (ii) ☐ *(only where check-box (b)(i) or (c)(i) is marked in left column)* additional copies including, where applicable, the copy for the purposes of international search under Rule 13*ter* :
   - (iii) ☐ together with relevant statement as to the identity of the copy or copies with the sequence listing mentioned in left column :
10. ☐ tables in computer readable form related to sequence listing *(indicate type and number of carriers)*
    - (i) ☐ copy submitted for the purposes of international search under Section 802(b-*quater*) only (and not as part of the international application) :
    - (ii) ☐ *(only where check-box (b)(ii) or (c)(ii) is marked in left column)* additional copies including, where applicable, the copy for the purposes of international search under Section 802(b-*quater*) :
    - (iii) ☐ together with relevant statement as to the identity of the copy or copies with the tables mentioned in left column :
11. ☒ other *(specify)*: .Transmittal Letter, cheque, postcard.. : 3

Figure of the drawings which should accompany the abstract: 36	**Language of filing of the international application:** English

Box No. X SIGNATURE OF APPLICANT, AGENT OR COMMON REPRESENTATIVE

Next to each signature, indicate the name of the person signing and the capacity in which the person signs (if such capacity is not obvious from reading the request).

WOODCOCK WASHBURN LLP

Steven B. Samuels
Agent for Applicant

For receiving Office use only

1. Date of actual receipt of the purported international application:
2. Drawings:
   - ☐ received:
   - ☐ not received:
3. Corrected date of actual receipt due to later but timely received papers or drawings completing the purported international application:
4. Date of timely receipt of the required corrections under PCT Article 11(2):
5. International Searching Authority (if two or more are competent): ISA /
6. ☐ Transmittal of search copy delayed until search fee is paid

For International Bureau use only

Date of receipt of the record copy by the International Bureau:

196

196

This sheet is not part of and does not count as a sheet of the international application.

PCT

FEE CALCULATION SHEET

Annex to the Request

For receiving Office use only

International Application No.

Date stamp of the receiving Office

Applicant's or agent's file reference: MSFT-4419

Applicant: MICROSOFT CORPORATION

CALCULATION OF PRESCRIBED FEES

1. TRANSMITTAL FEE . . . . . US 300.00 [T]

2. SEARCH FEE . . . . . US 300.00 [S]

International search to be carried out by: ISA/US

(If two or more International Searching Authorities are competent to carry out the international search, indicate the name of the Authority which is chosen to carry out the international search.)

3. INTERNATIONAL FILING FEE

Where items (b) and/or (c) of Box No. IX apply, enter Sub-total number of sheets / Where items (b) and (c) of Box No. IX do not apply, enter Total number of sheets: 160

[i1] first 30 sheets . . . . . US 1,134.00 [i1]

[i2] 130 (number of sheets in excess of 30) x 12.00 (fee per sheet) = US 1,580. [i2]

[i3] additional component (only if sequence listing and/or tables related thereto are filed in computer readable form under Section 801(a)(i), or both in that form and on paper, under Section 801(a)(ii)):

400 x _____ (fee per sheet) = _____ [i3]

Add amounts entered at i1, i2 and i3 and enter total at I . . . . US 2,924.00 [I]

(Applicants from certain States are entitled to a reduction of 75% of the international filing fee. Where the applicant is (or all applicants are) so entitled, the total to be entered at I is 25% of the international filing fee.)

4. FEE FOR PRIORITY DOCUMENT *(if applicable)* . . . . . US 40.00 [P]

5. TOTAL FEES PAYABLE . . . . . US3,564.00 TOTAL

Add amounts entered at T, S, I and P, and enter total in the TOTAL box

MODE OF PAYMENT

- [X] authorization to charge deposit account (see below)
- [ ] postal money order
- [ ] cash
- [ ] coupons
- [X] cheque
- [ ] bank draft
- [ ] revenue stamps
- [ ] other *(specify)*:

AUTHORIZATION TO CHARGE (OR CREDIT) DEPOSIT ACCOUNT

(This mode of payment may not be available at all receiving Offices)

Receiving Office: RO/ US

Deposit Account No.: 23-3050

Date: 29 July 2004

Name: Steven B. Samuels

Signature: _____

- [ ] Authorization to charge the total fees indicated above.
- [X] *(This check-box may be marked only if the conditions for deposit accounts of the receiving Office so permit)* Authorization to charge any deficiency or credit any overpayment in the total fees indicated above.
- [X] Authorization to charge the fee for priority document.

I sa 1320-210 (10)
FR
SJA/SBS/197
RWK

PATENT COOPERATION TREATY

From the INTERNATIONAL SEARCHING AUTHORITY

To:
STEVEN J. ROCCI
WOODCOCK WASHBURN LLP
ONE LIBERTY PLACE, 46TH FLOOR
PHILADELPHIA, PA 19103

RECEIVED
DEC 14 2004
Mary Ann Devine
Woodcock Washburn

RECEIVED
DEC 10 2004
Woodcock Washburn

PCT

NOTIFICATION OF TRANSMITTAL OF THE INTERNATIONAL SEARCH REPORT AND THE WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY, OR THE DECLARATION

(PCT Rule 44.1)

6/8/05

Applicant's or agent's file reference MSFT-4419	Date of mailing (day/month/year) FOR FURTHER ACTION See paragraphs 1 and 4 below
International application No. PCT/US04/24569	International filing date (day/month/year) 29 July 2004 (29.07.2004)
Applicant MICROSOFT CORPORATION	

1. ☒ The applicant is hereby notified that the international search report and the written opinion of the International Searching Authority have been established and are transmitted herewith.

   Filing of amendments and statement under Article 19:
   The applicant is entitled, if he so wishes, to amend the claims of the international application (see Rule 46):

   When? The time limit for filing such amendments is normally two months from the date of transmittal of the international search report.

   Where? Directly to the International Bureau of WIPO, 34 chemin des Colombettes 1211 Geneva 20, Switzerland, Facsimile No.: +41 22 740 14 35

   For more detailed instructions, see the notes on the accompanying sheet.

2. ☐ The applicant is hereby notified that no international search report will be established and that the declaration under Article 17(2)(a) to that effect and the written opinion of the International Searching Authority are transmitted herewith.

3. ☐ With regard to the protest against payment of (an) additional fee(s) under Rule 40.2, the applicant is notified that:

   ☐ the protest together with the decision thereon has been transmitted to the International Bureau together with the applicant's request to forward the texts of both the protest and the decision thereon to the designated Offices.

   ☐ no decision has been made yet on the protest; the applicant will be notified as soon as a decision is made.

4. **Reminders**

   Shortly after the expiration of 18 months from the priority date, the international application will be published by the International Bureau. If the applicant wishes to avoid or postpone publication, a notice of withdrawal of the international application, or of the priority claim, must reach the International Bureau as provided in Rules 90*bis*.1 and 90*bis*.3, respectively, before the completion of the technical preparations for international publication.

   The applicant may submit comments on an informal basis on the written opinion of the International Searching Authority to the International Bureau. The International Bureau will send a copy of such comments to all designated Offices unless an international preliminary examination report has been or is to be established. These comments would also be made available to the public but not before the expiration of 30 months from the priority date.

   Within 19 months from the priority date, but only in respect of some designated Offices, a demand for international preliminary examination must be filed if the applicant wishes to postpone the entry into the national phase until 30 months from the priority date (in some Offices even later); otherwise, the applicant must, within 20 months from the priority date, perform the prescribed acts for entry into the national phase before those designated Offices.

   In respect of other designated Offices, the time limit of 30 months (or later) will apply even if no demand is filed within 19 months.

   See the Annex to Form PCT/IB/301 and, for details about the applicable time limits, Office by Office, see the *PCT Applicant's Guide*, Volume II, National Chapters and the WIPO Internet site.

Name and mailing address of the ISA/ US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703) 305-3230	Authorized officer Jean M. Corrigan Telephone No. (703) 305-9000

Form PCT/ISA/220 (January 2004) *(See notes on accompanying sheet)*

RECEIVED
DEC 10 2004
DOCKET DEPT.
WWKMN

NO

198

PATENT COOPERATION TREATY

From the INTERNATIONAL SEARCHING AUTHORITY

To:
STEVEN J. ROCCI
WOODCOCK WASHBURN LLP
ONE LIBERTY PLACE, 46TH FLOOR
PHILADELPHIA, PA 19103

PCT

NOTIFICATION OF TRANSMITTAL OF THE INTERNATIONAL SEARCH REPORT AND THE WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY, OR THE DECLARATION

(PCT Rule 44.1)

	Date of mailing (*day/month/year*) 07 DEC 2004
Applicant's or agent's file reference MSFT-4419	FOR FURTHER ACTION See paragraphs 1 and 4 below
International application No. PCT/US04/24969	International filing date (*day/month/year*) 29 July 2004 (29.07.2004)
Applicant MICROSOFT CORPORATION	

1. ☒ The applicant is hereby notified that the international search report and the written opinion of the International Searching Authority have been established and are transmitted herewith.

   Filing of amendments and statement under Article 19:
   The applicant is entitled, if he so wishes, to amend the claims of the international application (see Rule 46):

   When? The time limit for filing such amendments is normally two months from the date of transmittal of the international search report.

   Where? Directly to the International Bureau of WIPO, 34 chemin des Colombettes 1211 Geneva 20, Switzerland. Facsimile No.: +41 22 740 14 35

   For more detailed instructions, see the notes on the accompanying sheet.

2. ☐ The applicant is hereby notified that no international search report will be established and that the declaration under Article 17(2)(a) to that effect and the written opinion of the International Searching Authority are transmitted herewith.

3. ☐ With regard to the protest against payment of (an) additional fee(s) under Rule 40.2, the applicant is notified that:

   ☐ the protest together with the decision thereon has been transmitted to the International Bureau together with the applicant's request to forward the texts of both the protest and the decision thereon to the designated Offices.

   ☐ no decision has been made yet on the protest; the applicant will be notified as soon as a decision is made.

4. **Reminders**

   Shortly after the expiration of 18 months from the priority date, the international application will be published by the International Bureau. If the applicant wishes to avoid or postpone publication, a notice of withdrawal of the international application, or of the priority claim, must reach the International Bureau as provided in Rules 90*bis*.1 and 90*bis*.3, respectively, before the completion of the technical preparations for international publication.

   The applicant may submit comments on an informal basis on the written opinion of the International Searching Authority to the International Bureau. The International Bureau will send a copy of such comments to all designated Offices unless an international preliminary examination report has been or is to be established. These comments would also be made available to the public but not before the expiration of 30 months from the priority date.

   Within 19 months from the priority date, but only in respect of some designated Offices, a demand for international preliminary examination must be filed if the applicant wishes to postpone the entry into the national phase until 30 months from the priority date (in some Offices even later); otherwise, the applicant must, within 20 months from the priority date, perform the prescribed acts for entry into the national phase before those designated Offices.

   In respect of other designated Offices, the time limit of 30 months (or later) will apply even if no demand is filed within 19 months.

   See the Annex to Form PCT/IB/301 and, for details about the applicable time limits, Office by Office, see the *PCT Applicant's Guide*, Volume II, National Chapters and the WIPO Internet site.

Name and mailing address of the ISA/ US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703) 305-3230	Authorized officer for Jean M. Corrigan Telephone No. (703) 305-9000

Form PCT/ISA/220 (January 2004) (*See notes on accompanying sheet*)

199

PATENT COOPERATION TREATY

PCT

INTERNATIONAL SEARCH REPORT

(PCT Article 18 and Rules 43 and 44)

Applicant's or agent's file reference MSFT-4419	FOR FURTHER ACTION	see Form PCT/ISA/220 as well as, where applicable, item 5 below.
International application No. PCT/US04/24569	International filing date (*day/month/year*) 29 July 2004 (29.07.2004)	(Earliest) Priority Date (*day/month/year*) 21 August 2003 (21.08.2003)
Applicant MICROSOFT CORPORATION		

This international search report has been prepared by this International Searching Authority and is transmitted to the applicant according to Article 18. A copy is being transmitted to the International Bureau.

This international search report consists of a total of 3 sheets.

☒ It is also accompanied by a copy of each prior art document cited in this report.

1. Basis of the Report
   a. With regard to the language, the international search was carried out on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.
      ☐ The international search was carried out on the basis of a translation of the international application furnished to this Authority (Rule 23.1(b)).
   b. ☐ With regard to any nucleotide and/or amino acid sequence disclosed in the international application, see Box No. I.
2. ☐ Certain claims were found unsearchable (See Box No. II)
3. ☐ Unity of invention is lacking (See Box No. III)
4. With regard to the title,
   ☒ the text is approved as submitted by the applicant.
   ☐ the text has been established by this Authority to read as follows:

5. With regard to the abstract,
   ☐ the text is approved as submitted by the applicant.
   ☒ the text has been established, according to Rule 38.2(b), by this Authority as it appears in Box No. IV. The applicant may, within one month from the date of mailing of this international search report, submit comments to this Authority.

6. With regard to the drawings,
   a. the figure of the drawings to be published with the abstract is Figure No. 36
      ☒ as suggested by the applicant.
      ☐ as selected by this Authority, because the applicant failed to suggest a figure.
      ☐ as selected by this Authority, because this figure better characterizes the invention.
   b. ☐ none of the figures is to be published with the abstract.

201

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US04/24569

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 17/30
US CL : 707/101, 100, 104, 103R
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/101, 100, 104, 103R

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
NPL, IEEE, ACM, WEST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 6,763,350 B2 (AGRAWAL et al.,) 13 July 2004, see the entire document.	1-42
Y, P	U 6,697,818 B2 (LI et al.,) 24 February 2004, see the entire document.	1-42

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 October 2004 (28.10.2004)	07 DEC 2004
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (703) 305-3230	Authorized officer Kim Vu Telephone No. (703) 305-9000

Form PCT/ISA/210 (second sheet) (January 2004)

202

PATENT COOPERATION TREATY

From the
INTERNATIONAL SEARCHING AUTHORITY

To:
STEVEN J. ROCCI
WOODCOCK WASHBURN LLP
ONE LIBERTY PLACE, 46TH FLOOR
PHILADELPHIA, PA 19103

PCT

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

(PCT Rule 43*bis*.1)

Date of mailing (*day/month/year*)	07 DEC 2004
Applicant's or agent's file reference MSFT-4419	FOR FURTHER ACTION See paragraph 2 below

International application No.	International filing date (*day/month/year*)	Priority date (*day/month/year*)
PCT/US04/24569	29 July 2004 (29.07.2004)	21 August 2003 (21.08.2003)

International Patent Classification (IPC) or both national classification and IPC
IPC(7): G06F 17/30 and US Cl.: 707/101, 100, 104, 103R

Applicant
MICROSOFT CORPORATION

1. This opinion contains indications relating to the following items:

- ☒ Box No. I — Basis of the opinion
- ☒ Box No. II — Priority
- ☐ Box No. III — Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
- ☐ Box No. IV — Lack of unity of invention
- ☒ Box No. V — Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
- ☐ Box No. VI — Certain documents cited
- ☐ Box No. VII — Certain defects in the international application
- ☐ Box No. VIII — Certain observations on the international application

2. FURTHER ACTION

If a demand for international preliminary examination is made, this opinion will be considered to be a written opinion of the International Preliminary Examining Authority ("IPEA") except that this does not apply where the applicant chooses an Authority other than this one to be the IPEA and the chosen IPEA has notified the International Bureau under Rule 66.1*bis*(b) that written opinions of this International Searching Authority will not be so considered.

If this opinion is, as provided above, considered to be a written opinion of the IPEA, the applicant is invited to submit to the IPEA a written reply together, where appropriate, with amendments, before the expiration of 3 months from the date of mailing of Form PCT/ISA/220 or before the expiration of 22 months from the priority date, whichever expires later.

For further options, see Form PCT/ISA/220.

3. For further details, see notes to Form PCT/ISA/220.

Name and mailing address of the ISA/ US	Authorized officer
Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450	Kim Vu Telephone No. (703) 305-9000
Facsimile No. (703) 305-3230	

Form PCT/ISA/237 (cover sheet) (January 2004)

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/24569

Box No. I Basis of this opinion

1. With regard to the language, this opinion has been established on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

☐ This opinion has been established on the basis of a translation from the original language into the following language _____, which is the language of a translation furnished for the purposes of international search (under Rules 12.3 and 23.1(b)).

2. With regard to any nucleotide and/or amino acid sequence disclosed in the international application and necessary to the claimed invention, this opinion has been established on the basis of:

a. type of material

☐ a sequence listing

☐ table(s) related to the sequence listing

b. format of material

☐ in written format

☐ in computer readable form

c. time of filing/furnishing

☐ contained in international application as filed.

☐ filed together with the international application in computer readable form.

☐ furnished subsequently to this Authority for the purposes of search.

3. ☐ In addition, in the case that more than one version or copy of a sequence listing and/or table relating thereto has been filed or furnished, the required statements that the information in the subsequent or additional copies is identical to that in the application as filed or does not go beyond the application as filed, as appropriate, were furnished.

4. Additional comments:

4

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/24569

Box No. II Priority

1. ☒ The following document has not yet been furnished:

   ☐ copy of the earlier application whose priority has been claimed (Rules 43*bis*.1 and 66.7(a)).

   ☒ translation of the earlier application whose priority has been claimed (Rules 43*bis*.1 and 66.7(b)).

   Consequently it has not been possible to consider the validity of the priority claim. This opinion has nevertheless been established on the assumption that the relevant date is the claimed priority date.

2. ☐ This opinion has been established as if no priority has been claimed due to the fact that the priority claim has been found invalid (Rules 43*bis*.1 and 64.1). Thus for the purposes of this opinion, the international filing date indicated above is considered to be the relevant date.

3. Additional observations, if necessary:

   In the event a copy of the non-US filed priority document is not furnished to this authority because the applicant fails to comply with PCT Rule 17.1 or the document is not available to this Authority from a digital library, the applicant is invited to provide this Authority with a copy of said document. The applicant is also invited to provide a translation of said document into English (if the document is not in English) within two months from the date of mailing of this written opinion. If this Authority is not provided with a copy of said non-US filed priority document and the applicant fails to comply with PCT Rule 17.1 and the priority document is not available to this Authority from a digital library and/or an English translation of said document, if it is not in English, is not provided to this Authority within the time period set forth above or by the time this Authority begins to draw up any written opinion (WO) of the International Preliminary Examining Authority (IPEA) or the International Preliminary Report on Patentability (IPRP) of the IPEA, any said WO or IPRP of the IPEA may be established as if the priority had not been claimed.

WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY

International application No.
PCT/US04/24969

Box No. V Reasoned statement under Rule 43 *bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Claims	1-42	YES
	Claims	NONE	NO
Inventive step (IS)	Claims	NONE	YES
	Claims	1-42	NO
Industrial applicability (IA)	Claims	1-42	YES
	Claims	NONE	NO

2. Citations and explanations:

Claims 1-42 lack of an inventive step under PCT Article 33(3) as being obvious over Agrawal et al., US Patent no. 6,763,350 in view of Li et al., US Patent no. 6,697,818.
As to claims 1-42:
Claims 1-42 are directed for a system for modeling real world application objects by extending items using extension, which provides additional data structures to the existing items, type structures, which were neither described nor enabled in the priority application. Therefore, claims 1-42 are being examined in light of the international file date. Agrawal, however, discloses substantially the invention as claimed, including the use of extending an item and an extension to extend the item (col.1, lines 15-col.11, line 48). On the other hand, Li discloses an analogous system that constitutes a discrete storable unit of information that can be manipulated by said hardware/software interface system (col.7, line 30-col.15, line 22). It would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the teachings of the cited references where the query module provided therein (see Agrawal's fig.1, item 20) to incorporate the use of constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system in order to provide a better performance query than is otherwise possible with conventional extenders due to a better understanding on the part of the universal extension module of a data model of underlying data types.

NOTES TO FORM PCT/ISA/220

These Notes are intended to give the basic instructions concerning the filing of amendments under Article 19. The Notes are based on the requirements of the Patent Cooperation Treaty, the Regulations and the Administrative Instructions under that Treaty. In case of discrepancy between these Notes and those requirements, the latter are applicable. For more detailed information, see also the *PCT Applicant's Guide*, a publication of WIPO.

In these Notes, "Article," "Rule" and "Section" refer to the provisions of the PCT, the PCT Regulations and the PCT Administrative Instructions, respectively.

INSTRUCTIONS CONCERNING AMENDMENTS UNDER ARTICLE 19

The applicant has, after having received the international search report, one opportunity to amend the claims of the international application. It should however be emphasized that, since all parts of the international application (claims, description and drawings) may be amended during the international preliminary examination procedure, there is usually no need to file amendments of the claims under Article 19 except where, e.g. the applicant wants the latter to be published for the purposes of provisional protection or has another reason for amending the claims before international publication. Furthermore, it should be emphasized that provisional protection is available in some States only.

What parts of the international application may be amended?

Under Article 19, only the claims may be amended.

During the international phase, the claims may also be amended (or further amended) under Article 34 before the International Preliminary Examining Authority. The description and drawings may only be amended under Article 34 before the International Preliminary Examining Authority.

Upon entry into the national phase, all parts of the international application may be amended under Article 28 or, where applicable, Article 41.

When? Within 2 months from the date of transmittal of the international search report or 16 months from the priority date, whichever time limit expires later. It should be noted, however, that the amendments will be considered as having been received on time if they are received by the International Bureau after the expiration of the applicable time limit but before the completion of the technical preparations for international publication (Rule 46.1).

Where not to file the amendments?

The amendments may only be filed with the International Bureau and not with the receiving Office or the International Searching Authority (Rule 46.2).

Where a demand for international preliminary examination has been/is filed, see below.

How? Either by cancelling one or more entire claims, by adding one or more new claims or by amending the text of one or more of the claims as filed.

A replacement sheet must be submitted for each sheet of the claims which, on account of an amendment or amendments, differs from the sheet originally filed.

All the claims appearing on a replacement sheet must be numbered in Arabic numerals. Where a claim is cancelled, no renumbering of the other claims is required. In all cases where claims are renumbered, they must be renumbered consecutively (Administrative Instructions, Section 205(b)).

The amendments must be made in the language in which the international application is to be published.

What documents must/may accompany the amendments?

Letter (Section 205(b)):

The amendments must be submitted with a letter.

The letter will not be published with the international application and the amended claims. It should not be confused with the "Statement under Article 19(1)" (see below, under "Statement under Article 19(1)").

The letter must be in English or French, at the choice of the applicant. However, if the language of the international application is English, the letter must be in English; if the language of the international application is French, the letter must be in French.

RO-105

SJR/SBS/RUK 207 (sc)

COPY SENT TO CLIENT 9-7-04 VM

PATENT COOPERATION TREATY

From the RECEIVING OFFICE

PCT

RECEIVED
AUG 3 0 2004
DOCKET DEPT. WWKMN

NOTIFICATION OF THE INTERNATIONAL APPLICATION NUMBER AND OF THE INTERNATIONAL FILING DATE

(PCT Rule 20.5(c))

To:
STEVEN J. ROCCI
WOODCOCK WASHBURN LLP
ONE LIBERTY PLACE, 46TH FLOOR
PHILADELPHIA, PENNSYLVANIA 19103

SEP - 8 2004

Mary Ann Devine
Woodcock Washburn

Date of mailing (day/month/year): 23 Aug 2004

Applicant's or agent's file reference: MSFT-4418

IMPORTANT NOTIFICATION

International application No.	International filing date (day/month/year)	Priority date (day/month/year)
PCT/US2004/024569	29 Jul 2004	21 Aug 2003

Applicant: MICROSOFT CORPORATION

Title of the invention: SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM

1. The applicant is hereby notified that the international application has been accorded the international application number and the international filing date indicated above.

2. The applicant is further notified that the record copy of the international application:

☒ was transmitted to the International Bureau on 23 Aug 2004

☐ has not yet been transmitted to the International Bureau for the reason indicated below and a copy of this notification has been sent to the International Bureau*:

RECEIVED
AUG 3 0 2004
Woodcock Washburn

☐ because the necessary national security clearance has not yet been obtained.

☐ because *(reasons to be specified)*:

* The International Bureau monitors the transmittal of the record copy by the receiving Office and will notify the applicant (with Form PCT/IB/301) of its receipt. Should the record copy not have been received by the expiration of 14 months from the priority date, the International Bureau will notify the applicant (Rule 22.1(c)).

3. FOREIGN TRANSMITTAL LICENSE INFORMATION — Completed by: TS

☐ Additional license for foreign transmittal not required. This subject matter is covered by a license already granted on the equivalent U.S. national application. Refer to that license for information concerning its scope.

☐ License for foreign transmittal not required. 37 CFR 5.11(e)(1) or 37 CFR 5.11(e)(2). However, a license may be required for additional subject matter. See 37 CFR 5.15(b).

☒ Foreign transmittal license granted, 35 U.S.C. 184; 37 CFR 5.11 on 18 Aug 2004 (date):

☒ 37 CFR 5.15(a) ☐ 37 CFR 5.15(b)

Name and mailing address of the receiving Office	Authorized officer
Mail Stop PCT, Commissioner for Patents P.O. Box 1450, Alexandria, VA 22313-1450	Tyetta Staton
Facsimile No. 703-306-3230	Telephone No. 703-305-3871

Form PCT/RO/105 (July 1992)

200

Ib-304

268

PCT/US04/024589

SJR/5851/RWK cb

PATENT COOPERATION TREATY

RECEIVED

NOV 0 2 2004

From the INTERNATIONAL BUREAU

NOTIFICATION CONCERNING SUBMISSION OR TRANSMITTAL OF PRIORITY DOCUMENT

(PCT Administrative Instructions Section 411)

306643.2

To

ROCCI, Steven J
Woodcock Washburn LLP
One Liberty Place
46th Floor
Philadelphia PA 19103
ETATS UNIS D'AMERIQUE

RECEIVED

NOV 0 2 2004

Woodcock Washburn

Date of mailing (day/month/year) 15 October 2004 (15.10.2004)	
Applicant's or agent's file reference MSFT 4418	IMPORTANT NOTIFICATION
International application No PCT/US04/024589	International filing date (day/month/year) 29 July 2004 (29.07.2004)
International publication date (day/month/year)	Priority date (day/month/year) 21 August 2003 (21.08.2003)
Applicant MICROSOFT CORPORATION et al	

1 By virtue of the Form which applicant may previously submit notifying concerning submission or transmittal of priority documents, the applicant is hereby notified of the date of receipt by the International Bureau of the priority document(s) relating to all earlier application(s) whose priority is claimed. Unless otherwise indicated by the letter "NR" in the right hand column or by an asterisk appearing next to a date of receipt, the priority document concerned was submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b).

2 (If applicable) The letters "NR" appearing in the right hand column denote a priority document which, on the date of mailing of this Form, had not yet been received by the International Bureau under Rule 17.1(a) or (b). Where under Rule 17.1(a) the priority document must be submitted by the applicant to the receiving Office or the International Bureau but the applicant fails to submit the priority document within the applicable time limit under this Rule, the attention of the applicant is directed to Rule 17.1(c) which provides that any designated Office may disregard the priority claim concerned before giving the applicant an opportunity, upon entry into the national phase, to furnish the priority document within a time limit which is reasonable under the circumstances.

3 (If applicable) An asterisk (*) appearing next to a date of receipt in the right hand column denotes a priority document submitted or transmitted to the International Bureau but not in compliance with Rule 17.1(a) or (b) (the priority document was received after the time limit prescribed in Rule 17.1(a) or the request to prepare and transmit the priority document was submitted to the receiving Office after the applicable time limit under Rule 17.1(b)). Even though the priority document was not furnished in compliance with Rule 17.1(a) or (b), the International Bureau will nevertheless transmit a copy of the document to the designated Offices for their consideration. In case such a copy is not accepted by the designated Office as the priority document, Rule 17.1(c) provides that no designated Office may disregard the priority claim concerned before giving the applicant an opportunity, upon entry into the national phase, to furnish the priority document within a time limit which is reasonable under the circumstances.

Priority date	Priority application No.	Country or regional Office or PCT receiving Office	Date of receipt of priority document
24 October 2003 (24.10.2003)	10/692,874	US	03 September 2004 (03.09.2004)

Microsoft Patent Dept

DEC 2 8 2004

D PD (ND) By

RECEIVED

NOV - 4 2004

Mary Ann Devine
Woodcock Washburn

The International Bureau of WIPO 34, chemin des Colombettes 1211 Geneva 20, Switzerland	Authorized officer Brasser Jerome
Facsimile No. +41 22 740 14 35	Facsimile No. +41 22 338 89 70 Telephone No. +41 22 338 8394

Form PCT/IB/304 (January 2004)

209

PCT INTERNATIONAL APPLICATION TRANSMITTAL LETTER	Date July 29, 2004
Regarding The International Application Of MICROSOFT CORPORATION	Docket or Reference Number MSFT-4419
Entitled SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM	

Affix Express Label Here

"Express Mail" mailing number

29 July 2004
Date of Deposit

To the United States Receiving Office (RO/US):
Accompanying this transmittal letter is the above-identified International application, including a completed Request form (PCT/RO/101). Please process the application according to the provisions of the Patent Cooperation Treaty:

The following requests are made of the RO/US:

1. ☒ PREPARATION AND TRANSMITTAL OF CERTIFIED COPY OF PRIORITY DOCUMENTS – Please prepare and transmit to the International Bureau a certified copy of the United States origin priority documents identified in Box VI of the Request form (37 CFR 1.451).
To cover the cost of copy preparation and certification (37 CFR 1.19(a)(3) and (b)(1)),

   ☒ a (check) (money order) in the amount of $see fee sheet is attached to this transmittal letter.
   ☐ the RO/US is hereby authorized to charge the following deposit account no.:

2. ☒ CHOICE OF INTERNATIONAL SEARCHING AUTHORITY – It is requested that the International Search be performed by the following International Searching Authority:

   ☒ United States Patent and Trademark Office (ISA/US)
   ☐ European Patent Office (ISA/EP)
   The appropriate Search fee for the above-named Authority is indicated on the Fee Calculation Sheet (PCT/RO/101 Annex).

3. ☒ SUPPLEMENTAL SEARCH FEES (ONLY WHEN ISA/US CONDUCTS THE INTERNATIONAL SEARCH.) – Please charge any Supplemental Search fees that may be required by the United States International Searching Authority (ISA/US) to deposit account no.: 23-3050.

   I understand that this authorization is subject to my oral confirmation thereof in each instance and that it in no way limits my right to submit a protest against payment of the Supplemental Search fees, but is merely an administrative aid to assure that the ISA/US may timely complete the Search Report.

NOTE: SUPPLEMENTAL SEARCH FEES FOR ISA/EP ARE PAYABLE DIRECTLY TO THE EUROPEAN PATENT OFFICE

4. ☒ DISCLOSURE INFORMATION – In order to assist in screening the accompanying International application for purposes of determining whether a license for foreign transmittal should and could be granted and for other purposes, the following information is supplied:

   A. ☐ There is no prior filed application relating to this invention.
   B. ☒ There is a prior application, application number US 10/646,580; PCT/US03/26144; and US 10/693,574 filed on 21 August 2003; 21 August 2003 and 24 October 2003 which contains subject matter that is

      1. ☐ substantially identical to that of the accompanying International application.
      2. ☐ less than that of the accompanying International application. The additional subject matter of the International application appears on page(s) and line(s) .
      3. ☐ more than that of the accompanying International application.

   C. ☐ Disclosure information cannot be covered by the language of Points 4A or 4B above due to the involvement of several prior applications or for other reasons. A separate sheet on which the disclosure information is explained is attached to this transmittal letter.

5. ☒ REQUEST FOR FOREIGN TRANSMITTAL LICENSE – According to the provisions of 35 U.S.C. 184 and 37 CFR 5.11, a license to transmit the accompanying International application to foreign agencies or international authorities is hereby requested.

SIGNER IS THE	NAME OF SIGNER (typed) Steven B. Samuels
☐ APPLICANT ☐ COMMON REPRESENTATIVE ☒ (ATTORNEY) (AGENT) REG. NO. 37,711	SIGNATURE

210

Notificacion Recibo de Copia Record (3) IB-301

PCT/US2004/024569

PATENT COOPERATION TREATY

210

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF RECEIPT OF RECORD COPY

(PCT Rule 24.2(a))

RECEIVED
OCT 2004
RECEIPT

To:

ROCCI, Steven, J.
Woodcock Washburn LLP
One Liberty Place
46th Floor
Philadelphia, PA 19103
United States of America

Date of mailing (day/month/year)	IMPORTANT NOTIFICATION
08 October 2004 (08.10.2004)	
Applicant's or agent's file reference	**International application No.**
MSFT-4419	PCT/US2004/024569

The applicant is hereby notified that the International Bureau has received the record copy of the international application as detailed below.

Name(s) of the applicant(s) and State(s) for which they are applicants:

MICROSOFT CORPORATION (for all designated States except US)
DEMIROSKI, Bekim et al (for US)

International filing date	:	29 July 2004 (29.07.2004)
Priority date(s) claimed	:	21 August 2003 (21.08.2003) 21 August 2003 (21.08.2003) 24 October 2003 (24.10.2003)
Date of receipt of the record copy by the International Bureau	:	27 August 2004 (27.08.2004)
List of designated Offices	:	

AP :BW,GH,GM,KE,LS,MW,MZ,NA,SD,SL,SZ,TZ,UG,ZM,ZW
EA :AM,AZ,BY,KG,KZ,MD,RU,TJ,TM
EP :AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HU,IE,IT,LU,MC,NL,PL,PT,RO,SE,SI,SK,TR
OA :BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG
National :AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BW,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NA,NI,NO,NZ,OM,PG,PH,PL,PT,RO,RU,SC,SD,SE,SG,SK,SL,SY,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,YU,ZA,ZM,ZW

The International Bureau of WIPO
34, chemin des Colombettes
1211 Geneva 20, Switzerland

Facsimile No. (41-22) 338.89.75

Authorized officer:
Alison OSBORNE (Fax : 338 89 75)

Telephone No. (41-22) 338 8472

Continuation of Form PCT/IB/301

211

NOTIFICATION OF RECEIPT OF RECORD COPY

Date of mailing (day/month/year)	IMPORTANT NOTIFICATION
08 October 2004 (08.10.2004)	
Applicant's or agent's file reference	**International application No.**
MSFT-4419	PCT/US2004/024569

ATTENTION

The applicant should carefully check the data appearing in this Notification. In case of any discrepancy between these data and the indications in the international application, the applicant should immediately inform the International Bureau.

In addition, the applicant's attention is drawn to the information contained in the Annex, relating to:

[X] time limits for entry into the national phase - see updated important information (as of April 2002)

[X] requirements regarding priority documents (if applicable)

A copy of this Notification is being sent to the receiving Office and to the International Searching Authority.

ANNEX TO FORM PCT/IB/301

International application No.
PCT/US2004/024569

INFORMATION ON TIME LIMITS FOR ENTERING THE NATIONAL PHASE

The applicant is reminded that the "national phase" must be entered before each of the designated Offices indicated on the cover sheet of this Notification by paying national fees and furnishing translations, as prescribed by Articles 22 and 39 and the applicable national laws. In addition, the applicant may also have to comply with other special requirements applicable in certain Offices. It is the applicant's responsibility to ensure the necessary steps to enter the national phase are taken in a timely fashion. Most Offices do not issue reminders to applicants in connection with the entry into the national phase.

The applicable time limit for entering the national phase will, subject to what is said in the following paragraph, be 30 MONTHS from the priority date, not only in respect of any elected Office if a demand for international preliminary examination is filed before the expiration of 19 months from the priority date (see Article 39(1)), but also in respect of any designated Office, in the absence of filing of such demand, where Article22(1) as modified with effect from 1 April 2002 applies in respect of that designated Office. For further details, see PCT Gazette No. 44/2001 of 1 November 2001, pages 19928, 19932 and 19934, as well as the PCT Newsletter, October and November 2001 and February 2002 issues.

In practice, time limits other than the 30-month time limit will continue to apply, for various periods of time, in respect of certain designated or elected Offices. For regular updates on the applicable time limits (20, 21, 30 or 31 months, or other time limit), Office by Office, refer to the PCT Gazette("Section IV" part published on a weekly basis), to the PCT Newsletter (on a monthly basis) and to the relevant National Chapters in Volume II of the PCT Applicant's Guide (the paper version of which is updated usually twice a year and the Internet version of which is updated usually on a weekly basis). Finally, a cumulative table of all applicable time limits for entering the national phase is available from WIPO's Internet site, via links from various pages the site including those of the Gazette, Newsletter and Guide, at http://www.wipo.int/pct/en/index.html.

Information about the requirements for filing a demand for international preliminary examination is set out in the PCT Applicant's Guide, Volume I/A, Chapter IX. Note that only an applicant who is a national or resident of a PCT Contracting State which is bound by Chapter II has the right to file a demand for international preliminary examination (at present, all PCT Contracting States are bound by Chapter II).

REQUIREMENTS REGARDING PRIORITY DOCUMENTS

For applicants who have not yet complied with the requirements regarding priority documents, the following is recalled.

Where the priority of an earlier national, regional or international application is claimed, the applicant must submit a copy of the said earlier application, certified by the authority with which it was filed ("the priority document") to the receiving Office (which will transmit it to the International Bureau) or directly to the International Bureau, before the expiration of 16 months from the priority date, provided that any such priority document may still be submitted to the International Bureau before that date of international publication of the international application, in which case that document will be considered to have been received by the International Bureau on the last day of the 16-month time limit (Rule 17.1(a)).

Where the priority document is issued by the receiving Office, the applicant may, instead of submitting the priority document, request the receiving Office to prepare and transmit the priority document to the International Bureau. Such request must be made before the expiration of the 16-month time limit and may be subjected by the receiving Office to the payment of a fee (Rule 17.1(b)).

If the priority document concerned is not submitted to the International Bureau or if the request to the receiving Office to prepare and transmit the priority document has not been made (and the corresponding fee, if any, paid) within the applicable time limit indicated under the preceding paragraphs, any designated State may disregard the priority claim, provided that no designated Office may disregard the priority claim concerned before giving the applicant an opportunity, upon entry into the national phase, to furnish the priority document within the time limit which is reasonable under the circumstances (Rule 17.1(a)).

Where several priorities are claimed, the priority date to be considered for the purposes of computing the 16-month time limit (and all other PCT time limits) is the filing date of the earliest application whose priority is claimed (Article 2(xi)(b)).

213

SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Patent Application No. 10/693,574 (Atty Docket No. MSFT-2847) filed on October 24, 2003, entitled " SYSTEMS AND METHODS FOR EXTENSIONS AND INHERITANCE FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; U.S. Patent Application No. 10/646,580 (Atty Docket No. MSFT-2735), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR DATA MODELING IN AN ITEM-BASED STORAGE PLATFORM"; and International Application No. PCT/US 03/26144 (Atty Docket No. MSFT-2779) filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR DATA MODELING IN AN ITEM-BASED STORAGE PLATFORM"; the contents of which are herein incorporated by reference.

[0002] This application is related by subject matter to the inventions disclosed in the following commonly assigned applications, the contents of which are hereby incorporated by reference into this present application in their entirety (and partially summarized herein for convenience): U.S. Patent Application No. 10/647,058 (Atty. Docket No. MSFT-1748), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR REPRESENTING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM BUT INDEPENDENT OF PHYSICAL REPRESENTATION"; U.S. Patent Application No. 10/646,941 (Atty. Docket No. MSFT-1749), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR SEPARATING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM FROM THEIR PHYSICAL ORGANIZATION"; U.S. Patent Application No. 10/646,940 (Atty. Docket No. MSFT-1750), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A BASE SCHEMA FOR ORGANIZING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE

2 of 5

SYSTEM"; U.S. Patent Application No. 10/646,632 (Atty. Docket No. MSFT-1751), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A CORE SCHEMA FOR PROVIDING A TOP-LEVEL STRUCTURE FOR ORGANIZING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; U.S. Patent Application No. 10/646,645 (Atty. Docket No. MSFT-1752), filed on August 21, 2003, entitled "SYSTEMS AND METHOD FOR REPRESENTING RELATIONSHIPS BETWEEN UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; U.S. Patent Application No. 10/646,575 (Atty. Docket No. MSFT-2733), filed on August 21, 2003, entitled "SYSTEMS AND METHODS FOR INTERFACING APPLICATION PROGRAMS WITH AN ITEM-BASED STORAGE PLATFORM"; U.S. Patent Application No. 10/646,646 (Atty. Docket No. MSFT-2734), filed on August 21, 2003, entitled "STORAGE PLATFORM FOR ORGANIZING, SEARCHING, AND SHARING DATA"; U.S. Patent Application No. 10/692,779 (Atty. Docket No. MSFT-2829), filed on October 24, 2003, entitled "SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A DIGITAL IMAGES SCHEMA FOR ORGANIZING UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; U.S. Patent Application No. 10/692,515 (Atty. Docket No. MSFT-2844), filed on October 24, 2003, entitled "SYSTEMS AND METHODS FOR PROVIDING SYNCHRONIZATION SERVICES FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; U.S. Patent Application No. 10/692,508 (Atty. Docket No. MSFT-2845), filed on even date herewith, entitled "SYSTEMS AND METHODS FOR PROVIDING RELATIONAL AND HIERARCHICAL SYNCHRONIZATION SERVICES FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM"; and U.S. Patent Application No. 10/693,362 (Atty. Docket No. MSFT-2846), filed on October 24, 2003, entitled "SYSTEMS AND METHODS FOR THE IMPLEMENTATION OF A SYNCHRONIZATION SCHEMAS FOR UNITS OF INFORMATION MANAGEABLE BY A HARDWARE/SOFTWARE INTERFACE SYSTEM".

FIELD OF THE INVENTION

[0003] The present invention relates generally to the field of information storage and retrieval, and, more particularly, to an active storage platform for organizing, searching, and sharing different types of data in a computerized system. Various embodiments of the present invention are directed to the use of extension and inheritance methods utilized by a hardware/software interface system to manage data.

BACKGROUND

[0004] Individual disk capacity has been growing at roughly seventy percent (70%) per year over the last decade. Moore's law accurately predicted the tremendous gains in central processing unit (CPU) power that has occurred over the years. Wired and wireless technologies have provided tremendous connectivity and bandwidth. Presuming current trends continue, within several years the average laptop computer will possess roughly one terabyte (TB) of storage and contain millions of files, and 500 gigabyte (GB) drives will become commonplace.

[0005] Consumers use their computers primarily for communication and organizing personal information, whether it is traditional personal information manager (PIM) style data or media such as digital music or photographs. The amount of digital content, and the ability to store the raw bytes, has increased tremendously; however the methods available to consumers for organizing and unifying this data has not kept pace. Knowledge workers spend enormous amounts of time managing and sharing information, and some studies estimate that knowledge workers spend 15-25% of their time on non-productive information related activities. Other studies estimate that a typical knowledge worker spends about 2.5 hours per day searching for information.

[0006] Developers and information technology (IT) departments invest significant amounts of time and money in building their own data stores for common storage abstractions to represent such things as people, places, times, and events. Not only does this result in duplicated work, but it also creates islands of common data with no mechanisms for common searching or sharing of that data. Just consider how many address books can exist today on a computer running the Microsoft Windows operating system. Many applications, such as e-mail clients and personal finance programs, keep individual address books, and there is little sharing among applications of the address book data that each such program individually maintains. Consequently, a finance program (like Microsoft Money) does not share addresses for payees

with the addresses maintained in an email contact folder (like the one in Microsoft Outlook). Indeed, many users have multiple devices and logically should synchronize their personal data amongst themselves and across a wide variety of additional sources, including cell phones to commercial services such as MSN and AOL; nevertheless, collaboration of shared documents is largely achieved by attaching documents to e-mail messages—that is, manually and inefficiently.

[0007] One reason for this lack of collaboration is that traditional approaches to the organization of information in computer systems have centered on the use of file-folder-and-directory-based systems ("file systems") to organize pluralities of files into directory hierarchies of folders based on an abstraction of the physical organization of the storage medium used to store the files. The Multics operating system, developed during the 1960s, can be credited with pioneering the use of the files, folders, and directories to manage storable units of data at the operating system level. Specifically, Multics used symbolic addresses within a hierarchy of files (thereby introducing the idea of a file path) where physical addresses of the files were not transparent to the user (applications and end-users). This file system was entirely unconcerned with the file format of any individual file, and the relationships amongst and between files was deemed irrelevant at the operating system level (that is, other than the location of the file within the hierarchy). Since the advent of Multics, storable data has been organized into files, folders, and directories at the operating system level. These files generally include the file hierarchy itself (the "directory") embodied in a special file maintained by the file system. This directory, in turn, maintains a list of entries corresponding to all of the other files in the directory and the nodal location of such files in the hierarchy (herein referred to as the folders). Such has been the state of the art for approximately forty years.

[0008] However, while providing a reasonable representation of information residing in the computer's physical storage system, a file system is nevertheless an abstraction of that physical storage system, and therefore utilization of the files requires a level of indirection (interpretation) between what the user manipulates (units having context, features, and relationships to other units) and what the operating system provides (files, folders, and directories). Consequently, users (applications and/or end-users) have no choice but to force units of information into a file system structure even when doing so is inefficient, inconsistent, or otherwise undesirable. Moreover, existing file systems know little about the structure of data

stored in individual files and, because of this, most of the information remains locked up in files that may only be accessed (and comprehensible) to the applications that wrote them. Consequently, this lack of schematic description of information, and mechanisms for managing information, leads to the creation of silos of data with little data sharing among the individual silos. For example, many personal computer (PC) users have more than five distinct stores that contain information about the people they interact with on some level—for example, Outlook Contacts, online account addressees, Windows Address Book, Quicken Payees, and instant messaging (IM) buddy lists—because organizing files presents a significant challenge to these PC users. Because most existing file systems utilize a nested folder metaphor for organizing files and folders, as the number of files increases the effort necessary to maintain an organization scheme that is flexible and efficient becomes quite daunting. In such situations, it would be very useful to have multiple classifications of a single file; however, using hard or soft links in existing file systems is cumbersome and difficult to maintain.

[0009] Several unsuccessful attempts to address the shortcomings of file systems have been made in the past. Some of these previous attempts have involved the use of content addressable memory to provide a mechanism whereby data could be accessed by content rather than by physical address. However, these efforts have proven unsuccessful because, while content addressable memory has proven useful for small-scale use by devices such as caches and memory management units, large-scale use for devices such as physical storage media has not yet been possible for a variety of reasons, and thus such a solution simply does not exist. Other attempts using object-oriented database (OODB) systems have been made, but these attempts, while featuring strong database characteristics and good non-file representations, were not effective in handling file representations and could not replicate the speed, efficiency, and simplicity of the file and folder based hierarchical structure at the hardware/software interface system level. Other efforts, such as those that attempted to use SmallTalk (and other derivatives), proved to be quite effective at handling file and non-file representations but lacked database features necessary to efficiently organize and utilize the relationships that exist between the various data files, and thus the overall efficiency of such systems was unacceptable. Yet other attempts to use BeOS (and other such operating systems research) proved to be inadequate at handling non-file representations—the same core shortcoming of traditional file systems—

despite being able to adequately represent files while providing some necessary database features.

[0010] Database technology is another area of the art in which similar challenges exits. For example, while the relational database model has been a great commercial success, in truth independent software vendors (ISV) generally exercise a small portion of the functionality available in relational database software products (such as Microsoft SQL Server). Instead, most of an application's interaction with such a product is in the form of simple "gets" and "puts". While there are a number of readily apparent reasons for this—such as being platform or database agnostic—one key reason that often goes unnoticed is that the database does not necessarily provide the exact abstractions that a major business application vendor really needs. For example, while the real world has the notion of "items", such as "customers" or "orders" (along with an order's embedded "line items" as items in and of themselves), relational databases only talk in terms of tables and rows. Consequently, while the application may desire to have aspects of consistency, locking, security, and/or triggers at the item level (to name a few), generally databases provide these features only at the table/row level. While this may work fine if each item gets mapped to a single row in some table in the database, in the case of an order with multiple line items there may be reasons why an item actually gets mapped to multiple tables and, when that is the case, the simple relational database system does not quite provide the right abstractions. Consequently, an application must build logic on top of the database to provide these basic abstractions. In other words, the basic relational model does not provide a sufficient platform for storage of data on which higher-level applications can easily be developed because the basic relational model requires a level of indirection between the application and the storage system–where the semantic structure of the data might only be visible in the application in certain instances. While some database vendors are building higher-level functionality into their products--such as providing object relational capabilities, new organizational models, and the like–none have yet to provide the kind of comprehensive solution needed, where a truly comprehensive solution is one which provides both useful data model abstractions (such as "Items," "Extensions," "Relationships," and so on) for useful domain abstractions (such as "Persons," "Locations," "Events," etc.).

[0011] In view of the foregoing deficiencies in existing data storage and database technologies, there is a need for a new storage platform that provides an improved ability to organize, search, and share all types of data in a computer system--a storage platform that extends and broadens the data platform beyond existing file systems and database systems, and that is designed to be the store for all types of data. The present invention, together with the related inventions incorporated by reference earlier herein, satisfies this need. In particular, the present invention provides methods of extension and inheritance for objects manipulated by a hardware/software interface system.

SUMMARY

[0012] The following summary provides an overview of various aspects of the invention described in the context of the related inventions incorporated-by-reference earlier herein (the "related inventions"). This summary is not intended to provide an exhaustive description of all of the important aspects of the invention, nor to define the scope of the invention. Rather, this summary is intended to serve as an introduction to the detailed description and figures that follow.

[0013] The present invention, as well as the related inventions, are collectively directed to a storage platform for organizing, searching, and sharing data. The storage platform of the present invention extends and broadens the concept of data storage beyond existing file systems and database systems, and is designed to be the store for all types of data including structured, non-structured, or semi-structured data.

[0014] The storage platform of the present invention comprises a data store implemented on a database engine. The database engine comprises a relational database engine with object relational extensions. The data store implements a data model that supports organization, searching, sharing, synchronization, and security of data. Specific types of data are described in schemas, and the platform provides a mechanism to extend the set of schemas to define new types of data (essentially subtypes of the basic types provides by the schemas). A synchronization capability facilitates the sharing of data among users or systems. File-system-like capabilities are provided that allow interoperability of the data store with existing file systems but without the limitation of such traditional file systems. A change tracking mechanism provides the ability track changes to the data store. The storage platform further

comprises a set of application program interfaces that enable applications to access all of the foregoing capabilities of the storage platform and to access the data described in the schemas.

[0015] The data model implemented by the data store defines units of data storage in terms of items, elements, and relationships. An item is a unit of data storable in a data store and can comprise one or more elements and relationships. An element is an instance of a type comprising one or more fields (also referred to herein as a property). A relationship is a link between two items. (As used herein, these and other specific terms may be capitalized in order to offset them from other terms used in close proximity; however, there is no intention whatsoever to distinguish between a capitalized term, e.g. "Item", and the same term when not capitalized, e.g., "item", and no such distinction should be presumed or implied.)

[0016] The computer system further comprises a plurality of Items where each Item constitutes a discrete storable unit of information that can be manipulated by a hardware/software interface system; a plurality of Item Folders that constitute an organizational structure for said Items; and a hardware/software interface system for manipulating a plurality of Items and wherein each Item belongs to at least one Item Folder and may belong to more than one Item Folder.

[0017] An Item or some of the Item's property values may be computed dynamically as opposed to being derived from a persistent store. In other words, the hardware/software interface system does not require that the Item be stored, and certain operations are supported such as the ability to enumerate the current set of Items or the ability to retrieve an Item given its identifier (which is more fully described in the sections that describe the application programming interface, or API) of the storage platform – for example, an Item might be the current location of a cell phone or the temperature reading on a temperature sensor. The hardware/software interface system may manipulate a plurality of Items, and may further comprise Items interconnected by a plurality of Relationships managed by the hardware/software interface system.

[0018] A hardware/software interface system for the computer system further comprises a core schema to define a set of core Items which said hardware/software interface system understands and can directly process in a predetermined and predictable way. To

manipulate a plurality of Items, the computer system interconnects said Items with a plurality of Relationships and manages said Relationships at the hardware/software interface system level.

[0019] The API of the storage platform provides data classes for each item, item extension, and relationship defined in the set of storage platform schemas. In addition, the application programming interface provides a set of framework classes that define a common set of behaviors for the data classes and that, together with the data classes, provide the basic programming model for the storage platform API. The storage platform API provides a simplified query model that enables application programmers to form queries based on various properties of the items in the data store, in a manner that insulates the application programmer from the details of the query language of the underlying database engine. The storage platform API also collects changes to an item made by an application program and then organizes them into the correct updates required by the database engine (or any kind of storage engine) on which the data store is implemented. This enables application programmers to make changes to an item in memory, while leaving the complexity of data store updates to the API.

[0020] Through its common storage foundation and schematized data, the storage platform of the present invention enables more efficient application development for consumers, knowledge workers and enterprises. It offers a rich and extensible application programming interface that not only makes available the capabilities inherent in its data model, but also embraces and extends existing file system and database access methods.

[0021] Within view of this overarching structure of interrelated inventions (described in detail in Section II of the Detailed Description), the present invention is specifically directed to the use of Extensions to extend functionality of Item and Property types, as well as to the use of Inheritance to facilitate efficient searching and organization among related Items (described in detail in Section III of the Detailed Description). Other features and advantages of the invention may become apparent from the following detailed description of the invention and accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] The foregoing summary, as well as the following detailed description of the invention, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary embodiments of

various aspects of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0023] Fig. 1 is a block diagram representing a computer system in which aspects of the present invention may be incorporated;

[0024] Fig. 2 is a block diagram illustrating a computer system divided into three component groups: the hardware component, the hardware/software interface system component, and the application programs component;

[0025] Fig. 2A illustrates the traditional tree-based hierarchical structure for files grouped in folders in a directory in a file-based operating system;

[0026] Fig. 3 is a block diagram illustrating a storage platform;

[0027] Fig. 4 illustrates the structural relationship between Items, Item Folders, and Categories;

[0028] Fig. 5A is a block diagram illustrating the structure of an Item;

[0029] Fig. 5B is a block diagram illustrating the complex property types of the Item of Fig. 5A;

[0030] Fig. 5C is a block diagram illustrating the "Location" Item wherein its complex types are further described (explicitly listed);

[0031] Fig. 6A illustrates an Item as a subtype of the Item found in the Base Schema;

[0032] Fig. 6B is a block diagram illustrating the subtype Item of Fig. 6A wherein its inherited types are explicitly listed (in addition to its immediate properties);

[0033] Fig. 7 is a block diagram illustrating the Base Schema including its two top-level class types, Item and PropertyBase, and the additional Base Schema types derived therefrom;

[0034] Fig. 8A is a block diagram illustrating Items in the Core Schema;

[0035] Fig. 8B is a block diagram illustrating the property types in the Core Schema;

[0036] Fig. 9 is a block diagram illustrating an Item Folder, its member Items, and the interconnecting Relationships between the Item Folder and its member Items;

[0037] Fig. 10 is a block diagram illustrating a Category (which, again, is an Item itself), its member Items, and the interconnecting Relationships between the Category and its member Items;

[0038] Fig. 11 is a diagram illustrating a reference type hierarchy of the data model of the storage platform;

[0039] Fig. 12 is a diagram illustrating how relationships are classified;

[0040] Fig. 13 is a diagram illustrating a notification mechanism;

[0041] Fig. 14 is a diagram illustrating an example in which two transactions are both inserting a new record into the same B-Tree;

[0042] Fig. 15 illustrates a data change detection process;

[0043] Fig. 16 illustrates an exemplary directory tree;

[0044] Fig. 17 shows an example in which an existing folder of a directory-based file system is moved into the storage platform data store;

[0045] Fig. 18 illustrates the concept of Containment Folders;

[0046] Fig. 19 illustrates the basic architecture of the storage platform API;

[0047] Fig. 20 schematically represents the various components of the storage platform API stack;

[0048] Fig. 21A is a pictorial representation of an exemplary Contacts Item schema;

[0049] Fig. 21B is a pictorial representation of the Elements for the exemplary Contacts Item schema of Fig. 21A;

[0050] Fig. 22 illustrates the runtime framework of the storage platform API;

[0051] Fig. 23 illustrates the execution of a "FindAll" operation;

[0052] Fig. 24 illustrates the process by which storage platform API classes are generated from the storage platform Schema;

[0053] Fig. 25 illustrates a schema on which a File API is based;

[0054] Fig. 26 is a diagram illustrating an access mask format used for data security purposes;

[0055] Fig. 27 (parts a, b, and c) depicts a new identically protected security region being carved out of an existing security region;

[0056] Fig. 28 is a diagram illustrating the concept of an Item search view;

[0057] Fig. 29 is a diagram illustrating an exemplary Item hierarchy;

[0058] Fig. 30A illustrates an interface Interface1 as a conduit through which first and second code segments communicate;

[0059] Fig. 30B illustrates an interface as comprising interface objects I1 and I2 which enable first and second code segments of a system to communicate via medium M;

[0060] Fig. 31A illustrates how the function provided by interface Interface1 may be subdivided to convert the communications of the interface into multiple interfaces Interface1A, Interface 1B, Interface 1C;

[0061] Fig. 31B illustrates how the function provided by interface I1 may be subdivided into multiple interfaces I1a, I1b, I1c;

[0062] Fig. 32A illustrates a scenario where a meaningless parameter precision can be ignored or replaced with an arbitrary parameter;

[0063] Fig. 32B illustrates a scenario where an interface is replaced by a substitute interface that is defined to ignore or add parameters to an interface;

[0064] Fig. 33A illustrates a scenario where a 1st and 2nd Code Segments are merged into a module containing them both;

[0065] Fig. 33B illustrates a scenario where part or all of an interface may be written inline into another interface to form a merged interface.

[0066] Fig. 34A illustrates how one or more pieces of middleware might convert communications on the first interface to conform them to one or more different interfaces;

[0067] Fig. 34B illustrates how a code segment can be introduced with an interface to receive the communications from one interface but transmit the functionality to second and third interfaces;

[0068] Fig. 35A illustrates how a just-in-time compiler (JIT) might convert communications from one code segment to another code segment;

[0069] Fig. 35B illustrates a JIT method of dynamically rewriting one or more interfaces may be applied to dynamically factor or otherwise alter said interface;

[0070] Fig. 36 illustrates a series of interrelated Items and a subset of their Relationships;

[0071] Fig. 37A illustrates the shortcoming of standard subtyping of an Item for application specific purposes;

[0072] Fig. 37B illustrates a partial solution to the problems of standard subtyping; and

[0073] Fig. 37C illustrates one embodiment of the present invention to extend an Item with an Extension that is distinct and separate from Contact itself, and thus enables a multi-typing functionality.

[Remainder of Page Intentionally Left Blank]

DETAILED DESCRIPTION

TABLE OF CONTENTS

I. INTRODUCTION ..... - 19 -

A. EXEMPLARY COMPUTING ENVIRONMENT ..... - 19 -

B. TRADITIONAL FILE-BASED STORAGE ..... - 23 -

II. WINFS STORAGE PLATFORM FOR ORGANIZING, SEARCHING, AND SHARING DATA ..... - 25 -

C. GLOSSARY ..... - 25 -

D. STORAGE PLATFORM OVERVIEW ..... - 26 -

E. THE DATA MODEL ..... - 27 -

1. Items ..... - 29 -

2. Item Identification ..... - 33 -

3. Item Folders and Categories ..... - 33 -

4. Schemas ..... - 35 -

a) Base Schema ..... - 35 -

b) Core Schema ..... - 36 -

5. Relationships ..... - 38 -

a) Relationship Declaration ..... - 39 -

b) Holding Relationship ..... - 40 -

c) Embedding Relationships ..... - 41 -

d) Reference Relationships ..... - 42 -

e) Rules and Constraints ..... - 43 -

f) Ordering of Relationships ..... - 43 -

6. Extensibility ..... - 49 -

a) Item extensions ..... - 50 -

b) Extending NestedElement types ..... - 54 -

F. DATABASE ENGINE ..... - 56 -

1. Data Store Implementation Using UDTs ..... - 57 -

2. Item Mapping ..... - 58 -

3. Extension Mapping ..... - 59 -

4. Nested Element Mapping ..... - 59 -

5. Object Identity ..... - 59 -

6. SQL Object Naming ..... - 59 -

7. Column Naming ..... - 60 -

8. Search Views ..... - 61 -

a) Item ..... - 62 -

(1) Master Item Search View ..... - 62 -

(2) Typed Item Search Views ..... - 62 -

b) Item Extensions ..... - 63 -

(1) Master Extension Search View ..... - 63 -

(2) Typed Extension Search Views ..... - 64 -

c) Nested Elements ..... - 64 -

d) Relationships ..... - 65 -

(1) Master Relationship Search View ..... - 65 -

(2) Relationship Instance Search Views ..... - 65 -

e) - 66 -

9. Updates ..... - 66 -

10. Change Tracking & Tombstones ..... - 67 -

a) Change Tracking ..... - 67 -

(1) Change Tracking in "Master" Search Views ..... - 67 -
(2) Change Tracking in "Typed" Search Views ..... - 68 -
b) Tombstones ..... - 69 -
(1) Item Tombstones ..... - 69 -
(2) Extension Tombstones ..... - 70 -
(3) Relationships Tombstone ..... - 70 -
(4) Tombstone Cleanup ..... - 71 -
11. Helper APIs and Functions ..... - 71 -
a) Function [System.Storage].GetItem ..... - 71 -
b) Function [System.Storage].GetExtension ..... - 71 -
c) Function [System.Storage].GetRelationship ..... - 72 -
12. Metadata ..... - 72 -
a) Schema Metadata ..... - 72 -
b) Instance Metadata ..... - 72 -
G. SECURITY ..... - 72 -
H. NOTIFICATIONS AND CHANGE TRACKING ..... - 73 -
I. SYNCHRONIZATION ..... - 74 -
1. Storage Platform-to-Storage Platform Synchronization ..... - 74 -
a) Synchronization (Sync) Controlling Applications ..... - 75 -
b) Schema annotation ..... - 76 -
c) Sync Configuration ..... - 77 -
(1) Community Folder - Mappings ..... - 77 -
(2) Profiles ..... - 78 -
(3) Schedules ..... - 79 -

d) Conflict Handling ..... - 79 -

(1) Conflict Detection ..... - 80 -

(a) Knowledge-based conflicts - 80 -

(b) Constraint-based conflicts - 80 -

(2) Conflict Processing ..... - 81 -

(a) Automatic Conflict resolution - 81 -

(b) Conflict Logging - 82 -

(c) Conflict inspection and resolution - 82 -

(d) Convergence of replicas and Propagation of Conflict Resolutions - 82 -

2. Synchronizing to non-storage platform data stores ..... - 83 -

a) Sync Services ..... - 84 -

(1) Change Enumeration ..... - 84 -

(2) Change Application ..... - 85 -

(3) Conflict Resolution ..... - 85 -

b) Adapter implementation ..... - 86 -

3. Security ..... - 86 -

4. Manageability ..... - 87 -

J. TRADITIONAL FILE SYSTEM INTEROPERABILITY ..... - 87 -

K. STORAGE PLATFORM API ..... - 88 -

III. EXTENSIONS AND INHERITANCE ..... - 95 -

A. TYPE SYSTEM ..... - 95 -

B. TYPE FAMILIES ..... - 100 -

1. Nested Element Types ..... - 101 -

2. Item Types ..... - 101 -

3. Relationship Types ..... - 101 -

a) Relationship Semantics ..... - 103 -

b) Relationship Rules and Contraints ..... - 105 -

4. Extension Types ..... - 106 -

C. ENHANCED FUNCTIONALITY ..... - 107 -

1. Inheritance ..... - 107 -

2. Extensions ..... - 108 -

IV. CONCLUSION ..... - 109 -

I. INTRODUCTION

[0074] The subject matter of the present invention is described with specificity to meet statutory requirements. However, the description itself is not intended to limit the scope of this patent. Rather, the inventors have contemplated that the claimed subject matter might also be embodied in other ways, to include different steps or combinations of steps similar to the ones described in this document, in conjunction with other present or future technologies. Moreover, although the term "step" may be used herein to connote different elements of methods employed, the term should not be interpreted as implying any particular order among or between various steps herein disclosed unless and except when the order of individual steps is explicitly described.

A. EXEMPLARY COMPUTING ENVIRONMENT

[0075] Numerous embodiments of the present invention may execute on a computer. Fig. 1 and the following discussion is intended to provide a brief general description of a suitable computing environment in which the invention may be implemented. Although not required, various aspects of the invention may be described in the general context of computer executable instructions, such as program modules, being executed by a computer, such as a client workstation or a server. Generally, program modules include routines, programs, objects, components, data structures and the like that perform particular tasks or implement particular abstract data types. Moreover, the invention may be practiced with other computer system configurations, including hand held devices, multi processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0076] As shown in Fig. 1, an exemplary general purpose computing system includes a conventional personal computer 20 or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 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. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start up, is stored in ROM 24. The personal computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer readable media provide non volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs) and the like may also be used in the exemplary operating environment. Likewise, the exemplary environment may also include many types of monitoring devices such as heat sensors and security or fire alarm systems, and other sources of information.

[0077] A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more application programs 36, other program modules 37 and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite disk, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor 47,

personal computers typically include other peripheral output devices (not shown), such as speakers and printers. The exemplary system of Fig. 1 also includes a host adapter 55, Small Computer System Interface (SCSI) bus 56, and an external storage device 62 connected to the SCSI bus 56.

[0078] The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another 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 personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise wide computer networks, intranets and the Internet.

[0079] When used in a LAN networking environment, the personal computer 20 is connected to the LAN 51 through a network interface or adapter 53. When used in a WAN networking environment, the personal computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. 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.

[0080] As illustrated in the block diagram of Fig. 2, a computer system 200 can be roughly divided into three component groups: the hardware component 202, the hardware/software interface system component 204, and the applications programs component 206 (also referred to as the "user component" or "software component" in certain contexts herein).

[0081] In various embodiments of a computer system 200, and referring back to Fig. 1, the hardware component 202 may comprise the central processing unit (CPU) 21, the memory (both ROM 24 and RAM 25), the basic input/output system (BIOS) 26, and various input/output (I/O) devices such as a keyboard 40, a mouse 42, a monitor 47, and/or a printer (not shown),

among other things. The hardware component 202 comprises the basic physical infrastructure for the computer system 200.

[0082] The applications programs component 206 comprises various software programs including but not limited to compilers, database systems, word processors, business programs, videogames, and so forth. Application programs provide the means by which computer resources are utilized to solve problems, provide solutions, and process data for various users (machines, other computer systems, and/or end-users).

[0083] The hardware/software interface system component 204 comprises (and, in some embodiments, may solely consist of) an operating system that itself comprises, in most cases, a shell and a kernel. An "operating system" (OS) is a special program that acts as an intermediary between application programs and computer hardware. The hardware/software interface system component 204 may also comprise a virtual machine manager (VMM), a Common Language Runtime (CLR) or its functional equivalent, a Java Virtual Machine (JVM) or its functional equivalent, or other such software components in the place of or in addition to the operating system in a computer system. The purpose of a hardware/software interface system is to provide an environment in which a user can execute application programs. The goal of any hardware/software interface system is to make the computer system convenient to use, as well as utilize the computer hardware in an efficient manner.

[0084] The hardware/software interface system is generally loaded into a computer system at startup and thereafter manages all of the application programs in the computer system. The application programs interact with the hardware/software interface system by requesting services via an application program interface (API). Some application programs enable end-users to interact with the hardware/software interface system via a user interface such as a command language or a graphical user interface (GUI).

[0085] A hardware/software interface system traditionally performs a variety of services for applications. In a multitasking hardware/software interface system where multiple programs may be running at the same time, the hardware/software interface system determines which applications should run in what order and how much time should be allowed for each application before switching to another application for a turn. The hardware/software interface system also manages the sharing of internal memory among multiple applications, and handles

input and output to and from attached hardware devices such as hard disks, printers, and dial-up ports. The hardware/software interface system also sends messages to each application (and, in certain case, to the end-user) regarding the status of operations and any errors that may have occurred. The hardware/software interface system can also offload the management of batch jobs (e.g., printing) so that the initiating application is freed from this work and can resume other processing and/or operations. On computers that can provide parallel processing, a hardware/software interface system also manages dividing a program so that it runs on more than one processor at a time.

[0086] A hardware/software interface system shell (simply referred to herein as a "shell") is an interactive end-user interface to a hardware/software interface system. (A shell may also be referred to as a "command interpreter" or, in an operating system, as an "operating system shell"). A shell is the outer layer of a hardware/software interface system that is directly accessible by application programs and/or end-users. In contrast to a shell, a kernel is a hardware/software interface system's innermost layer that interacts directly with the hardware components.

[0087] While it is envisioned that numerous embodiments of the present invention are particularly well-suited for computerized systems, nothing in this document is intended to limit the invention to such embodiments. On the contrary, as used herein the term "computer system" is intended to encompass any and all devices capable of storing and processing information and/or capable of using the stored information to control the behavior or execution of the device itself, regardless of whether such devices are electronic, mechanical, logical, or virtual in nature.

B. TRADITIONAL FILE-BASED STORAGE

[0088] In most computer systems today, "files" are units of storable information that may include the hardware/software interface system as well as application programs, data sets, and so forth. In all modern hardware/software interface systems (Windows, Unix, Linux, Mac OS, virtual machine systems, and so forth), files are the basic discrete (storable and retrievable) units of information (e.g., data, programs, and so forth) that can be manipulated by the hardware/software interface system. Groups of files are generally organized in "folders." In Microsoft Windows, the Macintosh OS, and other hardware/software interface systems, a folder

is a collection of files that can be retrieved, moved, and otherwise manipulated as single units of information. These folders, in turn, are organized in a tree-based hierarchical arrangement called a "directory" (discussed in more detail herein below). In certain other hardware/software interface systems, such as DOS, z/OS and most Unix-based operating systems, the terms "directory" and/or "folder" are interchangeable, and early Apple computer systems (e.g., the Apple IIe) used the term "catalog" instead of directory; however, as used herein, all of these terms are deemed to be synonymous and interchangeable and are intended to further include all other equivalent terms for and references to hierarchical information storage structures and their folder and file components.

[0089] Traditionally, a directory (a.k.a. a directory of folders) is a tree-based hierarchical structure wherein files are grouped into folders and folder, in turn, are arranged according to relative nodal locations that comprise the directory tree. For example, as illustrated in Fig. 2A, a DOS-based file system base folder (or "root directory") 212 may comprise a plurality of folders 214, each of which may further comprise additional folders (as "subfolders" of that particular folder) 216, and each of these may also comprise additional folders 218 ad infinitum. Each of these folders may have one or more files 220 although, at the hardware/software interface system level, the individual files in a folder have nothing in common other than their location in the tree hierarchy. Not surprisingly, this approach of organizing files into folder hierarchies indirectly reflects the physical organization of typical storage media used to store these files (e.g., hard disks, floppy disks, CD-ROMs, etc.).

[0090] In addition to the foregoing, each folder is a container for its subfolders and its files—that is, each folder owns its subfolders and files. For example, when a folder is deleted by the hardware/software interface system, that folder's subfolders and files are also deleted (which, in the case of each subfolder, further includes its own subfolders and files recursively). Likewise, each file is generally owned by only one folder and, although a file can be copied and the copy located in a different folder, a copy of a file is itself a distinct and separate unit that has no direct connection to the original (e.g., changes to the original file are not mirrored in the copy file at the hardware/software interface system level). In this regard, files and folders are therefore characteristically "physical" in nature because folders are the treated like physical

containers, and files are treated as discrete and separate physical elements inside these containers.

II. WINFS STORAGE PLATFORM FOR ORGANIZING, SEARCHING, AND SHARING DATA

[0091] The present invention, in combination with the related inventions incorporated by reference as discussed earlier herein, is directed to a storage platform for organizing, searching, and sharing data. The storage platform of the present invention extends and broadens the data platform beyond the kinds of existing file systems and database systems discussed above, and is designed to be the store for all types of data, including a new form of data called Items.

C. GLOSSARY

[0092] As used herein and in the claims, the following terms have the following meanings:

- An "Item" is an unit of storable information accessible to a hardware/software interface system that, unlike a simple file, is an object having a basic set of properties that are commonly supported across all objects exposed to an end-user by the hardware/software interface system shell. Items also have properties and relationships that are commonly supported across all Item types including features that allow new properties and relationships to be introduced (and discussed in great detail later herein).
- An "operating system" (OS) is a special program that acts as an intermediary between application programs and computer hardware. An operating system comprises, in most cases, a shell and a kernel.
- A "hardware/software interface system" is software, or a combination of hardware and software, that serves as the interface between the underlying hardware components of a computer system and applications that execute on the computer system. A hardware/software interface system typically comprises (and, in some embodiments, may solely consist of) an operating system. A

hardware/software interface system may also comprise a virtual machine manager (VMM), a Common Language Runtime (CLR) or its functional equivalent, a Java Virtual Machine (JVM) or its functional equivalent, or other such software components in the place of or in addition to the operating system in a computer system. The purpose of a hardware/software interface system is to provide an environment in which a user can execute application programs. The goal of any hardware/software interface system is to make the computer system convenient to use, as well as utilize the computer hardware in an efficient manner.

D. STORAGE PLATFORM OVERVIEW

[0093] Referring to Fig. 3, a storage platform 300 comprises a data store 302 implemented on a database engine 314. In one embodiment, the database engine comprises a relational database engine with object relational extensions. In one embodiment, the relational database engine 314 comprises the Microsoft SQL Server relational database engine. The data store 302 implements a data model 304 that supports the organization, searching, sharing, synchronization, and security of data. Specific types of data are described in schemas, such as schemas 340, and the storage platform 300 provides tools 346 for deploying those schemas as well as for extending those schemas, as described more fully below.

[0094] A change tracking mechanism 306 implemented within the data store 302 provides the ability track changes to the data store. The data store 302 also provides security capabilities 308 and a promotion/demotion capability 310, both of which are discussed more fully below. The data store 302 also provides a set of application programming interfaces 312 to expose the capabilities of the data store 302 to other storage platform components and application programs (*e.g.*, application programs 350a, 350b, and 350c) that utilize the storage platform. The storage platform of the present invention still further comprises an application programming interfaces (API) 322, which enables application programs, such as application programs 350a, 350b, and 350c, to access all of the foregoing capabilities of the storage platform and to access the data described in the schemas. The storage platform API 322 may be used by application programs in combination with other APIs, such as the OLE DB API 324 and the Microsoft Windows Win32 API 326.

[0095] The storage platform 300 of the present invention may provide a variety of services 328 to application programs, including a synchronization service 330 that facilitates the sharing of data among users or systems. For example, the synchronization service 330 may enable interoperability with other data stores 340 having the same format as data store 302, as well as access to data stores 342 having other formats. The storage platform 300 also provides file system capabilities that allow interoperability of the data store 302 with existing file systems, such as the Windows NTFS files system 318. In at least some embodiments, the storage platform 320 may also provide application programs with additional capabilities for enabling data to be acted upon and for enabling interaction with other systems. These capabilities may be embodied in the form of additional services 328, such as an Info Agent service 334 and a notification service 332, as well as in the form of other utilities 336.

[0096] In at least some embodiments, the storage platform is embodied in, or forms an integral part of, the hardware/software interface system of a computer system. For example, and without limitation, the storage platform of the present invention may be embodied in, or form an integral part of, an operating system, a virtual machine manager (VMM), a Common Language Runtime (CLR) or its functional equivalent, or a Java Virtual Machine (JVM) or its functional equivalent. Through its common storage foundation, and schematized data, the storage platform of the present invention enables more efficient application development for consumers, knowledge workers and enterprises. It offers a rich and extensible programming surface area that not only makes available the capabilities inherent in its data model, but also embraces and extends existing file system and database access methods.

[0097] In the following description, and in various ones of the figures, the storage platform 300 of the present invention may be referred to as "WinFS." However, use of this name to refer to the storage platform is solely for convenience of description and is not intended to be limiting in any way.

E. THE DATA MODEL

[0098] The data store 302 of the storage platform 300 of the present invention implements a data model that supports the organization, searching, sharing, synchronization, and security of data that resides in the store. In the data model of the present invention, an "Item" is

the fundamental unit of storage information. The data model provides a mechanism for declaring Items and Item extensions and for establishing relationships between Items and for organizing Items in Item Folders and in Categories, as described more fully below.

[0099] The data model relies on two primitive mechanisms, Types and Relationships. Types are structures which provide a format which governs the form of an instance of the Type. The format is expressed as an ordered set of Properties. A Property is a name for a value or set of values of a given Type. For example a USPostalAddress type might have the properties Street, City, Zip, State in which Street, City and State are of type String and Zip is of Type Int32. Street may be multi-valued (i.e. a set of values) allowing the address to have more than one value for the Street property. The system defines certain primitive types that can be used in the construction of other types – these include String, Binary, Boolean, Int16, Int32, Int64, Single, Double, Byte, DateTime, Decimal and GUID. The Properties of a Type may be defined using any of the primitive types or (with some restrictions noted below) any of the constructed types. For example a Location Type might be defined that had Properties Coordinate and Address where the Address Property is of Type USPostalAddress as described above. Properties may also be required or optional.

[0100] Relationships can be declared and represent a mapping between the sets of instances of two types. For example there may be a Relationship declared between the Person Type and the Location Type called LivesAt which defines which people live at which locations. The Relationship has a name, two endpoints, namely a source endpoint and a target endpoint. Relationships may also have an ordered set of properties. Both the Source and Target endpoints have a Name and a Type. For example the LivesAt Relationship has a Source called Occupant of Type Person and a Target called Dwelling of Type Location and in addition has properties StartDate and EndDate indicating the period of time for which the occupant lived at the dwelling. Note that a Person may live at multiple dwellings over time and a dwelling may have multiple occupants so the most likely place to put the StartDate and EndDate information is on the relationship itself.

[0101] Relationships define a mapping between instances that is constrained by the types given as the endpoint types. For example the LivesAt relationship cannot be a relationship in which an Automobile is the Occupant because an Automobile is not a Person.

[0102] The data model does allow the definition of a subtype-supertype relationship between types. The subtype-supertype relationship also known as the BaseType relationship is defined in such a way that if Type A is a BaseType for Type B it must be the case that every instance of B is also an instance of A. Another way of expressing this is that every instance that conforms to B must also conform to A. If, for example A has a property Name of Type String while B has a property Age of Type Int16, it follows that any instance of B must have both a Name and an Age. The type hierarchy may be envisaged as an tree with a single supertype at the root. The branches from the root provide the first level subtypes, the branches at this level provide the second level subtypes and so on to the leaf-most subtypes which themselves do not have any subtypes. The tree is not constrained to be of a uniform depth but cannot contain any cycles. A given Type may have zero or many subtypes and zero or one super type. A given instance may conform to at most one type together with that type's super types. To put it another way, for a given instance at any level in the tree the instance may conform to at most one subtype at that level. A type is said to be Abstract if instances of the type must also be an instance of a subtype of the type.

1. Items

[0103] An Item is a unit of storable information that, unlike a simple file, is an object having a basic set of properties that are commonly supported across all objects exposed to an end-user or application program by the storage platform. Items also have properties and relationships that are commonly supported across all Item types including features that allow new properties and relationships to be introduced, as discussed below.

[0104] Items are the objects for common operations such as copy, delete, move, open, print, backup, restore, replicate, and so forth. Items are the units that can be stored and retrieved, and all forms of storable information manipulated by the storage platform exist as Items, properties of Items, or Relationships between Items, each of which is discussed in greater detail herein below.

[0105] Items are intended to represent real-world and readily-understandable units of data like Contacts, People, Services, Locations, Documents (of all various sorts), and so on. Fig. 5A is a block diagram illustrating the structure of an Item. The unqualified name of the Item is "Location". The qualified name of the Item is "Core.Location" which indicates that this Item

structure is defined as a specific type of Item in the Core Schema. (The Core Schema is discussed in more detail later herein.)

[0106] The Location Item has a plurality of properties including EAddresses, MetropolitanRegion, Neighborhood, and PostalAddresses. The specific type of property for each is indicated immediately following the property name and is separated from the property name by a colon (":"). To the right of the type name, the number of values permitted for that property type is indicated between brackets ("[ ]") wherein an asterisk ("*") to the right of the colon (":") indicates an unspecified and/or unlimited number ("many"). A "1" to the right of the colon indicates that there can be at most one value. A zero ("0") to the left of the colon indicates that the property is optional (there may be no value at all). A "1" to the left of the colon indicates that there must be at least one value (the property is required). Neighborhood and MetropolitanRegion are both of type "nvarchar" (or equivalent) which is a predefined data type or "simple type" (and denoted herein by the lack of capitalization). EAddresses and PostalAddresses, however, are properties of defined types or "complex types" (as denoted herein by capitalization) of types EAddress and PostalAddress respectively. A complex type is type that is derived from one or more simple data types and/or from other complex types. The complex types for the properties of an Item also constitute "nested elements" since the details of the complex type are nested into the immediate Item to define its properties, and the information pertaining to these complex types is maintained with the Item that has these properties (within the Item's boundary, as discussed later herein). These concepts of typing are well known and readily appreciated by those of skill in the art.

[0107] Fig. 5B is a block diagram illustrating the complex property types PostalAddress and EAddress. The PostalAddress property type defines that an Item of property type PostalAddress can be expected to have zero or one City values, zero or one CountryCode values, zero or one MailStop values, and any number (zero to many) of PostalAddressTypes, and so on and so forth. In this way, the shape of the data for a particular property in an Item is hereby defined. The EAddress property type is similarly defined as shown. Although optionally used herein this Application, another way to represent the complex types in the Location Item is to draw the Item with the individual properties of each complex type listed therein. Fig. 5C is a block diagram illustrating the Location Item wherein its complex types are further described.

However, it should be understood that this alternative representation of the Location Item in this Fig. 5C is for the exact same Item illustrated in Fig. 5A. The storage platform of the present invention also allows subtyping whereby one property type can be a subtype of another (where the one property type inherits the properties of another, parent property type).

[0108] Similar to but distinct from properties and their property types, Items inherently represent their own Item Types that can also be the subject of subtyping. In other words, the storage platform in several embodiments of the present invention allows an Item to be a subtype of another Item (whereby the one Item inherits the properties of the other, parent Item). Moreover, for various embodiments of the present invention, every Item is a subtype of the "Item" Item type which is the first and foundational Item type found in the Base Schema. (The Base Schema will also be discussed in detail later herein.) Fig. 6A illustrates an Item, the Location Item in this Instance, as being a subtype of the Item Item type found in the Base Schema. In this drawing, the arrow indicates that the Location Item (like all other Items) is a subtype of the Item Item type. The Item Item type, as the foundational Item from which all other Items are derived, has a number of important properties such as ItemId and various timestamps, and thereby defines the standard properties of all Items in an operating system. In the present figure, these properties of the Item Item type are inherited by Location and thereby become properties of Location.

[0109] Another way to represent the properties in the Location Item inherited from the Item Item type is to draw Location with the individual properties of each property type from the parent Item listed therein. Fig. 6B is a block diagram illustrating the Location Item wherein its inherited types described in addition to its immediate properties. It should be noted and understood that this Item is the same Item illustrated in Fig. 5A, although in the present figure Location is illustrated with all of its properties, both immediate—shown in both this figure and Fig. 5A—and inherited—shown in this figure but not Fig. 5A (whereas in Fig. 5A these properties are referenced by showing with an arrow that the Location Item is a subtype of the Item Item type).

[0110] Items are stand-alone objects; thus, if you delete an Item, all of the Items immediate and inherited properties are also deleted. Similarly, when retrieving an Item, what is received is the Item and all of its immediate and inherited properties (including the information

pertaining to its complex property types). Certain embodiments of the present invention may enable one to request a subset of properties when retrieving a specific Item; however, the default for many such embodiments is to provide the Item with all of its immediate and inherited properties when retrieved. Moreover, the properties of Items can also be extended by adding new properties to the existing properties of that Item's type. These "extensions" are thereafter bona fide properties of the Item and subtypes of that Item type may automatically include the extension properties.

[0111] The "boundary" of the Item is represented by its properties (including complex property types, extensions, and so forth). An Item's boundary also represents the limit of an operation performed on an Item such as copy, delete, move, create, and so on. For example, in several embodiments of the present invention, when an Item is copied, everything within that Item's boundary is also copied. For each Item, the boundary encompasses the following:

- The Item Type of the Item and, if the Item is a subtype of another Item (as is the case in several embodiments of the present invention where all Items are derived from a single Item and Item Type in the Base Schema), any applicable subtype information (that is, information pertaining to the parent Item Type). If the original Item being copied is a subtype of another Item, the copy may also be a subtype of that same Item.
- The Item's complex-type properties and extensions, if any. If the original Item has properties of complex types (native or extended), the copy may also have the same complex types.
- The Item's records on "ownership relationships", that is, the Item's own list of what other Items (the "Target Items") are owned by the present Item (the "Owning Item"). This is particularly relevant in regard to Item Folders, discussed more fully below, and the rule stated below that all Items must belong to at least one Item Folder. Moreover, in regard to embedded items—discussed more fully below—an embedded item is considered to be part of the Item in which it is embedded for operations such as copy, delete, and the like.

2. Item Identification

[0112] Items are uniquely identified within the global items space with an ItemID. The Base.Item type defines a field ItemID of type GUID that stores the identity for the Item. An Item must have exactly one identity in the data store 302.

[0113] An item reference is a data structure that contains information to locate and identify an Item. In the data model, an abstract type is defined named ItemReference from which all item reference types derive. The ItemReference type defines a virtual method named Resolve. The Resolve method resolves the ItemReference and returns an Item. This method is overridden by the concrete subtypes of ItemReference, which implement a function that retrieves an Item given a reference. The Resolve method is invoked as part of the storage platform API 322.

[0114] ItemIDReference is a subtype of ItemReference. It defines a Locator and an ItemID field. The Locator field names (i.e. identifies) an item domain. It is processed by a locator resolution method that can resolve the value of the Locator to an item domain. The ItemID field is of type ItemID

[0115] ItemPathReference is a specialization of ItemReference that defines a Locator and a Path field. The Locator field identifies an item domain. It is processed by a locator resolution method that can resolve the value of the Locator to an item domain. The Path field contains a (relative) path in the storage platform namespace rooted at the item domain provided by the Locator.

[0116] This type of reference cannot be used in a set operation. The reference must generally be resolved through a path resolution process. The Resolve method of the storage platform API 322 provides this functionality.

[0117] The reference forms discussed above are represented through the reference type hierarchy illustrated in Fig. 11. Additional reference types that inherit from these types can be defined in the schemas. They can be used in a relationship declaration as type of the target field.

3. Item Folders and Categories

[0118] As discussed more fully below, groups of Items can are organized into special Items called Item Folders (which are not to be confused with file folders). Unlike in most file systems, however, an Item can belong to more than one Item Folder, such that when an Item is

accessed in one Item Folder and revised, this revised Item can then be accessed directly from another Item folder. In essence, although access to an Item may occur from different Item Folders, what is actually being accessed is in fact the very same Item. However, an Item Folder does not necessarily own all of its member Items, or may simply co-own Items in conjunction with other folders, such that the deletion of an Item Folder does not necessarily result in the deletion of the Item. Nevertheless, in several embodiments of the present invention, an Item must belong to at least one Item Folder so that if the sole Item Folder for a particular Item is deleted then, for some embodiments, the Item is automatically deleted or, in alternative embodiments, the Item automatically becomes a member of a default Item Folder (e.g., a "Trash Can" Item Folder conceptually similar to similarly-named folders used in various file-and-folder-based systems).

[0119] As also discussed more fully below, Items may also belong to Categories based on common described characteristic such as (a) an Item Type (or Types), (b) a specific immediate or inherited property (or properties), or (c) a specific value (or values) corresponding to an Item property. For example, a Item comprising specific properties for personal contact information might automatically belong to a Contact Category, and any Item having contact information properties would likewise automatically belong to this Category. Likewise, any Item having a location property with a value of "New York City" might automatically belong to a NewYorkCity Category.

[0120] Categories are conceptually different form Item Folders in that, whereas Item Folders may comprise Items that are not interrelated (i.e., without a common described characteristic), each Item in a Category has a common type, property, or value (a "commonality") that is described for that Category, and it is this commonality that forms the basis for its relationship to and among the other Items in the Category. Moreover, whereas an Item's membership in a particular Folder is not compulsory based on any particular aspect of that Item, for certain embodiments all Items having a commonality categorically related to a Category might automatically become a member of the Category at the hardware/software interface system level. Conceptually, Categories can also be thought of as virtual Item Folders whose membership is based on the results of a specific query (such as in the context of a

database), and Items that meet the conditions of this query (defined by the commonalities of the Category) would thus comprise the Category's membership.

[0121] Fig. 4 illustrates the structural relationship between Items, Item Folders, and Categories. A plurality of Items 402, 404, 406, 408, 410, 412, 414, 416, 418, and 420 are members of various Item Folders 422, 424, 426, 428, and 430. Some Items may belong to more than one Item Folder, e.g., Item 402 belong to Item Folders 422 and 424. Some Items, e.g., Item 402, 404, 406, 408, 410, and 412 are also members of one or more Categories 432, 434, and 436, while other times, e.g., Items 414, 416, 418, and 420, may belong to no Categories (although this is largely unlikely in certain embodiments where the possession of any property automatically implies membership in a Category, and thus an Item would have to be completely featureless in order not to be a member of any category in such an embodiment). In contrast to the hierarchical structure of folders, both Categories and Item Folders have structures more akin to directed graphs as shown. In any event, the Items, Item Folders, and Categories are all Items (albeit of different Item Types).

[0122] In contrast to files, folders, and directories, the Items, Item Folders, and Categories of the present invention are not characteristically "physical" in nature because they do not have conceptual equivalents of physical containers, and therefore Items may exist in more than one such location. The ability for Items to exist in more than one Item Folder location as well as being organized into Categories provides an enhanced and enriched degree of data manipulation and storage structure capabilities at the hardware/software interface level, beyond that currently available in the art.

4. Schemas

a) Base Schema

[0123] To provide a universal foundation for the creation and use of Items, various embodiments of the storage platform of the present invention comprise a Base Schema that establishes a conceptual framework for creating and organizing Items and properties. The Base Schema defines certain special types of Items and properties, and the features of these special foundational types from which subtypes can be further derived. The use of this Base Schema allows a programmer to conceptually distinguish Items (and their respective types) from

properties (and their respective types). Moreover, the Base Schema sets forth the foundational set of properties that all Items may possess as all Items (and their corresponding Item Types) are derived from this foundational Item in the Base Schema (and its corresponding Item Type).

[0124] As illustrated in Fig. 7, and in regard to several embodiments of the present invention, the Base Schema defines three top-level types: Item, Extension, and PropertyBase. As shown, the Item type is defined by the properties of this foundational "Item" Item type. In contrast, the top level property type "PropertyBase" has no predefined properties and is merely the anchor from which all other property types are derived and through which all derived property types are interrelated (being commonly derived from the single property type). The Extension type properties define which Item the extension extends as well as identification to distinguish one extension from another as an Item may have multiple extensions.

[0125] ItemFolder is a subtype of the Item Item type that, in addition to the properties inherited from Item, features a Relationship for establishing links to its members (if any), whereas both IdentityKey and Property are subtypes of PropertyBase. CategoryRef, in turn, is a subtype of IdentityKey.

b) Core Schema

[0126] Various embodiments of the storage platform of the present invention further comprise a Core Schema that provides a conceptual framework for top-level Items type structures. Fig. 8A is a block diagram illustrating Items in the Core Schema, and Fig. 8B is a block diagram illustrating the property types in the Core Schema. The distinction made between files with different extensions (*.com, *.exe, *.bat, *.sys, etc.) and other such criteria in file-and-folder-based systems is analogous to the function of the Core Schema. In the Item-based hardware/software interface system, the Core Schema defines a set of core Item types that, directly (by Item type) or indirectly (by Item subtype), characterize all Items into one or more Core Schema Item types which the Item-based hardware/software interface system understands and can directly process in a predetermined and predictable way. The predefined Item types reflect the most common Items in the Item-based hardware/software interface system and thus a level of efficiency is gained by the Item-based hardware/software interface system understanding these predefined Item types that comprise the Core Schema.

[0127] In certain embodiments, the Core Schema is not extendable—that is, no additional Item types can be subtyped directly from the Item type in the Base Schema except for the specific predefined derived Item types that are part of the Core Schema. By preventing extensions to the Core Schema (that is, by preventing the addition of new Items to the Core Schema), the storage platform mandates the use of the Core Schema Item types since every subsequent Item type is necessarily a subtype of a Core Schema Item type. This structure enables a reasonable degree of flexibility in defining additional Item types while also preserving the benefits of having a predefined set of core Item types.

[0128] For various embodiments of the present invention, and in reference to Fig. 8A, the specific Item types supported by the Core Schema may include one or more of the following:

- Categories: Items of this Item Type (and subtypes derived therefrom) represent valid Categories in the Item-based hardware/software interface system.
- Commodities: Items that are identifiable things of value.
- Devices: Items having a logical structure that supports information processing capabilities.
- Documents: Items with content that is not interpreted by the Item-based hardware/software interface system but is instead interpreted by an application program corresponding to the document type.
- Events: Items that record certain occurrences in the environment.
- Locations: Items representing physical locations (e.g., geographical locations).
- Messages: Items of communication between two or more principals (defined below).
- Principals: Items having at least one definitively provable identity aside from an ItemId (e.g., the identification of a person, organization, group, household, authority, service, etc.).
- Statements: Items having special information regarding the environment including, without limitation, policies, subscriptions, credentials, and so forth.

[0129] Likewise, and in reference to Fig. 8B, the specific property types supported by the Core Schema may include one or more of the following:

- Certificates (derived from the foundational PropertyBase type in the Base Schema)
- Principal Identity Keys (derived from the IdentityKey type in the Base Schema)
- Postal Address (derived from the Property type in the Base Schema)
- Rich Text (derived from the Property type in the Base Schema)
- EAddress (derived from the Property type in the Base Schema)
- IdentitySecurityPackage (derived from the Relationship type in the Base Schema)
- RoleOccupancy (derived from the Relationship type in the Base Schema)
- BasicPresence (derived from the Relationship type in the Base Schema)

These Items and Properties are further described by their respective properties set forth in Fig. 8A and Fig. 8B.

5. Relationships

[0130] Relationships are binary relationships where one Item is designated as source and the other Item as target. The source Item and the target Item are related by the relationship. The source Item generally controls the life-time of the relationship. That is, when the source Item is deleted, the relationship between the Items is also deleted.

[0131] Relationships are classified into: Containment and Reference relationships. The containment relationships control the life-time of the target Items, while the reference relationships do not provide any life-time management semantics. Fig. 12 illustrates the manner in which relationships are classified.

[0132] The Containment relationship types are further classified into Holding and Embedding relationships. When all holding relationships to an Item are removed, the Item is deleted. A holding relationship controls the life-time of the target through a reference counting mechanism. The embedding relationships enable modeling of compound Items and can be thought of as exclusive holding relationships. An Item can be a target of one or more holding relationships; but an Item can be target of exactly one embedding relationship. An Item that is a target of an embedding relationship can not be a target of any other holding or embedding relationships.

[0133] Reference relationships do not control the lifetime of the target Item. They may be dangling – the target Item may not exist. Reference relationships can be used to model references to Items anywhere in the global Item name space (*i.e.* including remote data stores).

[0134] Fetching an Item does not automatically fetch its relationships. Applications must explicitly request the relationships of an Item. In addition, modifying a relationship does not modify the source or the target Item; similarly, adding a relationship does not affect the source/target Item.

a) Relationship Declaration

[0135] The explicit relationship types are defined with the following elements:

- A relationship name is specified in the Name attribute.
- Relationship type, one of the following: Holding, Embedding, Reference. This is specified in the Type attribute.
- Source and target endpoints. Each endpoint specifies a name and the type of the referenced Item.
- The source endpoint field is generally of type ItemID (not declared) and it must reference an Item in the same data store as the relationship instance.
- For Holding and Embedding relationships, the target endpoint field must be of type ItemIDReference and it must reference an Item in the same store as the relationship instance. For Reference relationships the target endpoint can be of any ItemReference type and can reference Items in other storage platform data stores.
- Optionally one or more fields of a scalar or PropertyBase type can be declared. These fields may contain data associated with the relationship.
- Relationship instances are stored in a global relationships table.
- Every relationship instance is uniquely identified by the combination (source ItemID, relationship ID). The relationship ID is unique within a given source ItemID for all relationships sourced in a given Item regardless of their type.

[0136] The source Item is the owner of the relationship. While an Item designated as owner controls the life time of the relationship, the relationship itself is separate from the Items it

relates. The storage platform API 322 provides mechanisms for exposing relationships associated with an Item.

[0137] Here is an example of a relationship declaration:

```
<Relationship Name="Employment" BaseType="Reference" >
        <Source Name="Employee" ItemType="Contact.Person"/>
        <Target Name="Employer" ItemType="Contact.Organization"
          ReferenceType="ItemIDReference" />
        <Property Name="StartDate" Type="the storage
        platformTypes.DateTime" />
        <Property Name="EndDate" Type="the storage
        platformTypes.DateTime" />
        <Property Name="Office" Type="the storage
        platformTypes.DateTime" />
</Relationship>
```

[0138] This is an example of a Reference relationship. The relationship can not be created if the person Item that is referenced by the source reference does not exist. Also, if the person Item is deleted, the relationship instances between the person and organization are deleted. However, if the Organization Item is deleted, the relationship is not deleted and it is dangling.

b) Holding Relationship

[0139] Holding relationships are used to model reference count based life-time management of the target Items.

[0140] An Item can be a source endpoint for zero or more relationships to Items. An Item that is not an embedded Item can be a target of in one or more holding relationships.

[0141] The target endpoint reference type must be ItemIDReference and it must reference an Item in the same store as the relationship instance.

[0142] Holding relationships enforce lifetime management of the target endpoint. The creation of a holding relationship instance and the Item that it is targeting is an atomic operation. Additional holding relationship instances can be created that are targeting the same Item. When the last holding relationship instance with a given Item as target endpoint is deleted the target Item is also deleted.

[0143] The types of the endpoint Items specified in the relationship declaration will generally be enforced when an instance of the relationship is created. The types of the endpoint Items can not be changed after the relationship is established.

[0144] Holding relationships play a key role in forming the Item namespace. They contain the "Name" property that defines the name of the target Item relative to the source Item. This relative name is unique for all the holding relationships sourced from a given Item. The ordered list of this relative names starting from the root Item to a given Item forms the full name to the Item.

[0145] The holding relationships form a directed acyclic graph (DAG). When a holding relationship is created the system ensures that a cycle is not created, thus ensuring that the Item namespace forms a DAG.

[0146] While the holding relationship controls the life time of the target Item, it does not control the operational consistency of the target endpoint Item. The target Item is operationally independent from the Item that owns it through a holding relationship. Copy, Move, Backup and other operations on an Item that is a source of a holding relationship do not affect the Item that is a target of the same relationship – for example that is, backing up a Folder Item does not automatically backup all the Items in the folder (targets of the FolderMember relationship).

[0147] The following is an example of a holding relationship:

```
<Relationship Name="FolderMembers" BaseType="Holding" >
        <Source Name="Folder" ItemType="Base.Folder"/>
        <Target Name="Item" ItemType="Base.Item"
          ReferenceType="ItemIDReference" />
</Relationship>
```

[0148] The FolderMembers relationship enables the concept of a Folder as a generic collection of Items.

c) Embedding Relationships

[0149] Embedding relationships model the concept of exclusive control of the lifetime of the target Item. They enable the concept of compound Items.

[0150] The creation of an embedding relationship instance and the Item that it is targeting is an atomic operation. An Item can be a source of zero or more embedding relationship. However, an Item can be a target of one and only one embedding relationship. An Item that is a target of an embedding relationship can not be a target of a holding relationship.

[0151] The target endpoint reference type must be ItemIDReference and it must reference an Item in the same data store as the relationship instance.

[0152] The types of the endpoint Items specified in the relationship declaration will generally be enforced when an instance of the.relationship is created. The types of the endpoint Items can not be changed after the relationship is established.

[0153] Embedding relationships control the operational consistency of the target endpoint. For example the operation of serializing of an Item may include serialization of all the embedding relationships that source from that Item as well as all of their targets; copying an Item also copies all its embedded Items.

[0154] The following is an example declaration:

```
<Relationship Name="ArchiveMembers" BaseType="Embedding" >
        <Source Name="Archive" ItemType="Zip.Archive" />
        <Target Name="Member" ItemType="Base.Item "
          ReferenceType="ItemIDReference" />
        <Property Name="ZipSize" Type="the storage
        platformTypes.bigint" />
        <Property Name="SizeReduction" Type="the storage
        platformTypes.float" />
</Relationship>
```

d) Reference Relationships

[0155] The reference relationship does not control life time of the Item it references. Even more, the reference relationships do not guarantee the existence of the target, nor do they guarantee the type of the target as specified in the relationship declaration. This means that the reference relationships can be dangling. Also, the reference relationship can reference Items in other data stores. Reference relationships can be thought of as a concept similar to links in web pages.

[0156] An example of reference relationship declaration is the following:

```
<Relationship Name="DocumentAuthor" BaseType="Reference" >
        <Sourc ItemType="Document"
        ItemType="Base.Document"/>
        <Target ItemType="Author" ItemType="Base.Author"
          ReferenceType="ItemIDReference" />
        <Property Type="Role" Type="Core.CategoryRef" />
        <Property Type="DisplayName" Type="the storage
        platformTypes.nvarchar(256)" />
</Relationship>
```

[0157] Any reference type is allowed in the target endpoint. The Items that participate in a reference relationship can be of any Item type.

[0158] Reference relationships are used to model most non-lifetime management relationships between Items. Since the existence of the target is not enforced, the reference relationship is convenient to model loosely-coupled relationships. The reference relationship can be used to target Items in other data stores including stores on other computers.

e) Rules and Constraints

[0159] The following additional rules and constraints apply for relationships:

- An Item must be a target of (exactly one embedding relationship) or (one or more holding relationships). One exception is the root Item. An Item can be a target of zero or more reference relationships
- An Item that is a target of embedding relationship can not be source of holding relationships. It can be a source of reference relationships.
- An Item can not be a source of holding relationship if it is promoted from file. It can be a source of embedding relationships and reference relationships.
- An Item can that is promoted from a file can not be a target of an embedding relationship.

f) Ordering of Relationships

[0160] In at least one embodiment, the storage platform of the present invention supports ordering of relationships. The ordering is achieved through a property named "Order" in the base relationship definition. There is no uniqueness constraint on the Order field. The order

of the relationships with the same "order" property value is not guaranteed, however it is guaranteed that they may be ordered after relationships with lower "order" value and before relationships with higher "order" field value.

[0161] Applications can get the relationships in the default order by ordering on the combination (SourceItemID, RelationshipID, Order). All relationship instances sourced from a given Item are ordered as a single collection regardless of the type of the relationships in the collection. This however guarantees that all relationships of a given type (*e.g.*, FolderMembers) are an ordered subset of the relationship collection for a given Item.

[0162] The data store API 312 for manipulating relationships implement a set of operations that support ordering of relationships. The following terms are introduced to help explain the operations:

- *RelFirst* is the first relationship in the ordered collection with order value *OrdFirst*;
- *RelLast* is the last relationship in the ordered collection with order value *OrdLast*;
- *RelX* is a given relationship in the collection with order value *OrdX*;
- *RelPrev* is a closest relationship in the collection to *RelX* with order value *OrdPrev* smaller then *OrdX*; and
- *RelNext* is a closest relationship in the collection to RelX with order value OrdNext greater then *OrdX*.

[0163] The operations include but are not limited to:

- *InsertBeforeFirst(SourceItemID, Relationship)* inserts the relationship as the first relationship in the collection. The value of the "Order" property of the new relationship may be smaller then OrdFirst.
- *InsertAfterLast(SourceItemID, Relationship)* inserts the relationship as the last relationship in the collection. The value of the "Order" property of the new relationship may be greater then OrdLast.
- *InsertAt(SourceItemID, ord, Relationship)* inserts a relationship with the specified value for the "Order" property.

- *InsertBefore(SourceItemID, ord, Relationship)* inserts the relationship before the relationship with the given order value. The new relationship may be assigned "Order" value that is between OrdPrev and ord, noninclusive.
- *InsertAfter(SourceItemID, ord, Relationship)* inserts the relationship after the relationship with the given order value. The new relationship may be assigned "Order" value that is between ord and OrdNext, non-inclusive.
- *MoveBefore(SourceItemID, ord, RelationshipID)* moves the relationship with given relationship ID before the relationship with specified "Order" value. The relationship may be assigned a new "Order" value that is between OrdPrev and ord, non-inclusive.
- *MoveAfter(SourceItemID, ord, RelationshipID)* moves the relationship with given relationship ID after the relationship with specified "Order" value. The relationship may be assigned a new order value that is between ord and OrdNext, non-inclusive.

[0164] As previously mentioned, every Item must be a member of an Item Folder. In terms of Relationships, every Item must have a relationship with an Item Folder. In several embodiments of the present invention, certain relationships are represented by Relationships existing between the Items.

[0165] As implemented for various embodiments of the present invention, a Relationship provides a directed binary relationship that is "extended" by one Item (the source) to another Item (the target). A Relationship is owned by the source Item (the Item that extended it), and thus the Relationship is removed if the source is removed (e.g., the Relationship is deleted when the source Item is deleted). Moreover, in certain instances, a Relationship may share ownership of (co-own) the target Item, and such ownership might be reflected in the IsOwned property (or its equivalent) of the Relationship (as shown in Fig. 7 for the Relationship property type). In these embodiments, creation of a new IsOwned Relationship automatically increments a reference count on the target Item, and deletion of such a Relationship may decrement the reference count on the target Item. For these specific embodiments, Items continue to exist if they have a reference count greater than zero, and are automatically deleted if and when the count reaches zero. Again, an Item Folder is an Item that has (or is capable of

having) a set of Relationships to other Items, these other Items comprising the membership of the Item Folder. Other actual implementations of Relationships are possible and anticipated by the present invention to achieve the functionality described herein.

[0166] Regardless of actual implementation, a Relationship is a selectable connection from one object to another. The ability for an Item to belong to more than one Item Folder, as well as to one or more Categories, and whether these Items, Folders, and Categories are public or private, is determined by the meanings given to the existence (or lack thereof) in an Item-based structure. These logical Relationships are the meanings assigned to a set of Relationships, regardless of physical implementation, which are specifically employed to achieve the functionality described herein. Logical Relationships are established between the Item and its Item Folder(s) or Categories (and vice versa) because, in essence, Item Folders and Categories are each a special type of Item. Consequently, Item Folders and Categories can be acted upon the same way as any other Item—copied, added to an email message, embedded in a document, and so and so forth without limitation—and Item Folders and Categories can be serialized and de-serialized (imported and exported) using the same mechanisms as for other Items. (For example, in XML all Items might have a serialization format, and this format applies equally to Item Folders, Categories, and Items.)

[0167] The aforementioned Relationships, which represent the relationship between an Item and it Item Folder(s) can logically extend from the Item to the Item Folder, from the Item Folder to the Item, or both. A Relationship that logically extends from an Item to an Item Folder denotes that the Item Folder is public to that Item and shares its membership information with that Item; conversely, the lack of a logical Relationship from an Item to an Item Folder denotes that the Item Folder is private to that Item and does not share its membership information with that Item. Similarly, a Relationship that logically extends from an Item Folder to an Item denotes that the Item is public and sharable to that Item Folder, whereas the lack of a logical Relationship from the Item Folder to the Item denotes that the Item is private and non-sharable. Consequently, when an Item Folder is exported to another system, it is the "public" Items that are shared in the new context, and when an Item searches its Items Folders for other, sharable Items, it is the "public" Item Folders that provide the Item with information regarding sharable Items that belong thereto.

[0168] Fig. 9 is a block diagram illustrating an Item Folder (which, again, is an Item itself), its member Items, and the interconnecting Relationships between the Item Folder and its member Items. The Item Folder 900 has as members a plurality of Items 902, 904, and 906. Item Folder 900 has a Relationship 912 from itself to Item 902 which denotes that the Item 902 is public and sharable to Item Folder 900, its members 904 and 906, and any other Item Folders, Categories, or Items (not shown) that might access Item Folder 900. However, there is no Relationship from Item 902 to the Item Folder 900 which denotes that Item Folder 900 is private to Item 902 and does not share its membership information with Item 902. Item 904, on the other hand, does have a Relationship 924 from itself to Item Folder 900 which denotes that the Item Folder 900 is public and shares its membership information with Item 904. However, there is no Relationship from the Item Folder 900 to Item 904 which denotes that Item 904 is private and not sharable to Item Folder 900, its other members 902 and 906, and any other Item Folders, Categories, or Items (not shown) that might access Item Folder 900. In contrast with its Relationships (or lack thereof) to Items 902 and 904, Item Folder 900 has a Relationship 916 from itself to the Item 906 and Item 906 has a Relationship 926 back to Item Folder 900, which together denote that Item 906 is public and sharable to Item Folder 900, its members 902 and 904, and any other Item Folders, Categories, or Items (not shown) that might access Item Folder 900, and that Item Folder 900 is public and shares its membership information with Item 906.

[0169] As previously discussed, the Items in an Item Folder do not need to share a commonality because Item Folders are not "described." Categories, on the other hand, are described by a commonality that is common to all of its member Items. Consequently the membership of a Category is inherently limited to Items having the described commonality and, in certain embodiments, all Items meeting the description of a Category are automatically made members of the Category. Thus, whereas Item Folders allow trivial type structures to be represented by their membership, Categories allow membership based on the defined commonality.

[0170] Of course Category descriptions are logical in nature, and therefore a Category may be described by any logical representation of types, properties, and/or values. For example, a logical representation for a Category may be its membership to comprise Items have one of two properties or both. If these described properties for the Category are "A" and "B", then the

Categories membership may comprise Items having property A but not B, Items having property B but not A, and Items having both properties A and B. This logical representation of properties is described by the logical operator "OR" where the set of members described by the Category are Items having property A OR B. Similar logical operands (including without limitation "AND", "XOR", and "NOT" alone or in combination) can also be used describe a category as will be appreciated by those of skill in the art.

[0171] Despite the distinction between Item Folders (not described) and Categories (described), Categories Relationship to Items and Items Relationship to Categories essentially the same way as disclosed herein above for Item Folders and Items in many embodiments of the present invention.

[0172] Fig. 10 is a block diagram illustrating a Category (which, again, is an Item itself), its member Items, and the interconnecting Relationships between the Category and its member Items. The Category 1000 has as members a plurality of Items 1002, 1004, and 1006, all of which share some combination of common properties, values, or types 1008 as described (commonality description 1008') by the Category 1000. Category 1000 has a Relationship 1012 from itself to Item 1002 which denotes that the Item 1002 is public and sharable to Category 1000, its members 1004 and 1006, and any other Categories, Item Folders, or Items (not shown) that might access Category 1000. However, there is no Relationship from the Item 1002 to the Category 1000 which denotes that Category 1000 is private to Item 1002 and does not share its membership information with Item 1002. Item 1004, on the other hand, does have a . Relationship 1024 from itself to Category 1000 which denotes that the Category 1000 is public and shares its membership information with Item 1004. However, there is no Relationship extended from Category 1000 to the Item 1004 which denotes that Item 1004 is private and not sharable to Category 1000, its other members 1002 and 1006, and any other Categories, Item Folders, or Items (not shown) that might access Category 1000. In contrast to its Relationships (or lack thereof) with Items 1002 and 1004, Category 1000 has a Relationship 1016 from itself to Item 1006 and Item 1006 has a Relationship 1026 back to Category 1000, which altogether denotes that Item 1006 is public and sharable to Category 1000, its Item members 1002 and 1004, and any other Categories, Item Folders, or Items (not shown) that might access Category

1000, and that the Category 1000 is public and shares its membership information with Item 1006.

[0173] Finally, because Categories and Item Folders are themselves Items, and Items may Relationship to each other, Categories may Relationship to Item Folders and vice versa, and Categories, Item Folders, and Items can Relationship to other Categories, Item Folders, and Item respectively in certain alternative embodiments. However, in various embodiments, Item Folder structures and/or Category structures are prohibited, at the hardware/software interface system level, from containing cycles. Where Item Folder and Category structures are akin to directed graphs, the embodiments that prohibit cycles are akin to directed acyclic graphs (DAGs) which, by mathematical definition in the art of graph theory, are directed graphs wherein no path starts and ends at the same vertex.

6. Extensibility

[0174] The storage platform is intended to be provided with an initial set of schemas 340, as described above. In addition, however, in at least some embodiments, the storage platform allows customers, including independent software vendor (ISVs), to create new schemas 344 (*i.e.* new Item and Nested Element types). This section addresses the mechanism for creating such schemas by extending the Item types and Nested Element types (or simply "Element" types) defined in the initial set of schemas 340.

[0175] Preferably, extension of the initial set of Item and Nested Element types is constrained as follows:

- an ISV is allowed to introduce new Item types, *i.e.* subtype Base.Item;
- an ISV is allowed to introduce new Nested Element types, *i.e.* subtype Base.NestedElement;
- an ISV is allowed to introduce new extensions, *i.e.* subtype Base.NestedElement; but,
- an ISV cannot subtype any types (Item, Nested Element, or Extension types) defined by the initial set of storage platform schemas 340.

[0176] Since an Item type or Nested Element type defined by the initial set of storage platform schemas may not exactly match an ISV application's need, it is necessary to allow ISVs

to customize the type. This is allowed with the notion of Extensions. Extensions are strongly typed instances but (a) they cannot exist independently and (b) they must be attached to an Item or Nested Element.

[0177] In addition to addressing the need for schema extensibility, Extensions are also intended to address the "multi-typing" issue. Since, in some embodiments, the storage platform may not support multiple inheritance or overlapping subtypes, applications can use Extensions as a way to model overlapping type instances (*e.g.* Document is a legal document as well a secure document).

a) Item extensions

[0178] To provide Item extensibility, the data model further defines an abstract type named Base.Extension. This is a root type for the hierarchy of extension types. Applications can subtype Base.Extension to create specific extension types.

[0179] The Base.Extension type is defined in the Base schema as follows:

```
<Type Name="Base.Extension" IsAbstract="True">
        <Propety Name="ItemID"
                  Type="the storage platformTypes.uniqueIdentified"
                  Nullable="false"
                  MultiValued="false"/>
        <Property Name="ExtensionID"
                  Type="the storage platformTypes.uniqueidentified"
                  Nullable="false"
                  MultiValued="false"/>
</Type>
```

[0180] The ItemID field contains the ItemID of the item that the extension is associated with. An Item with this ItemID must exist. The extension can not be created if the item with the given ItemID does not exist. When the Item is deleted all the extensions with the same ItemID are deleted. The tuple (ItemID,ExtensionID) uniquely identifies an extension instance.

[0181] The structure of an extension type is similar to that of an item type:

- Extension types have fields;
- Fields can be of primitive or nested element types; and
- Extension types can be sub-typed.

[0182] The following restrictions apply for extension types

- Extensions can not be sources and targets of relationships;
- Extension type instances can not exist independently from an item; and
- Extension types can not be used as field types in the storage platform type definitions

[0183] There are no constraints on the types of extensions that can be associated with a given Item type. Any extension type is allowed to extend any item type. When multiple extension instances are attached to an item, they are independent from each other in both structure and behavior.

[0184] The extension instances are stored and accessed separately from the item. All extension type instances are accessible from a global extension view. An efficient query can be composed that will return all the instances of a given type of extension regardless of what type of item they are associated with. The storage platform APIs provides a programming model that can store, retrieve and modify extensions on items.

[0185] The extension types can be type sub-typed using the storage platform single inheritance model. Deriving from an extension type creates a new extension type. The structure or the behavior of an extension cannot override or replace the structure or behaviors of the item type hierarchy. Similar to Item types, Extension type instances can be directly accessed through the view associated with the extension type. The ItemID of the extension indicates which item they belong to and can be used to retrieve the corresponding Item object from the global Item view. The extensions are considered part of the item for the purposes of operational consistency. The Copy/Move, Backup/Restore and other common operations that the storage platform defines may operate on the extensions as part of the item.

[0186] Consider the following example. A Contact type is defined in the Windows Type set.

```
<Type Name="Contact" BaseType="Base.Item" >
        <Property Name="Name"
                  Type="String"
                  Nullable="false"
                  MultiValued="false"/>
        <Property Name="Address"
```

```
                Type="Address"
                Nullable="true"
                MultiValued="false"/>

    </Type>
```

[0187] A CRM application developer would like to attach a CRM application extension to the contacts stored in the storage platform. The application developer would define a CRM extension that would contain the additional data structure that the application can manipulate.

```
<Type Name="CRMExtension" BaseType="Base.Extension" >
    <Property Name="CustomerID"
                Type="String"
                Nullable="false"
                MultiValued="false"/>
    ...
</Type>
```

[0188] An HR application developer may want to also attach additional data with the Contact. This data is independent from the CRM application data. Again the application developer can create an extension

```
<Type Name="HRExtension" EBaseType="Base.Extension" >
    <Property Name="EmployeeID"
                Type="String"
                Nullable="false"
                MultiValued="false"/>
    ...
</Type>
```

[0189] CRMExtension and HRExtension are two independent extensions that can be attached to Contact items. They are created and accessed independently of each other.

[0190] In the above example, the fields and methods of the CRMExtension type cannot override fields or methods of the Contact hierarchy. It should be noted that instances of the CRMExtension type can be attached to Item types other than Contact.

[0191] When the Contact item is retrieved, its item extensions are not automatically retrieved. Given a Contact item, its related item extensions can be accessed by querying the global extension view for extensions with the same ItemId.

[0192] All CRMExtension extensions in the system can be accessed through the CRMExtension type view, regardless of which item they belong to. All item extension of an item share the same item id. In the above example, the Contact item instance and the attached CRMExtension and HRExtension instances the same ItemID.

[0193] The following table summarizes the similarities and differences between Item, Extension and NestedElement types:

Item vs Item Extension vs NestedElement

	Item	Item Extension	NestedElement
Item ID	Has its own item id	Shares the item id of the item	Does not have its own item id. Nested element is part of the item
Storage	Item hierarchy is stored in its own tables	Item extension hierarchy is stored in its own tables	Stored with item
Query/Search	Can query item tables	Can query item extension tables	Can generally be queried only within the containing item context
Query/Search scope	Can search across all instances of an item type	Can search across all instances of an item extension type	Can generally only search within nested element type instances of a singe (containing) item
Relationship semantics	Can have Relationships to items	No Relationships to item extensions	No Relationships to nested elements
Association to items	Can be related to other items via holding, embedded and soft Relationships	Can generally only be related via extensions. The extension semantics is similar to embedded item	Related to item via fields. Nested elements are part of the item

semantics

b) Extending NestedElement types

[0194] Nested Element types are not extended with the same mechanism as the Item types. Extensions of nested elements are stored and accessed with the same mechanisms as fields of nested element types.

[0195] The data model defines a root for nested element types named Element:

```
<Type Name="Element"
    IsAbstract="True">
        <Property Name="ElementID"
                  Type="the storage platformTypes.uniqueidentifier"
                  Nullable="false"
                  MultiValued="false"/>
</Type>
```

[0196] The NestedElement type inherits from this type. The NestedElement element type additionally defines a field that is a multi-set of Elements.

```
<Type Name="NestedElement" BaseType="Base.Element"
        IsAbstract="True">
        <Property Name="Extensions"
                  Type="Base.Element"
                  Nullable="false"
                  MultiValued="true"/>
</Type>
```

[0197] The NestedElement extensions are different from item extensions in the following ways:

- Nested element extensions are not extension types. They do not belong to the extension type hierarchy that is rooted in the Base.Extension type.
- Nested element extensions are stored along with the other fields of the item and are not globally accessible – a query can not be composed that retrieves all instances of a given extension type.
- These extensions are stored the same way as other nested elements (of the item) are stored. Like other nested sets, the NestedElement extensions are stored in a UDT. They are accessible through the Extensions field of the nested element type.

- The collection interfaces used to access multi-valued properties is also used for accessing and iterating over set of type extensions.

[0198] The following table summarizes and compares Item Extensions and NestedElement extensions.

Item extensions vs NestedElement extensions

	Item Extension	NestedElement Extension
Storage	Item extension hierarchy is stored in its own tables	Stored like nested elements
Query/Search	Can query item extension tables	Can generally only be queried within the containing item context
Query/Search scope	Can search across all instances of an item extension type	Can generally only search within nested element type instances of a singe (containing) item
Programmability	Need special extension APIs and special querying on extension tables	NestedElement extensions are like any other multi-valued field of nested element; normal nested element type APIs are used
Behavior	Can associate behavior	No behavior permitted (?)
Relationship semantics	No Relationships to item extensions	No Relationships to NestedElement extensions
Item ID	Shares the item id of the item	Does not have its own item id. NestedElement extension is part of the item

F. DATABASE ENGINE

[0199] As mentioned above, the data store is implemented on a database engine. In the present embodiment, the database engine comprises a relational database engine that implements the SQL query language, such as the Microsoft SQL Server engine, with object relational extensions. This section describes the mapping of the data model that the data store implements to the relational store and provides information on the logical API consumed by storage platform clients, in accordance with the present embodiment. It is understood, however, that a different mapping may be employed when a different database engine is employed. Indeed, in addition to implementing the storage platform conceptual data model on a relational database engine, it can also be implemented on other types of databases, *e.g.* object-oriented and XML databases.

[0200] An object-oriented (OO) database system provides persistence and transactions for programming language objects (*e.g.* C++, Java). The storage platform notion of an "item" maps well to an "Object" in object-oriented systems, though embedded collections would have to be added to Objects. Other storage platform type concepts, like inheritance and nested element types, also map object-oriented type systems. Object-oriented systems typically already support object identity; hence, item identity can be mapped to object identity. The item behaviors (operations) map well to object methods. However, object-oriented systems typically lack organizational capabilities and are poor in searching. Also, object-oriented systems to do not provide support for unstructured and semi-structured data. To support the complete storage platform data model described herein, concepts like relationships, folders, and extensions would need to be added to the object data model. In addition, mechanisms like promotions, synchronization, notifications, and security would need to be implemented.

[0201] Similar to object-oriented systems, XML databases, based on XSD (XML Schema Definition), support a single-inheritance based type system. The item type system of the present invention could be mapped to the XSD type model. XSDs also do not provide support for behaviors. The XSDs for items would have to be augmented with item behaviors. XML databases deal with single XSD documents and lack organization and broad search capabilities. As with object-oriented databases, to support the data model described herein, other concepts like relationships, and folders would need to be incorporated into such XML databases; also, mechanisms like synchronization, notifications and security would need to be implemented.

[0202] In regard to the following subsections, a few illustrations are provided to facilitate the general information disclosed: Fig. 13 is a diagram illustrating a notification mechanism. Fig. 14 is a diagram illustrating an example in which two transactions are both inserting a new record into the same B-Tree. Fig. 15 illustrates a data change detection process. Fig. 16 illustrates an exemplary directory tree. Fig. 17 shows an example in which an existing folder of a directory-based file system is moved into the storage platform data store.

1. Data Store Implementation Using UDTs

[0203] In the present embodiment, the relational database engine 314, which in one embodiment comprises the Microsoft SQL Server engine, supports built-in scalar types. Built-in scalar types are "native" and "simple". They are native in the sense that the user cannot define their own types and they are simple in that they cannot encapsulate a complex structure. User-defined types (hereinafter: UDTs) provide a mechanism for type extensibility above and beyond the native scalar type system by enabling users to extend the type system by defining complex, structured types. Once defined by a user, a UDT can be used anywhere in the type system that a built-in scalar type might be used

[0204] In accordance with an aspect of the present invention, the storage platform schemas are mapped to UDT classes in the database engine store. Data store Items are mapped to UDT classes deriving from the Base.Item type. Like Items, Extensions are also mapped to UDT classes and make use of inheritance. The root Extension type is Base.Extension, from which all Extension types are derived.

[0205] A UDT is a CLR class – it has state (*i.e.*, data fields) and behavior (*i.e.*, routines). UDTs are defined using any of the managed languages – C#, VB.NET, etc. UDT methods and operators can be invoked in T-SQL against an instance of that type. A UDT can be: the type of a column in a row, the type of a parameter of a routine in T-SQL, or the type of a variable in T-SQL

[0206] The mapping of storage platform schemas to UDT classes is fairly straightforward at a high level. Generally, a storage platform Schema is mapped to a CLR namespace. A storage platform Type is mapped to a CLR class. The CLR class inheritance mirrors the storage platform Type inheritance, and a storage platform Property is mapped to a CLR class property.

2. Item Mapping

[0207] Given the desirability for Items to be globally searchable, and the support in the relational database of the present embodiment for inheritance and type substitutability, one possible implementation for Item storage in the database store would be to store all Items in a single table with a column of type Base.Item. Using type substitutability, Items of all types could be stored, and searches could be filtered by Item type and sub-type using Yukon's "is of (*Type*)" operator.

[0208] However, due to concerns about the overhead associated with such an approach, in the present embodiment, the Items are divided by top-level type, such that Items of each type "family" are stored in a separate table. Under this partitioning scheme, a table is created for each Item type inheriting directly from Base.Item. Types inheriting below these are stored in the appropriate type family table using type substitutability, as described above. Only the first level of inheritance from Base.Item is treated specially.

[0209] A "shadow" table is used to store copies of globally searchable properties for all Items. This table may be maintained by the Update() method of the storage platform API, through which all data changes are made. Unlike the type family tables, this global Item table contains only the top-level scalar properties of the Item, not the full UDT Item object. The global Item table allows navigation to the Item object stored in a type family table by exposing an ItemID and a TypeID. The ItemID will generally uniquely identify the Item within the data store. The TypeID may be mapped using metadata, which is not described here, to a type name and the view containing the Item. Since finding an Item by its ItemID may be a common operation, both in the context of the global Item table and otherwise, a GetItem() function is provided to retrieve an Item object given an Item's ItemID.

[0210] For convenient access and to hide implementation details to the extent possible, all queries of Items might be against views built on the Item tables described above. Specifically, views may be created for each Item type against the appropriate type family table. These type views may select all Items of the associated type, including sub-types. For convenience, in addition to the UDT object, the views may expose columns for all of the top-level fields of that type, including inherited fields.

3. Extension Mapping

[0211] Extensions are very similar to Items and have some of the same requirements. As another root type supporting inheritance, Extensions are subject to many of the same considerations and trade-offs in storage. Because of this, a similar type family mapping is applied to Extensions, rather than a single table approach. Of course, in other embodiments, a single table approach could be used. In the present embodiment, an Extension is associated with exactly one Item by ItemID, and contains an ExtensionID that is unique in the context of the Item. As with Items, a function might be provided to retrieve an Extension given its identity, which consists of an ItemID and ExtensionID pair. A View is created for each Extension type, similar to the Item type views.

4. Nested Element Mapping

[0212] Nested Elements are types that can be embedded in Items, Extensions, Relationships, or other Nested Elements to form deeply nested structures. Like Items and Extensions, Nested Elements are implemented as UDT's, but they are stored within an Items and Extensions. Therefore, Nested Elements have no storage mapping beyond that of their Item and Extension containers. In other words, there are no tables in the system which directly store instances of NestedElement types, and there are no views dedicated specifically to Nested Elements.

5. Object Identity

[0213] Each entity in the data model, *i.e.,* each Item, Extension and Relationship, has a unique key value. An Item is uniquely identified by its ItemId. An Extension is uniquely identified by a composite key of (ItemId, ExtensionId). A Relationship is identified by a composite key (ItemId, RelationshipId). ItemId, ExtensionId and RelationshipId are GUID values.

6. SQL Object Naming

[0214] All objects created in the data store can be stored in a SQL schema name derived from the storage platform schema name. For example, the storage platform Base schema (often called "Base") may produce types in the "[System.Storage]" SQL schema such as

"[System.Storage].Item". Generated names are prefixed by a qualifier to eliminate naming conflicts. Where appropriate, an exclamation character (!) is used as a separator for each logical part of the name. The table below outlines the naming convention used for objects in the data store. Each schema element (Item, Extension, Relationship and View), is listed along with the decorated naming convention used to access instances in the data store.

Object	Name Decoration	Description	Example
Master Item Search View	**Master!Item**	Provides a summary of items in the current item domain.	[System.Storage]. **[Master!Item]**
Typed Item search view	*ItemType*	Provides all property data from *item* and any parent type(s).	[AcmeCorp.Doc]. [OfficeDoc]
Master Extension Search View	**Master!Extension**	Provides a summary of all extensions in the current item domain.	[System.Storage]. **[Master!Extension]**
Typed extension search view	**Extension!***extensionType*	Provides all property data for *extension*.	[AcmeCorp.Doc]. **[Extension!**StickyNote]
Master Relationship View	**Master!Relationship**	Provides a summary of all relationships in the current item domain.	[System.Storage]. **[Master!Relationship]**
Relationship view	**Relationship!***relationship Name*	Provides all data associated with a given *relationship*	[AcmeCorp.Doc]. **[Relationship!**AuthorsFrom Document]
View	**View!***viewName*	Provides the columns/types based on the schema *view* definition.	[AcmeCorp.Doc]. **[View!**DocumentTitles]

7. Column Naming

[0215] When mapping any object model into a store, the possibility of naming collisions occur due to additional information stored along with an application object. In order

to avoid naming collisions, all non-type specific columns (columns which do not map directly to a named Property in a type declaration) is be prefixed with an underscore (_) character. In the present embodiment, underscore (_) characters are disallowed as the beginning character of any identifier property. Further, in order to unify naming between CLR and the data store, all properties of a storage platform types or schema element (relationship, etc.) should have a capitalized first character.

8. Search Views

[0216] Views are provided by the storage platform for searching stored content. A SQL view is provided for each Item and Extension type. Further, views are provided to support Relationships and Views (as defined by the Data Model). All SQL views and underlying tables in the storage platform are read-only. Data may be stored or changed using the Update() method of the storage platform API, as described more fully below.

[0217] Each view explicitly defined in a storage platform schema (defined by the schema designer, and not automatically generated by the storage platform) is accessible by the named SQL view [<schema-name>].[View!<view-name>]. For example, a view named "BookSales" in the schema "AcmePublisher.Books" would be accessible using the name "[AcmePublisher.Books].[View!BookSales]". Since the output format of a view is custom on a per-view basis (defined by an arbitrary query provided by the party defining the view), the columns are directly mapped based on the schema view definition.

[0218] All SQL search views in the storage platform data store use the following ordering convention for columns:

- Logical "key" column (s) of view result such as *ItemId, ElementId, RelationshipId, ...*
- Metadata information on type of result such as *TypeId.*
- Change tracking columns such as *CreateVersion, UpdateVersion, ...*
- Type specific column(s) (Properties of the declared type)
- Type specific views (family views) also contain an object column which returns the object

[0219] Members of each type family are searchable using a series of Item views, with there being one view per Item type in the data store. Fig. 28 is a diagram illustrating the concept of an Item search view.

a) Item

[0220] Each Item search view contains a row for each instance of an Item of the specific type or its subtypes. For example, the view for Document could return instances of Document, LegalDocument and ReviewDocument. Given this example, the Item views can be conceptualized as shown in Fig. 29.

(1) Master Item Search View

[0221] Each instance of a storage platform data store defines a special Item view called the *Master Item View*. This view provides summary information on each Item in the data store. The view provides one column per Item type property, a column which described the type of the Item and several columns which are used to provide change tracking and synchronization information. The master item view is identified in a data store using the name "[System.Storage].[Master!Item]".

Column	Type	Description
ItemId	ItemId	The storage platform identity of the Item
_TypeId	TypeId	The TypeId of the Item - identifies the exact type of the Item and can be used to retrieve information on the type using a Metadata catalog.
_RootItemId	ItemId	The ItemId of the first non-embedded ancestor that controls the lifetime of this item.
<global change tracking>	...	Global change tracking information
<Item props>	n/a	One column per Item type property

(2) Typed Item Search Views

[0222] Each Item type also has a search view. While similar to the root Item view, this view also provides access to the Item object via the "_Item" column. Each typed item search

view is identified in a data store using the name [*schemaName*].[*itemTypeName*]. For example [AcmeCorp.Doc].[OfficeDoc].

Column	Type	Description
ItemId	ItemId	The storage platform identity of the Item
<type change tracking>	...	Type change tracking information
<parent props>	<property specific>	One column per parent property
<item props>	<property specific>	One column per exclusive property of this type
_Item	CLR type of Item	CLR object – type of declared Item

b) Item Extensions

[0223] All Item Extensions in a WinFS Store are also accessible using search views.

(1) Master Extension Search View

[0224] Each instance of a data store defines a special Extension view called the *Master Extension View*. This view provides summary information on each Extension in the data store. The view has a column per Extension property, a column which describes the type of the Extension and several columns which are used to provide change tracking and synchronization information. The master extension view is identified in a data store using the name "[System.Storage].[Master!Extension]".

Column	Type	Description
ItemId	ItemId	The storage platform identity of the Item with which this extension is associated
ExtensionId	ExtensionId (GUID)	Id of this extension instance
_TypeId	TypeId	The TypeId of the Extension - identifies the exact type of the extension and can be used to retrieve information on the extension using the Metadata catalog.

<global change tracking>	...	Global change tracking information
<ext properties>	<property specific>	One column per Extension type property

(2) Typed Extension Search Views

[0225] Each Extension type also has a search view. While similar to the master extension view, this view also provides access to the Item object via the _Extension column. Each typed extension search view is identified in a data store using the name [*schemaName*].[Extension!*extensionTypeName*]. For example [AcmeCorp.Doc].[Extension!OfficeDocExt].

Column	Type	Description
ItemId	ItemId	The storage platform identity of the Item with which this extension is associated
ExtensionId	ExtensionId (GUID)	Id of this extension instance
<type change tracking>	...	Type change tracking information
<parent props>	<property specific>	One column per parent property
<ext props>	<property specific>	One column per exclusive property of this type
_Extension	CLR type of Extension instance	CLR object – type of declared Extension

c) Nested Elements

[0226] All nested elements are stored within Items, Extensions or Relationships instances. As such, they are accessed by querying the appropriate Item, Extension, or Relationship search view.

d) Relationships

[0227] As discussed above, Relationships form the fundamental unit of linking between Items in a storage platform data store.

(1) Master Relationship Search View

[0228] Each data store provides a *Master Relationship View.* This view provides information on all relationship instances in the data store. The master relationship view is identified in a data store using the name "[System.Storage].[Master!Relationship]".

Column	Type	Description
ItemId	ItemId	Identity of source endpoint (ItemId)
RelationshipId	RelationshipId (GUID)	The id of the relationship instance
_RelTypeId	RelationshipTypeId	The RelTypeId of the Relationship - identifies the type of the relationship instance using the Metadata catalog.
<global change tracking>	...	Global change tracking information.
TargetItemReference	ItemReference	Identity of target endpoint
_Relationship	Relationship	Instance of the Relationship object for this instance

(2) Relationship Instance Search Views

[0229] Each declared Relationship also has a search view which returns all instances of the particular relationship. While similar to the master relationship view, this view also provides named columns for each property of the relationship data. Each relationship instance search view is identified in a data store using the name [*schemaName*].[**Relationship!***relationshipName*]. For example [AcmeCorp.Doc].[Relationship!DocumentAuthor].

Column	Type	Description
ItemId	ItemId	Identify of source endpoint (ItemId)

RelationshipId	RelationshipId (GUID)	The id of the relationship instance
<type change tracking>	...	Type change tracking information
TargetItemReference	ItemReference	Identity of target endpoint
<source name>	ItemId	Named property of source endpoint identity (alias for ItemId)
<target name>	ItemReference or derived class	Named property of target endpoint identity (alias and cast for TargetItemReference)
<rel property>	<property specific>	One column per property of the relationship definition
_Relationship	CLR type of Relationship instance	CLR object – type of declare Relationship

e)

9. Updates

[0230] All views in the storage platform data store are read-only. In order to create a new instance of a data model element (item, extension or relationship), or to update an existing instance, the ProcessOperation or ProcessUpdategram methods of the storage platform API must be used. The ProcessOperation method is a single stored procedure defined by the data store which consumes an "operation" that details an action to be performed. The ProcessUpdategram method is a stored procedure which takes an ordered set of operations, known as an "updategram", which collectively detail a set of actions to be performed..

[0231] The operation format is extensible and provides various operations over the schema elements. Some common operations include:

1. Item operations:
    a. CreateItem (Creates a new item in the context of an embedding or holding relationship)
    b. UpdateItem (updates an existing Item)
2. Relationship operations:
    a. CreateRelationship (creates an instance of a reference or holding relationship)
    b. UpdateRelationship (updates a relationship instance)

c. DeleteRelationship (removes a relationship instances)

3. Extension operations:

a. CreateExtension (adds an extension to an existing Item)

b. UpdateExtension (updates an existing extension)

c. DeleteExtension (deletes an extension)

10. Change Tracking & Tombstones

[0232] Change tracking and tombstone services are provided by the data store, as discussed more fully below. This section provides an outline of the change tracking information exposed in a data store.

a) Change Tracking

[0233] Each search view provided by the data store contains columns used to provide change tracking information; the columns are common across all Item, Extension and Relationship views. Storage platform Schema Views, defined explicitly by schema designers, do not automatically provide change tracking information – such information is provided indirectly through the search views on which the view itself is built.

[0234] For each element in the data store, change tracking information is available from two places – the "master" element view and the "typed" element view. For example, change tracking information on the AcmeCorp.Document.Document Item type is available from the Master Item View "[System.Storage].[MasterItem]" and typed Item search view [AcmeCorp.Document].[Document].

(1) Change Tracking in "Master" Search Views

[0235] Change tracking information in the master search views provides information on the creation and update versions of an element, information on which sync partner created the element, which sync partner last updated the element and the version numbers from each partner for creation and update. Partners in sync relationships (described below) are identified by *partner key*. A single UDT object named _ChangeTrackingInfo of type [System.Storage.Store].ChangeTrackingInfo contains all this information. The type is defined in

the System.Storage schema. _ChangeTrackingInfo is available in all global search views for Item, Extension and Relationship. The type definition of ChangeTrackingInfo is:

```
<Type Name="ChangeTrackingInfo" BaseType="Base.NestedElement">
  <FieldProperty Name="CreationLocalTS"       Type="SqlTypes.SqlInt64"
  Nullable="False" />
  <FieldProperty Name="CreatingPartnerKey"
  Type="SqlTypes.SqlInt32"        Nullable="False" />
  <FieldProperty Name="CreatingPartnerTS"
  Type="SqlTypes.SqlInt64"        Nullable="False" />
  <FieldProperty Name="LastUpdateLocalTS"
  Type="SqlTypes.SqlInt64"        Nullable="False" />
  <FieldProperty Name="LastUpdatingPartnerKey"
  Type="SqlTypes.SqlInt32"        Nullable="False" />
  <FieldProperty Name="LastUpdatingPartnerTS" Type="SqlTypes.SqlInt64"
  Nullable="False" />
</Type>
```

These properties contain the following information:

Column	Description
_CreationLocalTS	Creation time stamp by the local machine
_CreatingPartnerKey	PartnerKey of the partner who created this entity. If the entity was locally created, this is the local machine's PartnerKey.
_CreatingPartnerTS	Timestamp of the time at which this entity was created at the partner corresponding to _CreatingPartnerKey.
_LastUpdateLocalTS	Local timestamp corresponding to the update time at the local machine
_LastUpdatingPartnerKey	PartnerKey of the partner who last updated this entity. If the last update to the entity was done locally, this is the local machine's PartnerKey.
_LastUpdatingPartnerTS	Timestamp of the time at which this entity was updated at the partner corresponding to _LastUpdatingPartnerKey.

(2) Change Tracking in "Typed" Search Views

[0236] In addition to providing the same information as the global search view, each typed search view provides additional information recording the sync state of each element in the sync topology.

Column	Type	Description
<global change tracking>	...	Information from global change tracking
_ChangeUnitVersions	MultiSet<ChangeUnitVersion>	Description of version numbers of the change units within the particular element
_ElementSyncMetadata	ElementSyncMetadata	Additional version-independent metadata about this item that is only of interest to the Synchronization runtime.
_VersionSyncMetadata	VersionSyncMetadata	Additional version-specific metadata about this version that is only of interest to the Synchronization runtime

b) Tombstones

[0237] The data store provides tombstone information for Items, Extensions and Relationships. The tombstone views provide information about both live and tombstoned entities (items, extensions and relationships) in one place. The item and extension tombstone views do not provide access to the corresponding object, while the relationship tombstone view provides access to the relationship object (the relationship object is NULL in the case of a tombstoned relationship).

(1) Item Tombstones

[0238] Item tombstones are retrieved from the system via the view [System.Storage].[Tombstone!Item].

Column	Type	Description
ItemId	ItemId	Identity of the Item
_TypeID	TypeId	Type of the Item
<Item properties>	...	Properties defined for all items
_RootItemId	ItemId	ItemId of the first non-embedding item which contains this item.
_ChangeTrackingInfo	CLR instance of	Change tracking information for this item

	type ChangeTrackingInfo	
_IsDeleted	BIT	This is a flag that is 0 for live items, and 1 for tombstoned items.
_DeletionWallclock	UTCDATETIME	The UTC wall clock date time according to the partner which deleted the item. It is NULL if the Item is live.

(2) Extension Tombstones

[0239] Extension tombstones are retrieved from the system using the view [System.Storage].[Tombstone!Extension]. Extension change tracking information is similar to that provided for Items with the addition of the ExtensionId property.

Column	Type	Description
ItemId	ItemId	Identity of the Item which owns the Extension
ExtensionId	ExtensionId	Extension Id of the Extension
_TypeID	TypeId	Type of the extension
_ChangeTrackingInfo	CLR instance of type ChangeTrackingInfo	Change tracking information for this extension
_IsDeleted	BIT	This is a flag that is 0 for live items, and 1 for tombstoned extensions.
_DeletionWallclock	UTCDATETIME	The UTC wall clock date time according to the partner which deleted the extension. It is NULL if the extension is live.

(3) Relationships Tombstone

[0240] Relationship tombstones are retrieved from the system via the view [System.Storage].[Tombstone!Relationship]. Relationships tombstone information is similar to that provided for Extensions. However, additional information is provided on the target ItemRef of the relationship instance. In addition, the relationship object is also selected.

Column	Type	Description

ItemId	ItemId	Identity of the Item which owned the relationship (identity of relationship source endpoint)
RelationshipId	RelationshipId	RelationshipId of the relationship
_TypeID	TypeId	Type of the relationship
_ChangeTrackingInfo	CLR instance of type ChangeTrackingInfo	Change tracking information for this relationship
_IsDeleted	BIT	This is a flag that is 0 for live items, and 1 for tombstoned extensions.
_DeletionWallclock	UTCDATETIME	The UTC wall clock date time according to the partner which deleted the relationship. It is NULL if the relationship is live.
_Relationship	CLR instance of a Relationship	This is the relationship object for live relationship. It is NULL for tombstoned relationships.
TargetItemReference	ItemReference	Identity of target endpoint

(4) Tombstone Cleanup

[0241] In order to prevent unbounded growth of tombstone information, the data store provides a tombstone cleanup task. This task determines when tombstone information may be discarded. The task computes a bound on the local create / update version and then truncates the tombstone information by discarding all earlier tombstone versions.

11. Helper APIs and Functions

[0242] The Base mapping also provides a number of helper functions. These functions are supplied to aid common operations over the data model.

a) Function [System.Storage].GetItem

```
Returns an Item object given an ItemId
//
Item GetItem (ItemId ItemId)
```

b) Function [System.Storage].GetExtension

```
// Returns an extension object given an ItemId and ExtensionId
```

```
//
Extension GetExtension (ItemId itemId, ExtensionId ExtensionId)
```

c) Function [System.Storage].GetRelationship

```
// Returns an relationship object given an itemId and RelationshipId
//
Relationship GetRelationship (ItemId itemId, RelationshipId RelationshipId)
```

12. Metadata

[0243] There are two types of metadata represented in the Store: instance metadata (the type of an Item, etc), and type metadata.

a) Schema Metadata

[0244] Schema metadata is stored in the data store as instances of Item types from the Meta schema.

b) Instance Metadata

[0245] Instance metadata is used by an application to query for the type of an Item and finds the extensions associated with an Item. Given the ItemId for an Item, an application can query the global item view to return the type of the Item and use this value to query the Meta.Type view to return information on the declared type of the Item. For example,

```
// Return metadata Item object for given Item instance
//
SELECT m._Item AS metadataInfoObj
FROM [System.Storage].[Item] I INNER JOIN [Meta].[Type] m ON I._TypeId = m.ItemId
WHERE i.ItemId = @ItemId
```

G. SECURITY

[0246] In general, all securable objects arrange their access rights using the access mask format shown in the Fig. 26. In this format, the low-order 16 bits are for object-specific access rights, the next 7 bits are for standard access rights, which apply to most types of objects, and the 4 high-order bits are used to specify generic access rights that each object type can map

to a set of standard and object-specific rights. The ACCESS_SYSTEM_SECURITY bit corresponds to the right to access the object's SACL.

[0247] In the access mask structure of Fig. 26, item specific rights are placed in the Object Specific Rights section (low order 16-bits). Because in the present embodiment, the storage platform exposes two sets of APIs to administer security – Win32 and the storage platform API, the file system object specific rights must be considered in order to motivate the design of the storage platform object specific rights.

[0248] The security model for the storage platform of the present invention is fully described in the related applications incorporated by reference earlier herein. In this regard, Fig. 27 (parts a, b, and c) depicts a new identically protected security region being carved out of an existing security region, in accordance with one embodiment of a security model.

H. NOTIFICATIONS AND CHANGE TRACKING

[0249] According to another aspect of the present invention, the storage platform provides a notifications capability that allows applications to track data changes. This feature is primarily intended for applications which maintain volatile state or execute business logic on data change events. Applications register for notifications on items, item extensions and item relationships. Notifications are delivered asynchronously after data changes have been committed. Applications may filter notifications by item, extension and relationship type as well as type of operation.

[0250] According to one embodiment, the storage platform API 322 provides two kinds of interfaces for notifications. First, applications register for simple data change events triggered by changes to items, item extensions and item relationships. Second, applications create "watcher" objects to monitor sets of items, item extensions and relationships between items. The state of a watcher object can be saved and re-created after a system failure or after a system has gone off-line for an extended period of time. A single notification may reflect multiple updates.

[0251] Additional details regarding this functionality can be found in the related applications incorporated by reference earlier herein.

I. SYNCHRONIZATION

[0252] According to another aspect of the present invention, the storage platform provides a synchronization service 330 that (i) allows multiple instances of the storage platform (each with its own data store 302) to synchronize parts of their content according to a flexible set of rules, and (ii) provides an infrastructure for third parties to synchronize the data store of the storage platform of the present invention with with other data sources that implement proprietary protocols.

[0253] Storage platform-to-storage platform synchronization occurs among a group of participating replicas. For example, with reference to Fig. 3, it may be desirable to provide synchronization between the data store 302 of the storage platform 300 with another remote data store 338 under the control of another instance of the storage platform, perhaps running on a different computer system. The total membership of this group is not necessarily known to any given replica at any given time.

[0254] Different replicas can make the changes independently (*i.e.* concurrently). The process of synchronization is defined as making every replica aware of the changes made by other replicas. This synchronization capability is inherently multi-master.

[0255] The synchronization capability of the present invention allows replicas to:

- determine which changes another replica is aware of;
- request information about changes that this replica is not aware of;
- convey information about changes that the other replica is not aware of;
- determine when two changes are in conflict with each other;
- apply changes locally;
- convey conflict resolutions to other replicas to ensure convergence; and
- resolve the conflicts based on specified policies for conflict resolutions.

1. Storage Platform-to-Storage Platform Synchronization

[0256] The primary application of the synchronization service 330 of the storage platform of the present invention is to synchronize multiple instances of the storage platform (each with its own data store). The synchronization service operates at the level of the storage

platform schemas (rather than the underlying tables of the database engine 314). Thus, for example, "Scopes" are used to define synchronization sets as discussed below.

[0257] The synchronization service operates on the principle of "net changes". Rather than recording and sending individual operations (such as with transactional replication), the synchronization service sends the end-result of those operations, thus often consolidating the results of multiple operations into a single resulting change.

[0258] The synchronization service does not in general respect transaction boundaries. In other words, if two changes are made to a storage platform data store in a single transaction, there is no guarantee that these changes are applied at all other replicas atomically — one may show up without the other. The exception to this principle is that if two changes are made to the same Item in the same transaction, then these changes are guaranteed to be sent and applied to other replicas atomically. Thus, Items are the consistency units of the synchronization service.

a) Synchronization (Sync) Controlling Applications

[0259] Any application can connect to the synchronization service and initiate a sync operation. Such an application provides all of the parameters needed to perform synchronization (*see* sync profile below). Such applications are referred to herein as Sync Controlling Applications (SCAs).

[0260] When synchronizing two storage platform instances, sync is initiated on one side by an SCA. That SCA informs the local synchronization service to synchronize with the remote partner. On the other side, the synchronization service is awoken by the messages sent by the synchronization service from the originating machine. It responds based on the persistent configuration information (*see* mappings below) present on the destination machine. The synchronization service can be run on schedule or in response to events. In these cases, the synchronization service implementing the schedule becomes the SCA.

[0261] To enable synchronization, two steps need to be taken. First, the schema designer must annotate the storage platform schema with appropriate sync semantics (designating Change Units as described below). Second, synchronization must be properly configured on all of the machines having an instance of the storage platform that is to participate in the synchronization (as described below).

b) Schema annotation

[0262] A fundamental concept of the synchronization service is that of a Change Unit. A Change Unit is a smallest piece of schema that is individually tracked by the storage platform. For every Change Unit, the synchronization service may be able to determine whether it changed or did not change since the last sync.

[0263] Designating Change Units in the schema serves several purposes. First, it determines how chatty the synchronization service is on the wire. When a change is made inside a Change Unit, the entire Change Unit is sent to the other replicas, since the synchronization service does not know which part of the Change Unit was changed. Second, it determines the granularity of conflict detection. When two concurrent changes (these terms are defined in detail in subsequent sections) are made to the same change unit, the synchronization service raises a conflict; on the other hand, if concurrent changes are made to different change units, then no conflict is raised and the changes are automatically merged. Third, it strongly affects the amount of meta-data kept by the system. Much of the synchronization service meta-data is kept per-Change Unit; thus, making Change Units smaller increases the overhead of sync.

[0264] Defining Change Units requires finding the right trade-offs. For that reason, the synchronization service allows schema designers to participate in the process.

[0265] In one embodiment, the synchronization service does not support Change Units that are larger than an element. However, it does support the ability for schema designers to specify smaller change units than an element -- namely, grouping multiple attributes of an element into a separate Change Unit. In that embodiment, this is accomplished using the following syntax:

```
<Type Name="Appointment" MajorVersion="1" MinorVersion="0" ExtendsType="Base.Item"
        ExtendsVersion="1">

   <Field Name="MeetingStatus" Type="the storage platformTypes.uniqueIdentifier Nullable="False"/>
   <Field Name="OrganizerName" Type="the storage platformTypes.nvarchar(512)" Nullable="False"/>
   <Field Name="OrganizerEmail" Type="the storage platformTypes.nvarchar(512)"
      TypeMajorVersion="1"                    MultiValued="True"/>
    ...
   <ChangeUnit Name="CU_Status">
       <Field Name="MeetingStatus"/>
   </ChangeUnit>

   <ChangeUnit Name="CU_Organizer"/>
       <Field Name="OrganizerName" />
```

```
    <Field Name="OrganizerEmail" />
  </ChangeUnit>
  ...
</Type>
```

c) Sync Configuration

[0266] A group of storage platform partners that wish to keep certain parts of their data in sync are referred to as a sync community. While the members of the community want to stay in sync, they do not necessarily represent the data in exactly the same way; in other words, sync partners may transform the data they are synchronizing.

[0267] In a peer-to-peer scenario, it is impractical for peers to maintain transformation mappings for all of their partners. Instead, the synchronization service takes the approach of defining "Community Folders". A community folder is an abstraction that represents a hypothetical "shared folder" that all community members are synchronizing with.

[0268] This notion is best illustrated by an example. If Joe wants to keep My Documents folders of his several computers in sync, Joe defines a community folder called, say, JoesDocuments. Then, on every computer, Joe configures a mapping between the hypothetical JoesDocuments folder and the local My Documents folder. From this point on, when Joe's computers synchronize with each other, they talk in terms of documents in JoesDocuments, rather than their local items. This way, all Joe's computers understand each other without having to know who the others are — the Community Folder becomes the lingua franca of the sync community.

[0269] Configuring the synchronization service consists of three steps: (1) defining mappings between local folders and community folders; (2) defining sync profiles that determine what gets synchronized (e.g. whom to sync with and which subsets should be sent and which received); and (3) defining the schedules on which different sync profiles should run, or running them manually.

(1) Community Folder - Mappings

[0270] Community Folder mappings are stored as XML configuration files on individual machines. Each mapping has the following schema:

/mappings/communityFolder

This element names the community folder that this mapping is for. The name follows the syntax rules of Folders.

/mappings/localFolder

This element names the local folder that the mapping transforms into. The name follows the syntax rules of Folders. The folder must already exist for the mapping to be valid. The items within this folder are considered for synchronization per this mapping.

/mappings/transformations

This element defines how to transform items from the community folder to the local folder and back. If absent or empty, no transformations are performed. In particular, this means that no IDs are mapped. This configuration is primarily useful for creating a cache of a Folder.

/mappings/transformations/mapIDs

This element requests that newly generated local IDs be assigned to all of the items mapped from the community folder, rather than reusing community IDs. The Sync Runtime will maintain ID mappings to convert items back and forth.

/mappings/transformations/localRoot

This element requests that all root items in the community folder be made children of the specified root.

/mappings/runAs

This element controls under whose authority requests against this mapping are processed. If absent, *sender* is assumed.

/mappings/runAs/sender

The presence of this element indicates that the sender of messages to this mapping must be impersonated, and requests processed under his credentials.

(2) Profiles

[0271] A Sync Profile is a total set of parameters needed to kick off synchronization. It is supplied by an SCA to the Sync Runtime to initiate sync. Sync profiles for storage platform-to-storage platform synchronization contain the following information:

- Local Folder, to serve as the source and destination for changes;
- Remote Folder name to synchronize with - this Folder must be published from the remote partner by way of a mapping as defined above;
- Direction - the synchronization service supports send-only, receive-only, and send-receive sync;
- Local Filter -- selects what local information to send to the remote partner. Expressed as a the storage platform query over the local folder;
- Remote Filter – selects what remote information to retrieve from the remote partner - expressed as a storage platform query over the community folder;

- Transformations --- defines how to transform items to and from the local format;
- Local security --- specifies whether the changes retrieved from the remote endpoint are to be applied under the permissions of the remote endpoint (impersonated) or the user initiating the sync locally; and
- Conflict resolution policy — specifies whether conflicts should be rejected, logged, or automatically resolved - in the latter case, it specifies which conflict resolver to use, as well as the configuration parameters for it.

[0272] The synchronization service provides a runtime CLR class that allows simple building of Sync Profiles. Profiles can also be serialized to and from XML files for easy storage (often alongside schedules). However, there is no standard place in the storage platform where all the profiles are stored; SCAs are welcome to construct a profile on the spot without ever persisting it. Note that there is no need to have a local mapping to initiate sync. All sync information can be specified in the profile. The mapping is, however, required in order to respond to sync requests initiated by the remote side.

(3) Schedules

[0273] In one embodiment, the synchronization service does not provide its own scheduling infrastructure. Instead, it relies on another component to peform this task - the Windows Scheduler available with the Microsoft Windows operating system. The synchronization service includes a command-line utility that acts as an SCA and triggers synchronization based on a sync profile saved in an XML file. This utility makes it very easy to configure the Windows Scheduler to run synchronization either on schedule, or in response to events such as user logon or logoff.

d) Conflict Handling

[0274] Conflict handling in the synchronization service is divided into three stages: (1) conflict detection, which occurs at change application time - this step determines if a change can be safely applied; (2) automatic conflict resolution and logging - during this step (that takes place immediately after the conflict is detected) automatic conflict resolvers are consulted to see if the conflict can be resolved - if not, the conflict can be optionally logged; and (3) conflict inspection and resolution - this step takes place if some conflicts have been logged, and occurs outside of

(2) Conflict Processing

[0281] When a conflict is detected, the synchronization service can take one of three actions (selected by the sync initiator in the Sync Profile): (1) reject the change, returning it back to sender; (2) log a conflict into a conflict log; or (3) resolve the conflict automatically.

[0282] If the change is rejected, the synchronization service acts as if the change did not arrive at the replica. A negative acknowledgement is sent back to the originator. This resolution policy is primarily useful on head-less replicas (such as file servers) where logging conflicts is not feasible. Instead, such replicas force the others to deal with the conflicts by rejecting them.

[0283] Sync initiators configure conflict resolution in their Sync Profiles. The synchronization service supports combining multiple conflict resolvers in a single profile in the following ways – first, by specifying a list of conflict resolvers to be tried one after another, until one of them succeeds; and second, by associating conflict resolvers with conflict types, e.g. directing update-update knowledge-based conflicts to one resolver, but all the other conflicts to the log.

(a) Automatic Conflict resolution

[0284] The synchronization service provides a number of default conflict resolvers. This list includes:

- local-wins: disregard incoming changes if in conflict with locally stored data;
- remote-wins: disregard local data if in conflict with incoming changes;
- last-writer-wins: pick either local-wins or remote-wins per change unit based on the timestamp of the change (note that the synchronization service in general does not rely on clock values; this conflict resolver is the sole exception to that rule);
- Deterministic: pick a winner in a manner that is guaranteed to be the same on all replicas, but not otherwise meaningful – one embodiment of the synchronization services uses lexicographic comparisons of partner IDs to implement this feature.

[0285] In addition, ISVs can implement and install their own conflict resolvers. Custom conflict resolvers may accept configuration parameters; such parameters must be specified by the SCA in the Conflict Resolution section of the Sync Profile.

the context of the sync session - at this time, logged conflicts can be resolved and removed from the log.

(1) Conflict Detection

[0275] In the present embodiment, the synchronization service detects two types of conflicts: knowledge-based and constraint-based.

(a) Knowledge-based conflicts

[0276] A knowledge-based conflict occurs when two replicas make independent changes to the same Change Unit. Two changes are called independent if they are made without knowledge of each other — in other words, the version of the first is not covered by the knowledge of the second and vice versa. The synchronization service automatically detects all such conflicts based on the replicas' knowledge as described above.

[0277] It is sometimes helpful to think of conflicts as forks in the version history of a change unit. If no conflicts occur in the life of a change unit, its version history is a simple chain --- each change occurring after the previous one. In the case of a knowledge-based conflict, two changes occur in parallel, causing the chain to split and become a version tree.

(b) Constraint-based conflicts

[0278] There are cases where independent changes violate an integrity constraint when applied together. For instance, two replicas creating a file with the same name in the same directory could cause such a conflict to occur.

[0279] A constraint-based conflict involves two independent changes (just like a knowledge-based one), but they do not affect the same change unit. Rather, they affect different change units but with a constraint existing between them.

[0280] The synchronization service detects constraint violations at change application time and raises constraint-based conflicts automatically. Resolving constraint-based conflicts usually requires custom code that modifies the changes in such as way as to not violate the constraint; The synchronization service does not provide a general-purpose mechanism for doing so.

(2) Conflict Processing

[0281] When a conflict is detected, the synchronization service can take one of three actions (selected by the sync initiator in the Sync Profile): (1) reject the change, returning it back to sender; (2) log a conflict into a conflict log; or (3) resolve the conflict automatically.

[0282] If the change is rejected, the synchronization service acts as if the change did not arrive at the replica. A negative acknowledgement is sent back to the originator. This resolution policy is primarily useful on head-less replicas (such as file servers) where logging conflicts is not feasible. Instead, such replicas force the others to deal with the conflicts by rejecting them.

[0283] Sync initiators configure conflict resolution in their Sync Profiles. The synchronization service supports combining multiple conflict resolvers in a single profile in the following ways – first, by specifying a list of conflict resolvers to be tried one after another, until one of them succeeds; and second, by associating conflict resolvers with conflict types, e.g. directing update-update knowledge-based conflicts to one resolver, but all the other conflicts to the log.

(a) Automatic Conflict resolution

[0284] The synchronization service provides a number of default conflict resolvers. This list includes:

- local-wins: disregard incoming changes if in conflict with locally stored data;
- remote-wins: disregard local data if in conflict with incoming changes;
- last-writer-wins: pick either local-wins or remote-wins per change unit based on the timestamp of the change (note that the synchronization service in general does not rely on clock values; this conflict resolver is the sole exception to that rule);
- Deterministic: pick a winner in a manner that is guaranteed to be the same on all replicas, but not otherwise meaningful – one embodiment of the synchronization services uses lexicographic comparisons of partner IDs to implement this feature.

[0285] In addition, ISVs can implement and install their own conflict resolvers. Custom conflict resolvers may accept configuration parameters; such parameters must be specified by the SCA in the Conflict Resolution section of the Sync Profile.

[0286] When a conflict resolver handles a conflict, it returns the list of operations that need to be performed (in lieu of the conflicting change) back to the runtime. The synchronization service then applies these operations, having properly adjusted remote knowledge to include what the conflict handler has considered.

[0287] It is possible that another conflict is detected while applying the resolution. In such a case, the new conflict must be resolved before the original processing resumes.

[0288] When thinking of conflicts as branches in the version history of an item, conflict resolutions can be viewed as joins --- combining two branches to form a single point. Thus, conflict resolutions turn version histories into DAGs.

(b) Conflict Logging

[0289] A very particular kind of a conflict resolver is the Conflict Logger. The synchronization service logs conflicts as Items of type ConflictRecord. These records are related back to the items that are in conflict (unless the items themselves have been deleted). Each conflict record contains: the incoming change that caused the conflict; the type of the conflict: update-update, update-delete, delete-update, insert-insert, or constraint; and the version of the incoming change and the knowledge of the replica sending it. Logged conflicts are available for inspection and resolution as described below.

(c) Conflict inspection and resolution

[0290] The synchronization service provides an API for applications to examine the conflict log and to suggest resolutions of the conflicts in it. The API allows application to enumerate all conflicts, or conflicts related to a given Item. It also allows such applications to resolve logged conflicts in one of three ways: (1) remote wins — accepting the logged change and overwriting the conflicting local change; (2) local wins --- ignoring conflicting parts of the logged change; and (3) suggest new change --- where the application proposes a merge that, in its opinion, resolves the conflict. Once conflicts are resolved by an application, the synchronization service removes them from the log.

(d) Convergence of replicas and Propagation of Conflict Resolutions

[0291] In complex synchronization scenarios, the same conflict can be detected at multiple replicas. If this occurs, several things can happen: (1) the conflict can be resolved on one replica, and the resolution be sent to the other; (2) the conflict is resolved on both replicas automatically; or (3) the conflict is resolved on both replicas manually (through the conflict inspection API).

[0292] To ensure convergence, the synchronization service forwards conflict resolutions to other replicas. When a change that resolves a conflict arrives at a replica, the synchronization service automatically finds any conflict records in the log that are resolved by this update and eliminates them. In this sense, a conflict resolution at one replica is binding on all the other replicas.

[0293] If different winners are chosen by different replicas for the same conflict, the synchronization service applies the principle of binding conflict resolution and picks one of the two resolutions to win over the other automatically. The winner is picked in a deterministic fashion that is guaranteed to produce the same results at all times (one embodiment uses replica ID lexicographic comparisons).

[0294] If different "new changes" are suggested by different replicas for the same conflict, the synchronization service treats this new conflict as a special conflict and uses the Conflict Logger to prevent it from propagating to other replicas. Such situation commonly arises with manual conflict resolution.

2. Synchronizing to non-storage platform data stores

[0295] According to another aspect of the storage platform of the present invention, the storage platform provides an architecture for ISVs to implement Sync Adapters that allow the storage platform to synchronize to legacy systems such as Microsoft Exchange, AD, Hotmail, etc. Sync Adapters benefit from the many Sync Service provided by the synchronization service, as described below.

[0296] Despite the name, Sync Adapters do not need to be implemented as plug-ins into some storage platform architecture. If desired, a "sync adapter" can simply be any application that utilizes the synchronization service runtime interfaces to obtain services such as change enumeration and application.

[0297] In order to make it simpler for others to configure and run synchronization to a given backend, Sync Adapter writers are encouraged to expose the standard Sync Adapter interface, which runs sync given the Sync Profile as described above. The profile provides configuration information to the adapter, some of which adapters pass to the Sync Runtime to control runtime services (e.g. the Folder to synchronize).

a) Sync Services

[0298] The synchronization service provides a number of sync services to adapter writers. For the rest of this section, it is convenient to refer to the machine on which the storage platform is doing synchronization as the "client" and the non-storage platform backend that the adapter is talking to as the "server".

(1) Change Enumeration

[0299] Based on the change-tracking data maintained by the synchronization service, Change Enumeration allows sync adapters to easily enumerate the changes that have occurred to a data store Folder since the last time synchronization with this partner was attempted.

[0300] Changes are enumerated based on the concept of an "anchor" — an opaque structure that represents information about the last synchronization. The anchor takes the form of the storage platform Knowledge, as described in the proceeding sections. Sync adapters utilizing change enumeration services fall into two broad categories: those using "stored anchors" vs. those using "supplied anchors".

[0301] The distinction is based on where the information about the last sync is stored -- on the client, or on the server. It is often easier for adapters to store this information on the client --- the backend is often not capable of conveniently storing this information. On the other hand, if multiple clients synchronize to the same backend, storing this information on the client is inefficient and in some cases incorrect --- it makes one client unaware of the changes that the other client has already pushed up to the server. If an adapter wants to use a server-stored anchor, the adapter needs to supply it back to the storage platform at the time of change enumeration.

[0302] In order for the storage platform to maintain the anchor (either for local or remote storage), the storage platform needs to be made aware of the changes that were

successfully applied at the server. These and only these changes can be included in the anchor. During change enumeration, Sync Adapters use an Acknowledgement interface to report which changes were successfully applied. At the end of synchronization, adapters using supplied anchors must read the new anchor (which incorporates all of the successfully-applied changes) and send it to their backend.

[0303] Often, Adapters need to store adapter-specific data along with the items they insert into the storage platform data store. Common examples of such data are remote IDs and remote versions (timestamps). The synchronization service provides a mechanism for storing this data, and Change Enumeration provides a mechanism to receive this extra data along with the changes being returned. This eliminates the need for adapters to re-query the database in most cases.

(2) Change Application

[0304] Change Application allows Sync Adapters to apply changes received from their backend to the local storage platform. Adapters are expected to transform the changes to the storage platform schema. Fig. 24 illustrates the process by which storage platform API classes are generated from the storage platform Schema.

[0305] The primary function of change application is to automatically detect conflicts. As in the case of Storage Platform-to-Storage Platform sync, a conflict is defined as two overlapping changes being made without knowledge of each other. When adapters use Change Application, they must specify the anchor with respect to which conflict detection is performed. Change Application raises a conflict if an overlapping local change that is not covered by the adapter's knowledge is detected. Similar to Change Enumeration, adapters may use either stored or supplied anchors. Change Application supports efficient storage of adapter-specific meta-data. Such data may be attached by the adapter to the changes being applied, and might be stored by the synchronization service. The data might be returned on next change enumeration.

(3) Conflict Resolution

[0306] The Conflict Resolution mechanisms described above (logging and automatic resolution options) are available to sync adapters as well. Sync adapters may specify the policy for conflict resolution when applying changes. If specified, conflicts may be passed on to the

specified conflict handler and resolved (if possible). Conflicts can also be logged. It is possible that the adapter may detect a conflict when attempting to apply a local change to the backend. In such a case, the adapter may still pass the conflict on to the Sync Runtime to be resolved according to policy. In addition, Sync Adapters may request that any conflicts detected by the synchronization service be sent back to them for processing. This is particularly convenient in the case where the backend is capable of storing or resolving conflicts.

b) Adapter Implementation

[0307] While some "adapters" are simply applications utilizing runtime interfaces, adapters are encouraged to implement the standard adapter interfaces. These interfaces allow Sync Controlling Applications to: request that the adapter perform synchronization according to a given Sync Profile; cancel on-going synchronization; and receive progress reporting (percentage complete) on an ongoing sync.

3. Security

[0308] The synchronization service strives to introduce as little as possible into the security model implemented by the storage platform. Rather than defining new rights for synchronization, existing rights are used. Specifically,

- anyone who can read a data store Item can enumerate changes to that item;
- anyone who can write to a data store Item can apply changes to that item; and
- anyone who can extend a data store Item can associate sync meta-data with that item.

[0309] The synchronization service does not maintain secure authorship information. When a change is made at replica A by user U and forwarded to replica B, the fact that the change was originally made at A (or by U) is lost. If B forwards this change to replica C, this is done under B's authority, not that of A. This leads to the following limitation: if a replica is not trusted to make its own changes to an item, it cannot forward changes made by others.

[0310] When the synchronization service is initiated, it is done by a Sync Controlling Application. The synchronization service impersonates the identity of the SCA and performs all operations (both locally and remotely) under that identity. To illustrate, observe that user U

cannot cause the local synchronization service to retrieve changes from a remote storage platform for items that user U does not have read access.

4. Manageability

[0311] Monitoring a distributed community of replicas is a complex problem. The synchronization service may use a "sweep" algorithm to collect and distribute information about the status of the replicas. The properties of the sweep algorithm ensure that information about all configured replicas is eventually collected and that failing (non-responsive) replicas are detected.

[0312] This community-wide monitoring information is made available at every replica. Monitoring tools can be run at an arbitrarily-chosen replica to examine this monitoring information and make administrative decisions. Any configuration changes must be made directly at the affected replicas.

J. TRADITIONAL FILE SYSTEM INTEROPERABILITY

[0313] As mentioned above, the storage platform of the present invention is, in at least some embodiments, intended to be embodied as an integral part of the hardware/software interface system of a computer system. For example, the storage platform of the present invention may be embodied as an integral part of an operating system, such as the Microsoft Windows family of operating systems. In that capacity, the storage platform API becomes a part of the operating system APIs through which application programs interact with the operating system. Thus, the storage platform becomes the means through which application programs store information on the operating system, and the Item based data model of the storage platform therefore replaces the traditional files system of such an operating system. For example, as embodied in the Microsoft Windows family of operating systems, the storage platform might replace the NTFS file system implemented in that operating system. Presently, application programs access the services of the NTFS file system through the Win32 APIs exposed by the Windows family of operating systems.

[0314] Recognizing, however, that completely replacing the NTFS file system with the storage platform of the present invention would require recoding of existing Win32-based application programs and that such recoding may be undesirable, it would be beneficial for the storage platform of the present invention to provide some interoperability with existing file

systems, such as NTFS. In one embodiment of the present invention, therefore, the storage platform enables application programs which rely on the Win32 programming model to access the contents of both the data store of the storage platform as well as the traditional NTFS file system. To this end, the storage platform uses a naming convention that is a superset of the Win32 naming conventions to facilitate easy interoperability. Further, the storage platform supports accessing files and directories stored in a storage platform volume through the Win32 API.

[0315] Additional details regarding this functionality can be found in the related applications incorporated by reference earlier herein.

K. STORAGE PLATFORM API

[0316] The storage platform comprises an API that enables application programs to access the features and capabilities of the storage platform discussed above and to access items stored in the data store. This section describes one embodiment of a storage platform API of the storage platform of the present invention. Details regarding this functionality can be found in the related applications incorporated by reference earlier herein, with some of this information summarized below for convenience.

[0317] Referring to Fig. 18, a Containment Folder is an item which contains holding Relationships to other Items and is the equivalent of the common concept of a file system folder. Each Item is "contained" within at least one containment folder.

[0318] Fig. 19 illustrates the basic architecture of the storage platform API, in accordance with the present embodiment. The storage platform API uses SQLClient 1900 to talk to the local data store 302 and may also use SQLClient 1900 to talk to remote data stores (e.g., data store 340). The local store 302 may also talk to the remote data store 340 using either DQP (Distributed Query Processor) or through the the storage platform synchronization service ("Sync") described above. The storage platform API 322 also acts as the bridge API for data store notifications, passing application's subscriptions to the notification engine 332 and routing notifications to the application (*e.g.*, application 350a, 350b, or 350c), as also described above. In one embodiment, the storage platform API 322 may also define a limited "provider" architecture so that it can access data in Microsoft Exchange and AD.

[0319] Fig. 20 schematically represents the various components of the storage platform API. The storage platform API consists of the following components: (1) data classes 2002, which represent the storage platform element and item types, (2) runtime framework 2004, which manages object persistence and provides support classes 2006; and (3) tools 2008, which are used to generate CLR classes from the storage platform schemas.

[0320] The hierarchy of classes resulting from a given schema directly reflects the hierarchy of types in that schema. As an example, consider the Item types defined in the Contacts schema as shown in Fig. 21A and Fig. 21B.

[0321] Fig. 22 illustrates the runtime framework in operation. The runtime framework operates as follows:

1. An application 350a, 350b, or 350c binds to an item in the storage platform.
2. The framework 2004 creates an ItemContext object 2202 corresponding to the bound item and returns it to the application.
3. The application submits a Find on this ItemContext to get a collection of Items; the returned collection is conceptually an object graph 2204 (due to relationships).
4. The application changes, deletes, and inserts data.
5. The application saves the changes by calling the Update() method.

[0322] Fig. 23 illustrates the execution of a "FindAll" operation.

[0323] Fig. 24 illustrates the process by which storage platform API classes are generated from the storage platform Schema

[0324] Fig. 25 illustrates the schema on which the File API is based. The storage platform API includes a namespace for dealing with file objects. This namespace is called System.Storage.Files. The data members of the classes in System.Storage.Files directly reflect the information stored in the storage platform store; this information is "promoted" from the file system objects or may be created natively using the Win32 API. The System.Storage.Files namespace has two classes: FileItem and DirectoryItem. The members of these classes and methods thereof can be readily divined by looking at the schema diagram in Fig. 25. FileItem and DirectoryItem are read-only from the storage platform API. In order to modify them, one has to use the Win32 API or classes in System.IO.

[0325] In regard to APIs, a programming interface (or more simply, interface) may be viewed as any mechanism, process, protocol for enabling one or more segment(s) of code to communicate with or access the functionality provided by one or more other segment(s) of code. Alternatively, a programming interface may be viewed as one or more mechanism(s), method(s), function call(s), module(s), object(s), etc. of a component of a system capable of communicative coupling to one or more mechanism(s), method(s), function call(s), module(s), etc. of other component(s). The term "segment of code" in the preceding sentence is intended to include one or more instructions or lines of code, and includes, e.g., code modules, objects, subroutines, functions, and so on, regardless of the terminology applied or whether the code segments are separately compiled, or whether the code segments are provided as source, intermediate, or object code, whether the code segments are utilized in a runtime system or process, or whether they are located on the same or different machines or distributed across multiple machines, or whether the functionality represented by the segments of code are implemented wholly in software, wholly in hardware, or a combination of hardware and software.

[0326] Notionally, a programming interface may be viewed generically, as shown in Fig. 30A or Fig. 30B. Fig. 30A illustrates an interface Interface1 as a conduit through which first and second code segments communicate. Fig. 30B illustrates an interface as comprising interface objects I1 and I2 (which may or may not be part of the first and second code segments), which enable first and second code segments of a system to communicate via medium M. In the view of Fig. 30B, one may consider interface objects I1 and I2 as separate interfaces of the same system and one may also consider that objects I1 and I2 plus medium M comprise the interface. Although Figs. 30A and 30B show bi-directional flow and interfaces on each side of the flow, certain implementations may only have information flow in one direction (or no information flow as described below) or may only have an interface object on one side. By way of example, and not limitation, terms such as application programming interface (API), entry point, method, function, subroutine, remote procedure call, and component object model (COM) interface, are encompassed within the definition of programming interface.

[0327] Aspects of such a programming interface may include the method whereby the first code segment transmits information (where "information" is used in its broadest sense and includes data, commands, requests, etc.) to the second code segment; the method whereby the

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

[0328] This notion of a programming interface is known to those skilled in the art and is clear from the foregoing detailed description of the invention. There are, however, other ways to implement a programming interface, and, unless expressly excluded, these too are intended to be encompassed by the claims set forth at the end of this specification. Such other ways may appear to be more sophisticated or complex than the simplistic view of Figs. 30A and 30B, but they nonetheless perform a similar function to accomplish the same overall result. We will now briefly describe some illustrative alternative implementations of a programming interface.

[0329] Factoring: A communication from one code segment to another may be accomplished indirectly by breaking the communication into multiple discrete communications. This is depicted schematically in Figs. 31A and 31B. As shown, some interfaces can be described in terms of divisible sets of functionality. Thus, the interface functionality of Figs. 30A and 30B may be factored to achieve the same result, just as one may mathematically provide 24, or 2 times 2 time 3 times 2. Accordingly, as illustrated in Fig. 31A, the function provided by interface Interface1 may be subdivided to convert the communications of the interface into multiple interfaces Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As illustrated in Fig. 31B, the function provided by interface I1 may be subdivided into multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly, interface I2 of the second code segment which receives information from the first code segment may be

factored into multiple interfaces I2a, I2b, I2c, etc. When factoring, the number of interfaces included with the 1st code segment need not match the number of interfaces included with the 2nd code segment. In either of the cases of Figs. 31A and 31B, the functional spirit of interfaces Interface1 and I1 remain the same as with Figs. 30A and 30B, respectively. The factoring of interfaces may also follow associative, commutative, and other mathematical properties such that the factoring may be difficult to recognize. For instance, ordering of operations may be unimportant, and consequently, a function carried out by an interface may be carried out well in advance of reaching the interface, by another piece of code or interface, or performed by a separate component of the system. Moreover, one of ordinary skill in the programming arts can appreciate that there are a variety of ways of making different function calls that achieve the same result.

[0330] Redefinition: In some cases, it may be possible to ignore, add or redefine certain aspects (e.g., parameters) of a programming interface while still accomplishing the intended result. This is illustrated in Figs. 32A and 32B. For example, assume interface Interface1 of Fig. 30A includes a function call Square(input, precision, output), a call that includes three parameters, input, precision and output, and which is issued from the 1st Code Segment to the 2nd Code Segment. If the middle parameter precision is of no concern in a given scenario, as shown in Fig. 32A, it could just as well be ignored or even replaced with a meaningless (in this situation) parameter. One may also add an additional parameter of no concern. In either event, the functionality of square can be achieved, so long as output is returned after input is squared by the second code segment. Precision may very well be a meaningful parameter to some downstream or other portion of the computing system; however, once it is recognized that precision is not necessary for the narrow purpose of calculating the square, it may be replaced or ignored. For example, instead of passing a valid precision value, a meaningless value such as a birth date could be passed without adversely affecting the result. Similarly, as shown in Fig. 32B, interface I1 is replaced by interface I1', redefined to ignore or add parameters to the interface. Interface I2 may similarly be redefined as interface I2', redefined to ignore unnecessary parameters, or parameters that may be processed elsewhere. The point here is that in some cases a programming interface may include aspects, such as parameters, that

are not needed for some purpose, and so they may be ignored or redefined, or processed elsewhere for other purposes.

[0331] Inline Coding: It may also be feasible to merge some or all of the functionality of two separate code modules such that the "interface" between them changes form. For example, the functionality of Figs. 30A and 30B may be converted to the functionality of Figs. 33A and 33B, respectively. In Fig. 33A, the previous 1st and 2nd Code Segments of Fig. 30A are merged into a module containing both of them. In this case, the code segments may still be communicating with each other but the interface may be adapted to a form which is more suitable to the single module. Thus, for example, formal Call and Return statements may no longer be necessary, but similar processing or response(s) pursuant to interface Interface1 may still be in effect. Similarly, shown in Fig. 33B, part (or all) of interface I2 from Fig. 30B may be written inline into interface I1 to form interface I1". As illustrated, interface I2 is divided into I2a and I2b, and interface portion I2a has been coded in-line with interface I1 to form interface I1". For a concrete example, consider that the interface I1 from Fig. 30B performs a function call square (input, output), which is received by interface I2, which after processing the value passed with input (to square it) by the second code segment, passes back the squared result with output. In such a case, the processing performed by the second code segment (squaring input) can be performed by the first code segment without a call to the interface.

[0332] Divorce: A communication from one code segment to another may be accomplished indirectly by breaking the communication into multiple discrete communications. This is depicted schematically in Figs. 34A and 34B. As shown in Fig. 34A, one or more piece(s) of middleware (Divorce Interface(s), since they divorce functionality and / or interface functions from the original interface) are provided to convert the communications on the first interface, Interface1, to conform them to a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. This might be done, e.g., where there is an installed base of applications designed to communicate with, say, an operating system in accordance with an Interface1 protocol, but then the operating system is changed to use a different interface, in this case interfaces Interface2A, Interface2B and Interface2C. The point is that the original interface used by the 2nd Code Segment is changed such that it is no longer compatible with the interface used by the 1st Code Segment, and so an intermediary is used to make the old and new interfaces

compatible. Similarly, as shown in Fig. 34B, a third code segment can be introduced with divorce interface DI1 to receive the communications from interface I1 and with divorce interface DI2 to transmit the interface functionality to, for example, interfaces I2a and I2b, redesigned to work with DI2, but to provide the same functional result. Similarly, DI1 and DI2 may work together to translate the functionality of interfaces I1 and I2 of Fig. 30B to a new operating system, while providing the same or similar functional result.

[0333] Rewriting: Yet another possible variant is to dynamically rewrite the code to replace the interface functionality with something else but which achieves the same overall result. For example, there may be a system in which a code segment presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is provided to a Just-in-Time (JIT) compiler or interpreter in an execution environment (such as that provided by the .Net framework, the Java runtime environment, or other similar runtime type environments). The JIT compiler may be written so as to dynamically convert the communications from the 1st Code Segment to the 2nd Code Segment, i.e., to conform them to a different interface as may be required by the 2nd Code Segment (either the original or a different 2nd Code Segment). This is depicted in Figs. 35A and 35B. As can be seen in Fig. 35A, this approach is similar to the Divorce scenario described above. It might be done, e.g., where an installed base of applications are designed to communicate with an operating system in accordance with an Interface 1 protocol, but then the operating system is changed to use a different interface. The JIT Compiler could be used to conform the communications on the fly from the installed-base applications to the new interface of the operating system. As depicted in Fig. 35B, this approach of dynamically rewriting the interface(s) may be applied to dynamically factor, or otherwise alter the interface(s) as well.

[0334] It should also be noted that the above-described scenarios for achieving the same or similar result as an interface via alternative embodiments may also be combined in various ways, serially and/or in parallel, or with other intervening code. Thus, the alternative embodiments presented above are not mutually exclusive and may be mixed, matched and combined to produce the same or equivalent scenarios to the generic scenarios presented in Figs. 30A and 30B. It is also noted that, as with most programming constructs, there are other similar ways of achieving the same or similar functionality of an interface which may not be described herein, but nonetheless are represented by the spirit and scope of the invention, i.e., it is noted

that it is at least partly the functionality represented by, and the advantageous results enabled by, an interface that underlie the value of an interface.

III. EXTENSIONS AND INHERITANCE

[0335] A foundational concept of the present invention is the utilization of Items that, to a certain extent, model real-world application objects with complex structures, behaviors, and operations described by a schema and enforced by the hardware/software interface system. To provide rich sub-typing functionality, and in various embodiments of the present invention, a hardware/software interface system (which, for convenience, we shall simply refer to as "WinFS" hereafter) may provides a mechanism by which Items (and Item types) can be extended using "Extensions." Extensions provide additional data structures (Properties, Relationships, etc.) to already existing Items type structures.

[0336] As previously discussed herein (and particularly discussed in Sections II.E.6.(a) and II.F.3), while the storage platform is intended to be provided with an initial set of schemas, in at least some embodiments the storage platform allows customers, including independent software vendor (ISVs), to create new schemas (i.e., new Item and Nested Element types). Since an Item type or Nested Element type defined by the initial set of storage platform schemas may not exactly match an ISV application's need, it is necessary to allow ISVs to customize the type. This is allowed with the notion of Extensions. Extensions are strongly typed instances but (a) they cannot exist independently and (b) they must be attached to an Item or Nested Element. Also, in addition to addressing the need for schema extensibility, Extensions are also intended to address "multi-typing" issues. Since, in some embodiments, the storage platform may not support multiple inheritance or overlapping subtypes, applications can use Extensions as a way to model overlapping type instances (e.g., a Document may be a "legal document" as well a "secure document").

A. TYPE SYSTEM

[0337] In various embodiments of the present invention, the WinFS type system provides a mechanism for defining structures of data. The type system is used to represent data stored in WinFS. A WinFS type is declared in a WinFS schema. A WinFS schema defines a

namespace which serves as a logical grouping for a set of types and other WinFS schema elements. WinFS Schemas may be declared using a WinFS Schema Definition Language (SDL) that might use an XML format. The following is an example of a possible schema declaration:

```
<Schema Namespace="System.Storage" >
  Type definitions
</Schema>
```

[0338] WinFS Schemas also serve as a unit for type versioning. WinFS defines several system schemas that bootstrap the system. These are the System.Storage schema namespace that contains the type declarations of the root types in the system, and the System.Storage.WinFS schema namespace that declares all the primitive scalar types in the system.

[0339] TheWinFS type system declares a set of simple scalar types. These types are used as the most primitive building block for all other types in the WinFS type system. These types are declared in the schema namespace System.Storage.WinFS. The following table defines the set of primitive types.

WinFS Type	Managed SQL Type	CLR Type	Description
String	SqlString	String	Variable-length Unicode data with a maximum length of length of 2^31 characters. Length can be fixed from 1 – 4000 characters or unconstrained using the "max" keyword.
Binary	SqlBinary	Byte[]	Variable-length binary data with a maximum length of length of 2^32 bytes. Length can be fixed from 1 – 8000 bytes or unconstrained using the "max" keyword.
Boolean	SqlBoolean	Boolean	Null-able Boolean value
Byte	SqlByte	Byte	A singe unsigned byte
Int16	SqlInt16	Int16	Integer data from -2^15 (-32,768) through 2^15 - 1 (32,767).
Int32	SqlInt32	Int32	Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647).
Int64	SqlInt64	Int64	Integer (whole number) data from -2^63 (-

			9223372036854775808) through 2^63-1 (9223372036854775807).
Single	SqlSingle	Single	Floating precision number data from -1.79E + 308 through 1.79E + 308
Double	SqlDouble	Double	Floating precision number data from -3.40E + 38 through 3.40E + 38
Decimal	SqlDecimal	Decimal	SQLDecimal has a larger value range then CLR Decimal type. In the store the precision is always 28 digits and scale is 0.
DateTime	SqlDateTime	DateTime	Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.
Guid	SqlGuid	Guid	A globally unique identifier (GUID).
Xml	SqlXmlReader	XmlReader	For Milestone B, WinFS.Xml will be mapped to "String" type. True XML datatype support is expected in milestone C.
Stream	TBD	TBD	A binary data type that is using file stream backed storage for efficient access. This type will be supported in milestone C.

[0340] A WinFS enumeration is a scalar type that declares a set of named constants called a value list. An enumeration type can be used in any place where a scalar type can be used. Here is an example for an enumeration declaration:

```
<Enumeration Name="Gender" >
 <Value Name="Male" />
 <Value Name="Female" />
</Enumeration>
```

The values of the enumeration are zero based. In the example above Gender.Male represents value 0 and Gender.Female represents value 1.

[0341] A complex type is defined by a name and a set of properties. A Property is a member field of the type, and is defined by a name and a type. The type of a Property can be either scalar (including enumeration type) or other complex type. A WinFS type that can be used

as type of a Property is called a nested type. An instance of a nested type can only exist as a value of a Property of a complex WinFS type—the instance is nested within an instance of a complex type. A nested type is declared using the NestedType schema element. Here are some examples of valid type declarations:

```
<NestedType Name="Address" BaseType="System.Storage.NestedObject">
 <Property Name="Street" Type="WinFS.String" Size="256"
Nullable="false" />
 <Property Name="City" Type=" WinFS.String" Size="256"
Nullable="false" />
 <Property Name="State" Type=" WinFS.String" Size="256"
Nullable="false" />
 <Property Name="Country" Type="WinFSString" Size="256"
Nullable="false" />
</NestedType>

<ItemType Name="Person" BaseType="System.Storage.Item">
 <Property Name="Name" Type="WinFS.String" Size="256"
Nullable="false" />
 <Property Name="Age" Type="WinFS.Int32"
Nullable="false" Default="1"/>
 <Property Name="Picture" Type="WinFS.Binary" Size="max"/>
 <Property Name="Addresses" Type="MultiSet" MultiSetOfType="Address"/>
</ItemType>
```

[0342] For properties of type String and Binary a Size attribute must be specified. This attribute specifies the maximum size of the values contained in the Property. A Property can optionally declare a nullability constraint using the Nullable attribute. A value "false" for this attribute indicates that the application must provide a value when creating an instance of the type. Another optional Property attribute is the Default attribute that specifies the default value

for the Property. This value will be assigned to the Property at instance creation time if the application did not provide it.

[0343] The Addresses Property in the example above is of type MultiSet. A Property of type MultiSet is also referred to as a multi-valued Property. In the example the MultiSet contains a set of instance of type Address. A MultiSet is similar to a collection. It can contain zero or more instances of a complex type. The type of the instance in the MultiSet must be a complex nested type. MultiSet type does not support instances of WinFS scalar types (including enumeration types). A Property of type MultiSet can not be nullable and can not have a default value.

[0344] WinFS supports single-inheritance of types. All types in WinFS must inherit from one and only on WinFS type. The inheriting type is referred to as the derived type and the type from which this type was derived is referred to as the base type. The base type in the BaseType attribute of the WinFS type declaration elements. Suppose that type A derives from base type B which in turn derives from type C. The type C is an ancestor type of the types A and B. The type A is a descendant type of the types B and C. A data instance stored in WinFS is always an instance of a single type. We can however treat that data instance as an instance of a set of types containing the type and all its ancestor types. For a data instance which is an instance of such a set of types, we call the type which is not the ancestor of any other type in the set the most-derived type. A singly-typed data instance is an instance of exactly one most-derived type. In general, we will refer to the most-derived type of a singly-typed element as its type. The derived type inherits all the properties declared in its base type. The derived type can declare new properties but can not override the properties defined in the base type. A Property declared in the derived type must not use the same name as a Property of the base type.

[0345] The main advantage of inheritance in the data model comes from substitutability of inherited types. Consider the following example:

```
<NestedType Name="Name" BaseType="System.Storage.NestedObject" >
  <Property Name="FirstName" Type="WinFS.String" />
  <Property Name="LastName" Type="WinFS.String" />
</Nestedype>
```

```
<NestedType Name="NameWithMiddleInitial" BaseType="Name" >
  <Property Name="MiddleInitial" Type="WinFS.String" />
</NestedType>

<NestedType Name="Person" BaseType="System.Storage.Item" >
  <Property Name="RealName" Type="Name" />
  <Property Name="OtherNames" Type="MultiSet" MultiSetOfType="Name" />
</NestedType>
```

In the above example, type Person has a Property RealName of type Name and a Property OtherNames which is a set of type Name. Normally it would be required that the Property RealName only have instances whose type is Name. However, with inheritance, single-valued instances are allowed to be the value of RealName as long as the type Name is one of ancestors of the most-derived type of that element. So, an instance of NameWithMiddleInitial will be allowed to be the value of the Property RealName.

[0346] The same rule is extended to set properties. The Property OtherNames contains a set of elements. For each singly-typed instance which is a member of that set, the most-derived type of that instance must have Name as one of its ancestors. So, some of the instances in the set OtherNames may be instances of type Name while others maybe instances of type NameWithMiddleInitial.

[0347] Inheritance also enables convenient querying in that it is possible in the WinFS system to find all instances of a certain type. When looking for all instances of a type, the query engine will also return all instances whose most-derived types are descendents of this type. However, these operations are only supported for Item, Extension, and Relationship types (not Property types). For nested types (a.k.a., Nested Elements, Property, or complex property types), the operation is only supported for instances contained within a single multiset Property.

B. TYPE FAMILIES

[0348] In summary, the WinFS type system defines four distinct type families:

- Nested Element types (a.k.a., Nested types or Property types)
- Item types
- Relationship types
- Extension types

Each type family has a different set of properties and usage in the WinFS type system. The System.Storage schema namespace declares four types that serve as root types for each of the type family. The following sections describe the type families in detail.

1. Nested Element Types

[0349] Unlike the other WinFS Type families, nested types can be used as types of properties of complex WinFS types. Instances of a nested type can only be nested inside an instance of another complex type. However, Instances of nested types cannot be queried globally—that is, applications cannot compose a simple query that returns all instances of a given nested type in the WinFS store.

2. Item Types

[0350] A WinFS Item is an instance of a type whose ancestor is the type System.Storage.Item. This type is a complex type that is the root of the Item type family. System.Storage.Item declares a Property of name ItemId of type Guid. This is a special Property of an Item that serves as a primary key of the Item. The value of this Property is guaranteed to be unique for Items in a given WinFS store. This Property is non-nullable, and must be assigned by the application when creating an instance of an Item type. The ItemId Property is also immutable – it can never change and must not be reused.

[0351] The query engine can return instances of a given Item type in a WinFS store. This query may returns all instances of the type and all its type descendent types. Later herein is described the central role that Items have in the WinFS system operational semantics.

3. Relationship Types

[0352] Relationship types enable Relationships to exist between Items. WinFS Relationship types describe binary Relationships where one Item is designated as the source and

the other Item is designated as the target. A Relationship is an instance of a type whose ancestor is the type System.Storage.Relationship. This type is a root of the Relationship type hierarchy. System.Storage.Relationship type declares the following properties:

- SourceItemId – the ItemId of the Item that is a source of the Relationship instance
- RelationshipId – a unique identifier of the Relationship relative to the source Item; the pair (SourceItemId, RelationshipId) forms the primary key for Relationship types in WinFS
- TargetItemId – the ItemId of the target of the Relationship;
- Mode – one of the 3 possible Relationship instance modes: Holding, Embedding or Reference
- Name – contains the name of the Relationship for holding Relationships
- IsHidden – a Boolean attribute that applications can optionally use to filter Relationships that do not need to be displayed

The SourceItemId, RelationshipId, TargetItemId and Mode Property values are immutable. They are assigned at Relationship instance creation time and can not be changed.

[0353] A Relationship type is declared as a complex type with the following additional constraints:

- Source and target endpoint specification: each endpoint specifies a name and the type of the referenced Item
- Allowed modes of the instances of the Relationship type: a Relationship instance can not have a value for the Mode Property that is not allowed in the Relationship declaration

[0354] Here is an example of a Relationship declaration:

```
<RelationshipType Name="DocumentAuthor"
BaseType="System.Storage.Relationship"
AllowsHolding="true"
AllowsEmbedding="false"
AllowsReference="true" >
 <Source Name="Document" Type="Core.Document"/>
```

```
<Target Name="Author" Type="Core.Contact" />
<Property Name="Role" Type="WinFS.String" />
<Property Name="DisplayName" Type="WinFS.String" />
</RelationshipType>
```

The DocumentAuthor Relationship is declared with restrictions of instances to Holding or Reference mode. This means that an instance of the DocumentAuthor Relationship can have instances with value Mode="Reference" or Mode="Holding". Instances with value Mode="Embedding" will not be allowed.

[0355] The Relationship declares a source endpoint named "Document" of Item type "Core.Document" and a target endpoint of type "Core.Contact". The Relationship also declares two additional properties. The Relationship instances are stored and accessed separately from the Item. All Relationship type instances are accessible from a global Extension view. A query can be composed that will return all the instances of a given type of Relationship.

[0356] Given an Item all the Relationships for which the Item is a source can be enumerated based on the SourceItemId Property of the Relationship. Similarly, for a given Item all the Relationship in the same store for which the Item is a target can be enumerated using the TargetItemId Property of the Relationship.

a) Relationship Semantics

[0357] The following sections describe the semantics of the different Relationship instance modes:

[0358] Holding Relationships: Holding Relationships are used to model reference count based life-time management of the target Items. An Item can be a source endpoint for zero or more Relationships to Items. An Item that is not an embedded Item can be a target of one or more holding Relationships. The target Item must be in the same store as the Relationship instance.

[0359] Holding Relationships enforce lifetime management of the target endpoint. The creation of a holding Relationship instance and the Item that it is targeting is an atomic operation. Additional holding Relationship instances can be created that are targeting the same

Item. When the last holding Relationship instance with a given Item as target endpoint is deleted the target Item is also be deleted.

[0360] The types of the endpoint Items specified in the Relationship declaration will be enforced when an instance of the Relationship is created. The types of the endpoint Items can not be changed after the Relationship is established.

[0361] Holding Relationships play a key role in forming the WinFS Item namespace. All holding Relationships participate in the namespace declaration. The "Name" Property in the Relationship declaration defines the name of the target Item relative to the source Item. This relative name is unique for all the holding Relationships sourced from a given Item and cannot be null. The ordered list of this relative names starting from the root Item to a given Item forms the full name to the Item.

[0362] The holding Relationships form a directed acyclic graph (DAG). When a holding Relationship is created the system ensures that a cycle is not created and thus ensuring that the WinFS Item namespace forms a DAG. For more information of the WinFS namespace and Item paths refer to the "WinFS Namespace" specification.

[0363] Embedding Relationships: Embedding Relationships are modeling the concept of exclusive control of the lifetime of the target Item. They enable the concept of compound Items. The creation of an embedding Relationship instance and the Item that it is targeting is an atomic operation. An Item can be a source of zero or more embedding Relationship. However, an Item can be a target of a one and only one embedding Relationship. An Item that is a target of embedding Relationship can not be a target of a holding Relationship. The target Item must be in the same store as the Relationship instance.

[0364] The types of the endpoint Items specified in the Relationship declaration will be enforced when an instance of the Relationship is created. The types of the endpoint Items can not be changed after the Relationship is established. The embedding Relationships do not participate in the WinFS namespace. The value of the Name Property of an embedding Relationship must be null.

[0365] Reference Relationships: Reference Relationships do not control life time of the Item it references. Reference Relationships do not guarantee the existence of the target, nor do they guarantee the type of the target as specified in the Relationship declaration. This means that

the reference Relationships can be dangling. Also, the reference Relationship can reference Items in other WinFS stores.

[0366] In WinFS reference Relationships will be used to model most non lifetime management Relationships between Items. Since the existence of the target is not enforced, the reference Relationship is convenient to model loosely-coupled Relationships. The reference Relationship can be used to target Items in other WinFS stores including stores on other machines. The embedding Relationships do not participate in the WinFS namespace. The value of the Name Property of an embedding Relationship must be null.

b) Relationship Rules and Contraints

[0367] The following additional rules and constraints apply to Relationships:

- An Item must be a target of (exactly one embedding Relationship) or (one or more holding Relationships). The only exception is the root Item. An Item can be a target of zero or more reference Relationships.
- An Item that is a target of embedding Relationship can not be source of holding Relationships. It can be a source of reference Relationships.
- An Item can not be a source of holding Relationship if it is promoted from file. It can be a source of embedding Relationships and reference Relationships.
- An Item can that is promoted from a file can not be a target of an embedding Relationship.

[0368] When a Relationship type A is derived from a base Relationship type B the following rules apply:

- Relationship type A can further restrict the endpoint types. The endpoint types must be subtypes of the corresponding endpoint type in the base Relationship B. If the endpoint is further restricted a new name for the endpoint must be declared. If the endpoint is not restricted the specification of the endpoint is optional
- Relationship type A can further restrict the allowed instance modes declared in the base Relationship. The restricted set of instance modes must be a subset of the base type set of allowed instance types.

- The names of the endpoints are treated as Property names: they can not be the same as a name of a Property or a name of an endpoint of the type or its base type.
- The Source and Target elements are optional if the corresponding endpoint type is not further restricted by the derived Relationship

Here is an example of a declaration of a Relationship type that derives from the DocumentAuthor Relationship defined herein above:

```
<RelationshipType Name="LegalDocumentAuthor"
BaseType="Core.DocumentAuthor"
AllowsHolding="false"
AllowsEmbedding="false"
AllowsReference="true" >
 <Source Name="LegalDocument" Type="Legal.Document "/>
 <Property Name="CaseNumber" Type="WinFS.String" />
</RelationshipType>
```

The LegalDocumentAuthor Relationship further restricts the source endpoint but not the target endpoint. The source endpoint type is Legal.Document which derives from Core.Document. The target endpoint is not restricted further in this case, hence the target element is omitted. The Relationship also further restricts the allowed instance modes. It disallows the Holding mode and only remaining allowed mode is Reference.

4. Extension Types

[0369] A WinFS Extension is an instance of a type whose ancestor is the type System.Storage.Extension. This type is a complex type that is the root of the Extension type family.

- System.Storage.Extension defines two properties:
- ItemId – the ItemId of the Item that the Extension is associated with
- ExtensionId – a unique identifier for the Extension relative to the ItemId. The pair (ItemId, ExtensionId) uniquely identifies an Extension instance.

[0370] The following restrictions apply for Extension types:

- Extension type instances can not exist independently from an Item. An Item type instance with the same ItemId as the Extension ItemId must exist in the store before the Extension type instance is created. The Extension can not be created if the Item with the given ItemId does not exist. When the Item is deleted all the Extensions with the same ItemID are deleted.
- At most one instance of a given most-derived Extension type can be associated with an individual Item.
- Extensions can not be sources and targets of Relationships.

[0371] There are no constraints on the types of Extensions that can be associated with a given Item type. Any Extension type is allowed to extend any Item type. When multiple instances of different Extension types are attached to an Item, they are independent from each other in both structure and behavior. The Extension instances are stored and accessed separately from the Item. All Extension type instances are accessible from a global Extension view. A query can be composed that will return all the instances of a given type of Extension regardless of what type of Item they are associated with. The ItemId of the Extension indicates which Item they belong to and can be used to retrieve the corresponding Item object from the global Item view. Also, given an Item all Extension instances associated with the Item can be enumerated using the ItemId Property of the Extension.

C. ENHANCED FUNCTIONALITY

[0372] In several embodiments of the present invention, a hardware/software interface system utilizes Extensions and Inheritance in order to formalize relationships between various Items and thereby enhance the ability to query a plurality of Items.

1. Inheritance

[0373] Fig. 36 illustrates a series of interrelated Items and a subset of their Relationships. The Document Item 3602 and a Contact Item 3604 are directly related by a designated relationship 3606 which, in this case, is an "author relationship"—that is, the Contact 3604 is the "author" of the Document 3602. In this example, the Picture Item 3622, the Music

Item 3624, and the Special Item 3626 all inherit from the Document Item 3604 because each Items' type is a subtype of the Document Item type. Similarly, the Person Item 3642 and the Organization Item 3644 inherit from the Contact Item 3604. In several embodiments of the present invention, these inheriting Items (Picture 3622, Music 3624, Special 3626, Person 3642, and Organization 3644) not only inherit the Properties of the respective parent Items (Document 3602 and Contact 3604), but they also inherit the designated Relationships between these two parent Items. For example, Picture 3622 inherits a relationship 3662 to Contact 3604, a relationship 3664 to Person 3642, and a relationship 3666 to Organization 3644. A similar set of Relationships are also inherited by each of the other Items shown.

[0374] However, it is important to note that relationship inheritance is not automatic and does not occur in every context. For example, attributes that describe when a type can be inherited (i.e., inheritance controls) are not themselves inheritable. Inheritance parameters are maintained and regulated by the hardware/software interface system.

2. Extensions

[0375] Fig. 37A illustrates the shortcoming of standard subtyping of an Item for application specific purposes. In this figure, a Contact is accessable by four applications, APP1, APP2, APPX, and APPY. APP1 and APP2 access the standard Contact, but each of APPX and APPY need an extended contact object (add additional fields) and thus derive Contact' and Contact'', each of which inherits from Contact. However, the problem is that there are now three different instances of the basic Contact Item—one in Contact, one in Contact', and one in Contact''.

[0376] A partial solution to this problem, illustrated in Fig. 37B, is to extend the properties of Contact to include the fields necessary for an application that requires such. In this case, Contact is extended to include the Additional Fields required by APPX. However, directly extending the fields of an Item such as Contact can only be done once, and thus APPY cannot employ this method.

[0377] In one embodiment of the present invention, a more comprehensive solution is to extend the Contact with an Extension that is distinct and separate from Contact itself as illustrated in Fig. 37C. In this way, APPX can extend Contact to include its APPX Additional Fields while APPY can also separately extend Contact to include its APPY Additional Fields.

These Extensions are then themselves searchable and queryable, and thus these Extensions enable a form of multi-typing for the hardware/software interface system.

IV. CONCLUSION

[0378] As the foregoing illustrates, the present invention is directed to a storage platform for organizing, searching, and sharing data. The storage platform of the present invention extends and broadens the concept of data storage beyond existing file systems and database systems, and is designed to be the store for all types of data, including structured, non-structured, or semi-structured data, such as relational (tabular) data, XML, and a new form of data called Items. Through its common storage foundation and schematized data, the storage platform of the present invention enables more efficient application development for consumers, knowledge workers and enterprises. It offers a rich and extensible application programming interface that not only makes available the capabilities inherent in its data model, but also embraces and extends existing file system and database access methods. It is understood that changes may be made to the embodiments described above without departing from the broad inventive concepts thereof. Accordingly, the present invention is not limited to the particular embodiments disclosed, but is intended to cover all modifications that are within the spirit and scope of the invention as defined by the appended claims.

[0379] As is apparent from the above, all or portions of the various systems, methods, and aspects of the present invention may be embodied in the form of program code (i.e., instructions). This program code may be stored on a computer-readable medium, such as a magnetic, electrical, or optical storage medium, including without limitation a floppy diskette, CD-ROM, CD-RW, DVD-ROM, DVD-RAM, magnetic tape, flash memory, hard disk drive, or any other machine-readable storage medium, wherein, when the program code is loaded into and executed by a machine, such as a computer or server, the machine becomes an apparatus for practicing the invention. The present invention may also be embodied in the form of program code that is transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, over a network, including the Internet or an intranet, or via any other form of transmission, wherein, when the program code is received and loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the

invention. When implemented on a general-purpose processor, the program code combines with the processor to provide a unique apparatus that operates analogously to specific logic circuits.

What is Claimed:

1. A method for extending an Item, said Item constituting a discrete storable unit of information that can be manipulated by a hardware/software interface system, said method comprising the utilization of a strongly typed instance (an "Extension") to extend said Item, said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system.

2. The method of claim 1 wherein said Extension is attached to said Item.

3. The method of claim 1 wherein said Extension cannot exist independently from said Item, such that if said Item ceases to exist, said Extension also ceases to exist.

4. The method of claim 1 wherein said Item is extended by a plurality of Extensions.

5. The method of claim 4 wherein said plurality of Extensions is used to model overlapping type instances.

6. A method for extending a Property, said Property constituting a complex property type that can be manipulated by a hardware/software interface system, said method comprising the utilization of a strongly typed instance (an "Extension") to extend said Property, said extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system and which is associated with said Property.

7. The method of claim 6 wherein said Extension is attached to said Property.

8. The method of claim 6 wherein said Extension cannot exist independently from said Property, such that if said Property ceases to exist, said Extension also ceases to exist.

9. The method of claim 6 wherein said Property is extended by a plurality of Extensions.

10. The method of claim 9 wherein said plurality of Extensions is used to model overlapping type instances.

11. A method for a hardware/software interface system to organize and efficiently query a plurality of Items, said Item constituting discretely storable units of information that can be manipulated by a hardware/software interface system, said plurality of Items comprising a first Relationship that relates a first Item and a second Item, said method comprising, for the instantiation of a third Item, said third Item being a subtyped instance of said first Item, said third Item automatically inherits from said first Item a Relationship with said second Item.

12. The method of claim 11 wherein, for the instantiation of a fourth Item, said fourth Item being a subtyped instance of said second Item, said fourth Item automatically inherits from said second Item a Relationship with said first Item.

13. The method of claim 12 wherein said fourth Item further automatically inherits from said second Item a Relationship with said third Item.

14. The method of claim 11 wherein, for each of a first plurality of subtyped instances of said first Item, each of said first plurality of subtyped instances automatically inherits from said first Item a Relationship with said second Item.

15. The method of claim 14 wherein, for each of a second plurality of subtyped instances of said second Item, said second plurality of subtyped instances further automatically inherit from said second Item Relationships with said first Item.

16. The method of claim 15 wherein each of said first plurality of subtyped instances automatically inherits from said first Item a Relationship to each of said second plurality of subtyped instances.

17. The method of claim 16 wherein each of said second plurality of subtyped instances automatically inherits from said second Item a Relationship to each of said first plurality of subtyped instances.

18. A hardware/software interface system for manipulating a plurality of Items, wherein an Item constitutes a discrete storable unit of information that can be manipulated by said hardware/software interface system, said system comprising a subsystem for extending an Item with a strongly typed instance (an "Extension"), said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system.

19. The system of claim 18 wherein said Extension is attached to said Item.

20. The system of claim 18 wherein said Extension cannot exist independently from said Item, such that if said Item ceases to exist, said Extension also ceases to exist.

21. The system of claim 18 wherein said Item is extended by a plurality of Extensions.

22. A hardware/software interface system for manipulating a plurality of Properties, said Properties constituting complex property types that can be manipulated by a hardware/software interface system, said system comprising a subsystem for extending an Property with a strongly typed instance (an "Extension"), said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system.

23. The system of claim 22 wherein said Extension is attached to said Property.

24. The system of claim 22 wherein said Extension cannot exist independently from said Property, such that if said Property ceases to exist, said Extension also ceases to exist.

25. The system of claim 22 wherein said Property is extended by a plurality of Extensions.

26. A hardware/software interface system for manipulating a plurality of Items, wherein an Item constitutes a discrete storable unit of information that can be manipulated by said hardware/software interface system, said system comprising a subsystem for organizing and efficiently querying said plurality of Items, said plurality of Items comprising a first Relationship that relates a first Item and a second Item, wherein said subsystem:

for the instantiation of a third Item, said third Item being a subtyped instance of said first Item, automatically establishes an relationship between said third Item and said second Item;

for the instantiation of a fourth Item, said fourth Item being a subtyped instance of said second Item, automatically establishes an relationship between said fourth Item and said first Item; and

automatically establishes an relationship between said fourth Item and said first Item.

27. The system of claim 26 wherein, for each of a first plurality of subtyped instances of said first Item, and for each of a second plurality of subtyped instances of said second Item, said subsystem:

automatically establishes a relationship for each of said first plurality of subtyped instances with said second Item;

automatically establishes a relationship for each of said second plurality of subtyped instances with said first Item; and

automatically establishes a relationship for each of said first plurality of subtyped instances with each of said second plurality of subtyped instances.

28. A hardware/software interface system for manipulating a plurality of Items, wherein an Item constitutes a discrete storable unit of information that can be manipulated by said hardware/software interface system, said system comprising a subsystem for extending an Item with a strongly typed instance (an "Extension"), said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system.

29. The system of claim 28 wherein said Extension is attached to said Item.

30. The system of claim 28 wherein said Extension cannot exist independently from said Item, such that if said Item ceases to exist, said Extension also ceases to exist.

31. The system of claim 28 wherein said Item is extended by a plurality of Extensions.

32. A hardware/software interface system for manipulating a plurality of Properties, said Properties constituting complex property types that can be manipulated by a hardware/software interface system, said system comprising a subsystem for extending an Property with a strongly typed instance (an "Extension"), said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system.

33. The system of claim 32 wherein said Extension is attached to said Property.

34. The system of claim 32 wherein said Extension cannot exist independently from said Property, such that if said Property ceases to exist, said Extension also ceases to exist.

35. The system of claim 32 wherein said Property is extended by a plurality of Extensions.

36. A hardware/software interface system for manipulating a plurality of Items, wherein an Item constitutes a discrete storable unit of information that can be manipulated by said hardware/software interface system, said system comprising a subsystem for organizing and efficiently querying said plurality of Items, said plurality of Items comprising a first Relationship that relates a first Item and a second Item, wherein said subsystem:

for the instantiation of a third Item, said third Item being a subtyped instance of said first Item, automatically establishes an relationship between said third Item and said second Item;

for the instantiation of a fourth Item, said fourth Item being a subtyped instance of said second Item, automatically establishes an relationship between said fourth Item and said first Item; and

automatically establishes an relationship between said fourth Item and said first Item.

37. The system of claim 36 wherein, for each of a first plurality of subtyped instances of said first Item, and for each of a second plurality of subtyped instances of said second Item, said subsystem:

automatically establishes a relationship for each of said first plurality of subtyped instances with said second Item;

automatically establishes a relationship for each of said second plurality of subtyped instances with said first Item; and

automatically establishes a relationship for each of said first plurality of subtyped instances with each of said second plurality of subtyped instances.

38. A computer-readable medium comprising computer-readable instructions for extending an Item, said Item constituting a discrete storable unit of information that can be manipulated by a hardware/software interface system, said computer-readable instructions comprising instructions for the utilization of a strongly typed instance (an "Extension") to extend said Item, said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system wherein said Extension is attached to said Item and wherein said Extension also ceases to exist when said Item ceases to exist.

39. A computer-readable medium comprising computer-readable instructions for extending an Property, said Property constituting a complex property type that can be manipulated by a hardware/software interface system, said computer-readable instructions comprising instructions for the utilization of a strongly typed instance (an "Extension") to extend said Property, said Extension constituting a discrete storable unit of information that can be manipulated by said hardware/software interface system wherein said Extension is attached to said Property and wherein said Extension also ceases to exist when said Property ceases to exist.

40. A computer-readable medium comprising computer-readable instructions for organizing and efficiently querying a plurality of Items, said Item constituting discretely storable units of

information that can be manipulated by a hardware/software interface system, said computer-readable instructions comprising instructions for:

instantiating a first Item, a second Item, and a first Relationship that relates a first Item and a second Item;

instantiating a third Item, said third Item being a subtyped instance of said first Item; and

automatically establishing an inherited Relationship between said third Item and said second Item.

41. The computer-readable medium of claim 40 further comprising instructions for:

instantiating a fourth Item, said fourth Item being a subtyped instance of said second Item; and

automatically establishing an inherited Relationship between said fourth Item and said first Item.

42. The computer-readable medium of claim 41 further comprising instructions for automatically establishing an inherited Relationship between said third Item and said fourth Item.

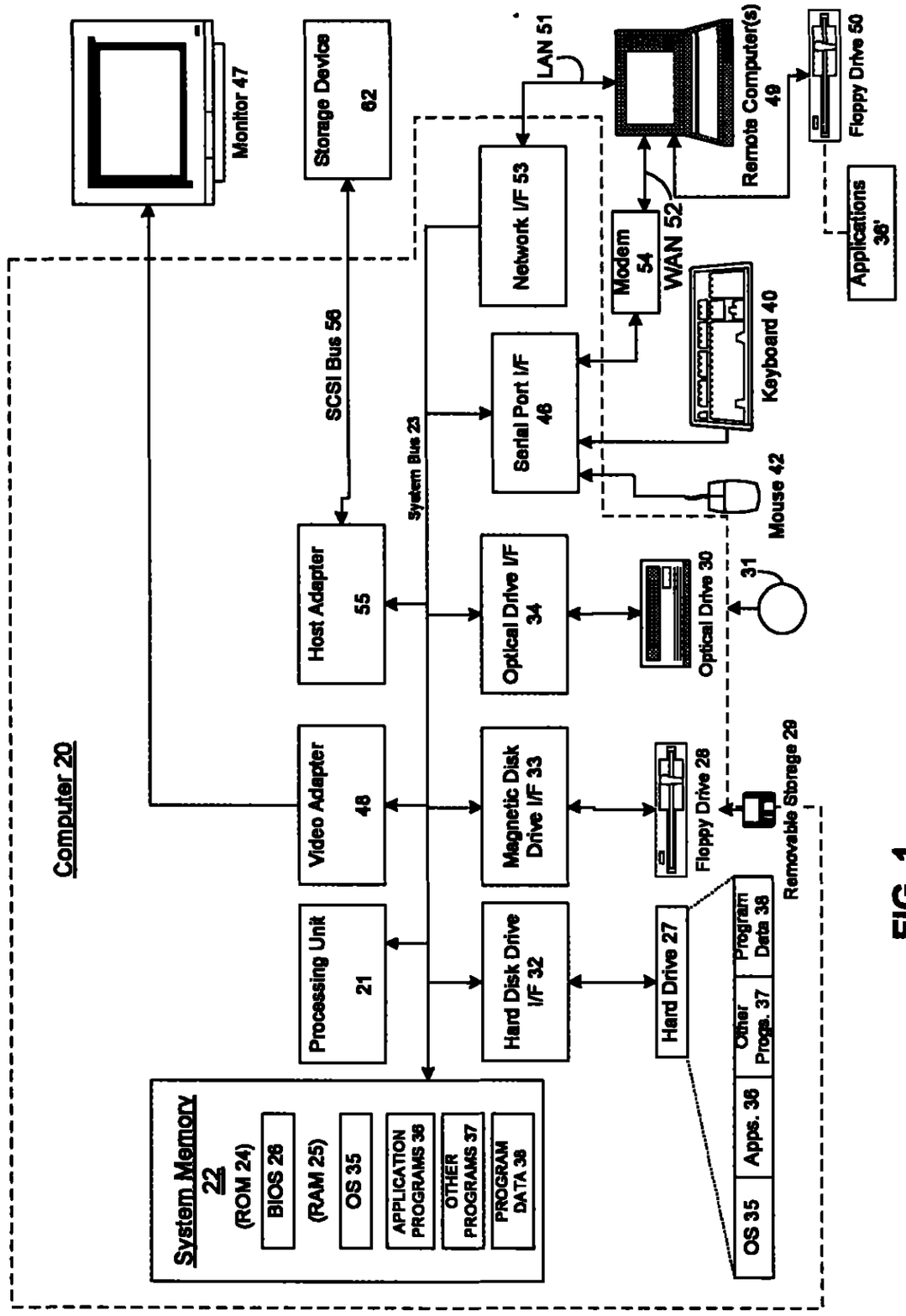

Computer 20
Monitor 47
Storage Device 62
LAN 51
Remote Computer(s) 49
Floppy Drive 50
Applications 36'
Network I/F 53
Modem 54
WAN 52
SCSI Bus 56
System Bus 23
Serial Port I/F 46
Keyboard 40
Mouse 42
Host Adapter 55
Optical Drive I/F 34
Optical Drive 30
31
Video Adapter 48
Magnetic Disk Drive I/F 33
Floppy Drive 28
Removable Storage 29
Processing Unit 21
Hard Disk Drive I/F 32
Hard Drive 27
Program Data 38
Other Progs. 37
Apps. 36
OS 35
System Memory 22
(ROM 24)
BIOS 26
(RAM 25)
OS 35
APPLICATION PROGRAMS 36
OTHER PROGRAMS 37
PROGRAM DATA 38

FIG. 1

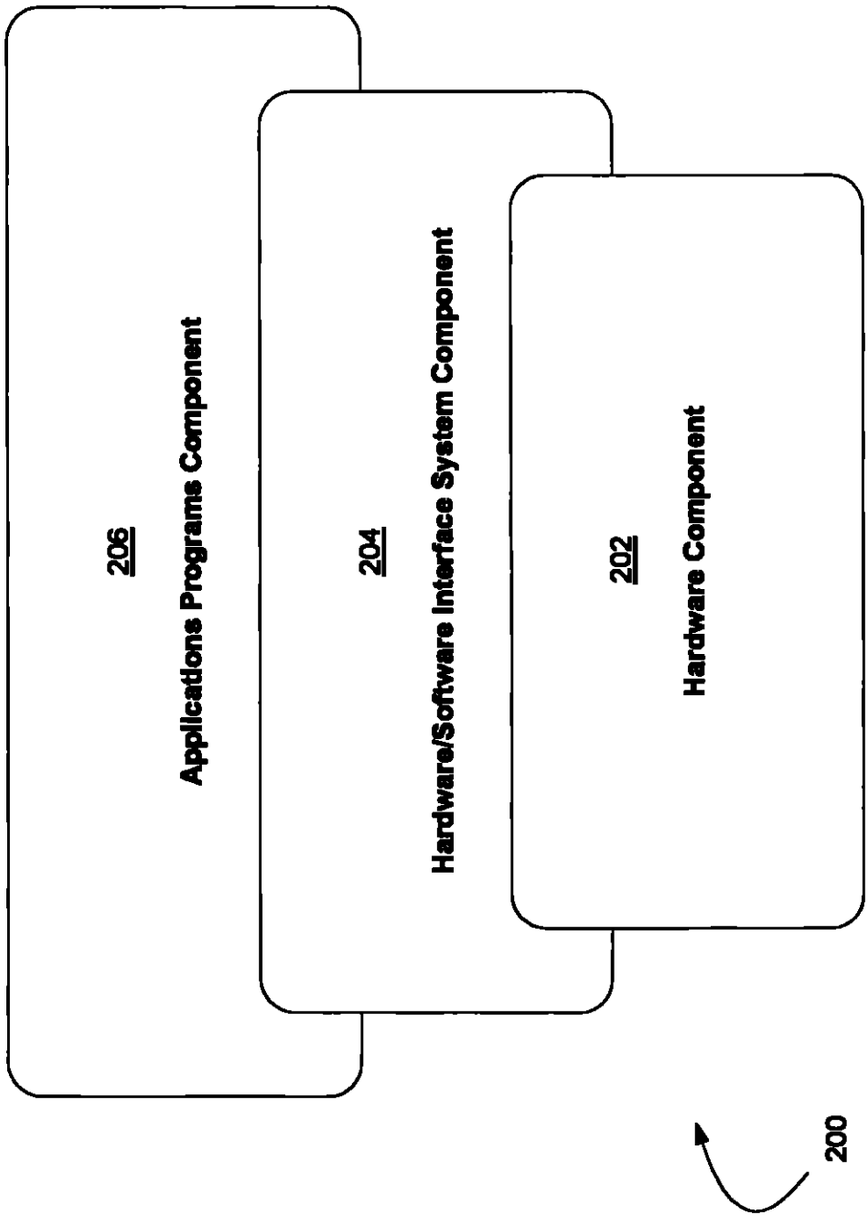

206
Applications Programs Component
204
Hardware/Software Interface System Component
202
Hardware Component
200

FIG. 2

FIG. 2A

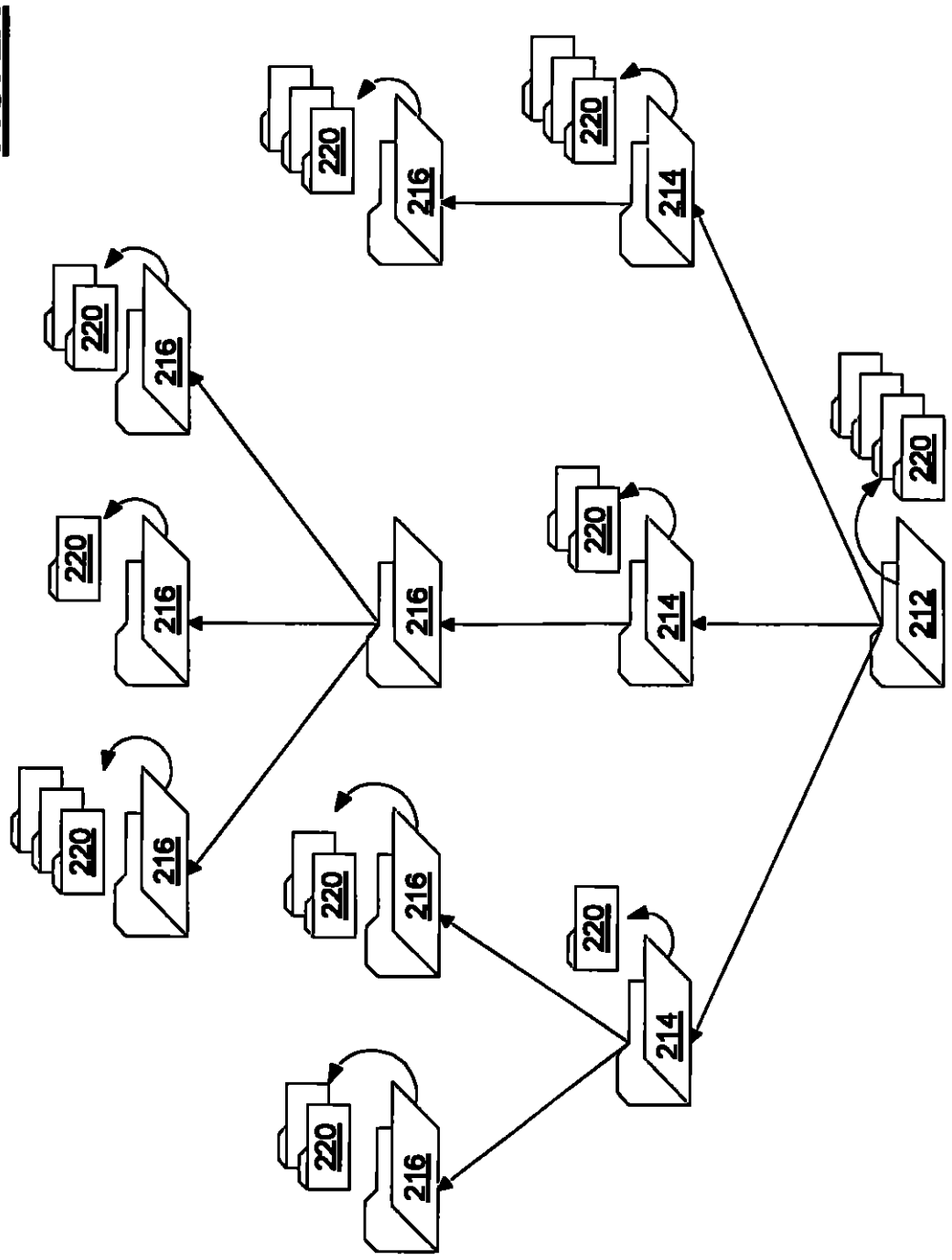
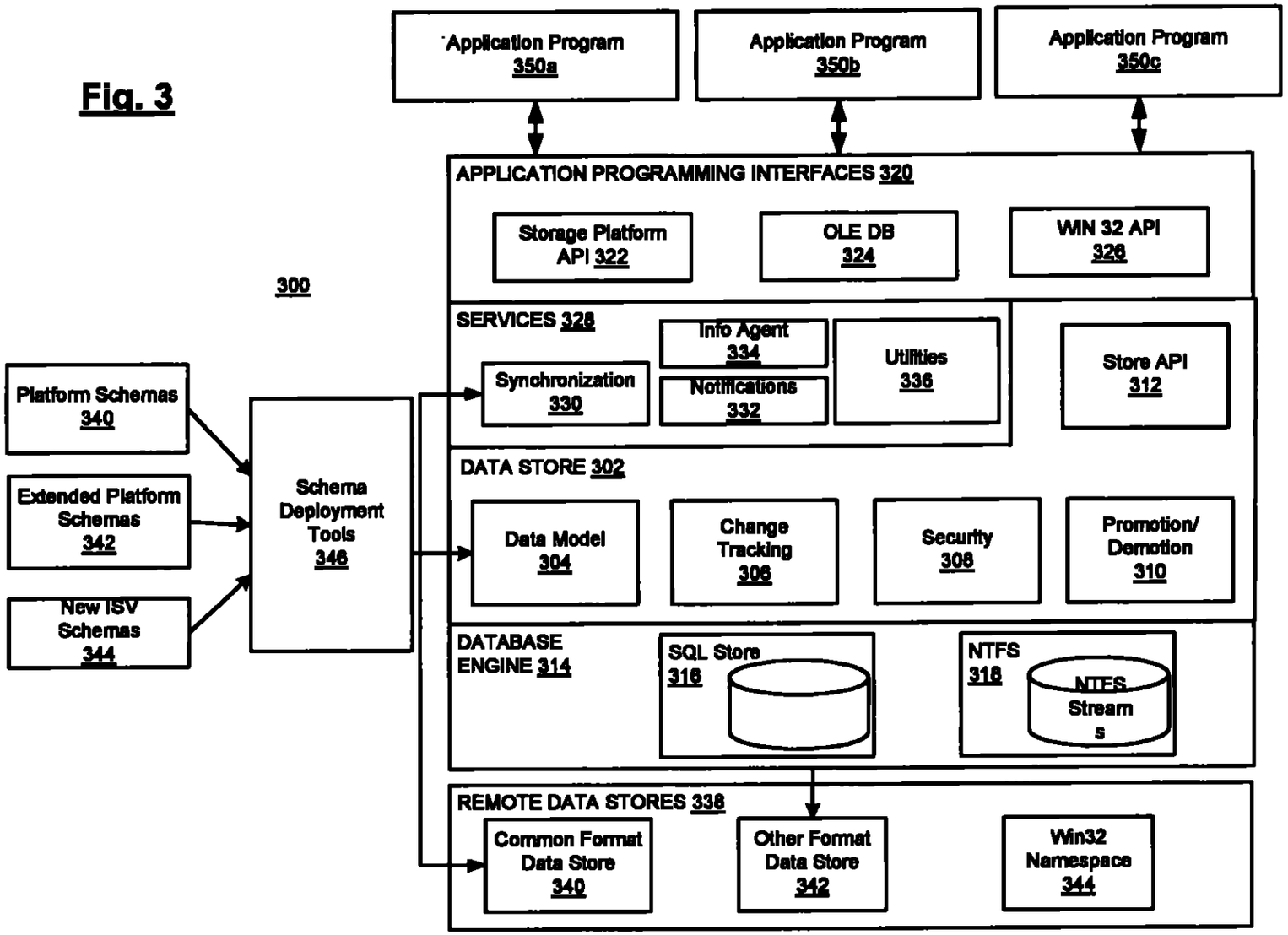


Fig. 3


Application Program 350a
Application Program 350b
Application Program 350c
APPLICATION PROGRAMMING INTERFACES 320
Storage Platform API 322
OLE DB 324
WIN 32 API 326
300
SERVICES 328
Info Agent 334
Synchronization 330
Notifications 332
Utilities 336
Store API 312
Platform Schemas 340
Extended Platform Schemas 342
New ISV Schemas 344
Schema Deployment Tools 346
DATA STORE 302
Data Model 304
Change Tracking 306
Security 308
Promotion/ Demotion 310
DATABASE ENGINE 314
SQL Store 316
NTFS 318
NTFS Streams
REMOTE DATA STORES 338
Common Format Data Store 340
Other Format Data Store 342
Win32 Namespace 344

334

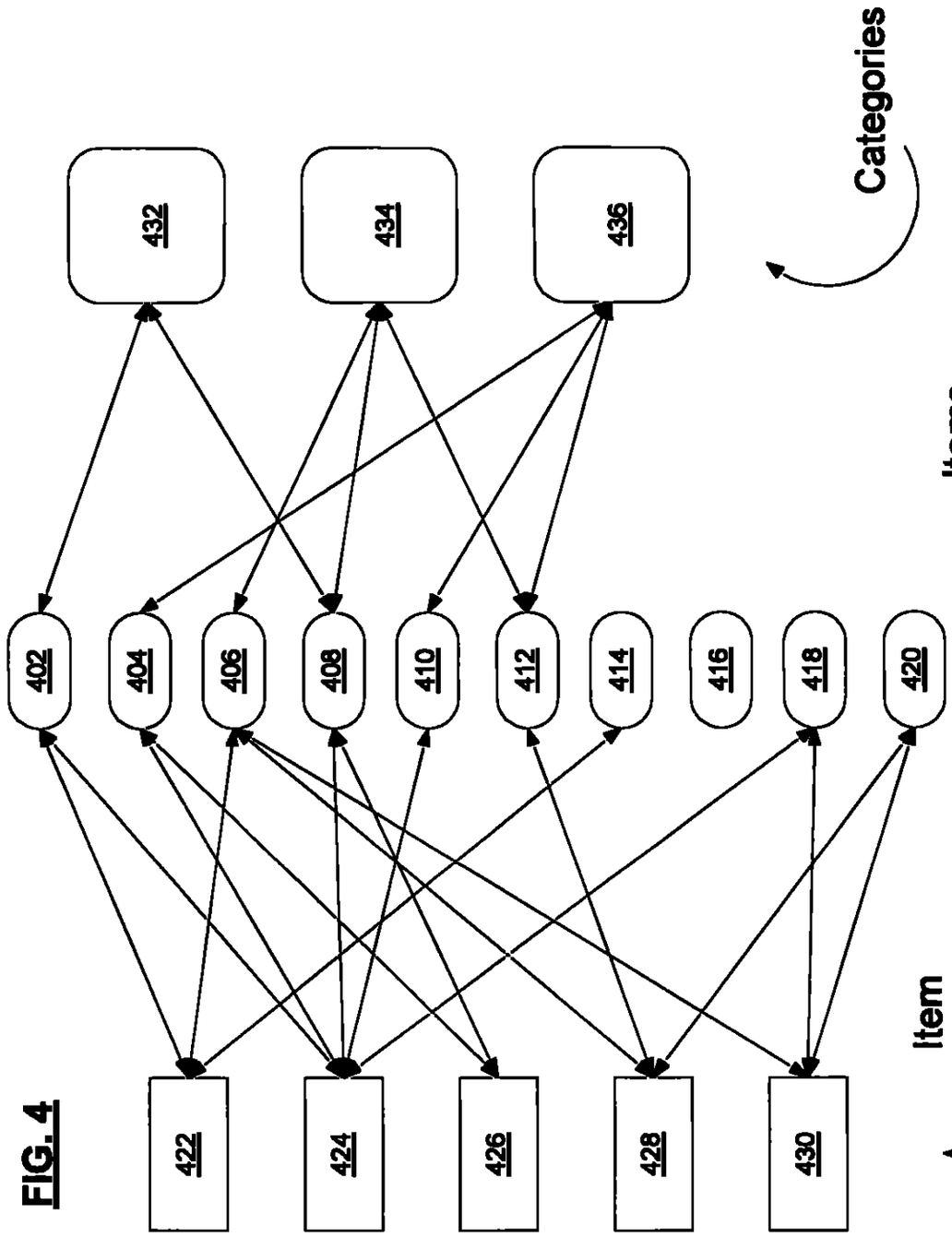

Categories
Items
Item Fold rs
432
434
436
402
404
406
408
410
412
414
416
418
420
422
424
426
428
430

FIG. 4

FIG. 5A

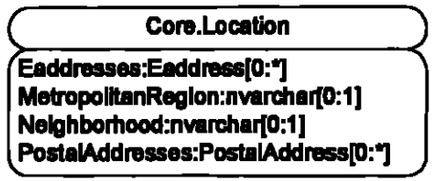


FIG. 5B

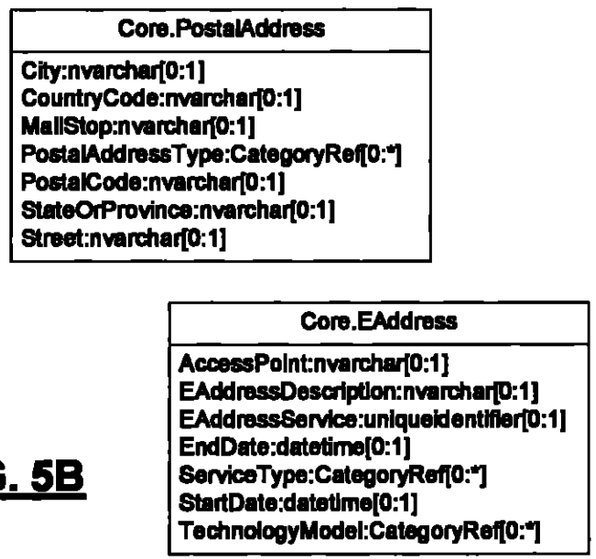
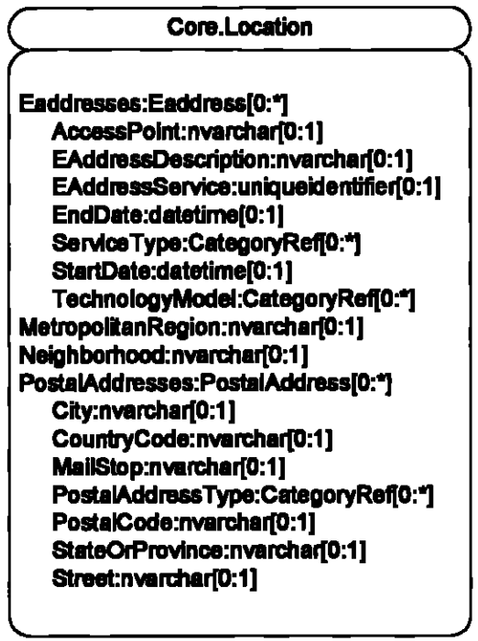
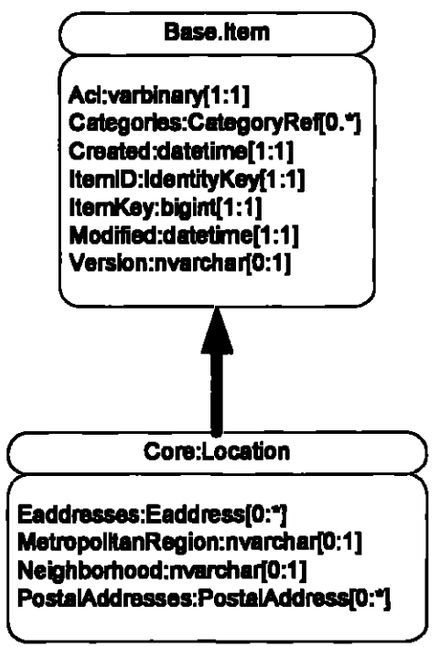


FIG. 5C



336

FIG. 6A


Base.Item
Acl:varbinary[1:1]
Categories:CategoryRef[0.*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Core:Location
Eaddresses:Eaddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
PostalAddresses:PostalAddress[0:*]

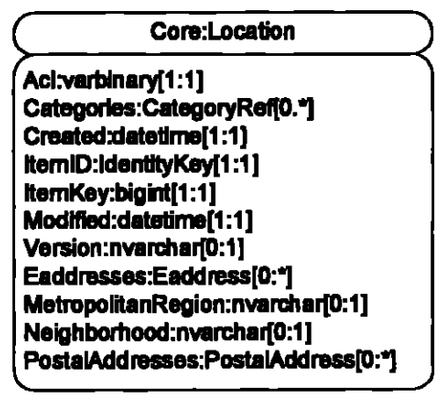

Core:Location
Acl:varbinary[1:1]
Categories:CategoryRef[0.*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Eaddresses:Eaddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
PostalAddresses:PostalAddress[0:*]

FIG. 6B

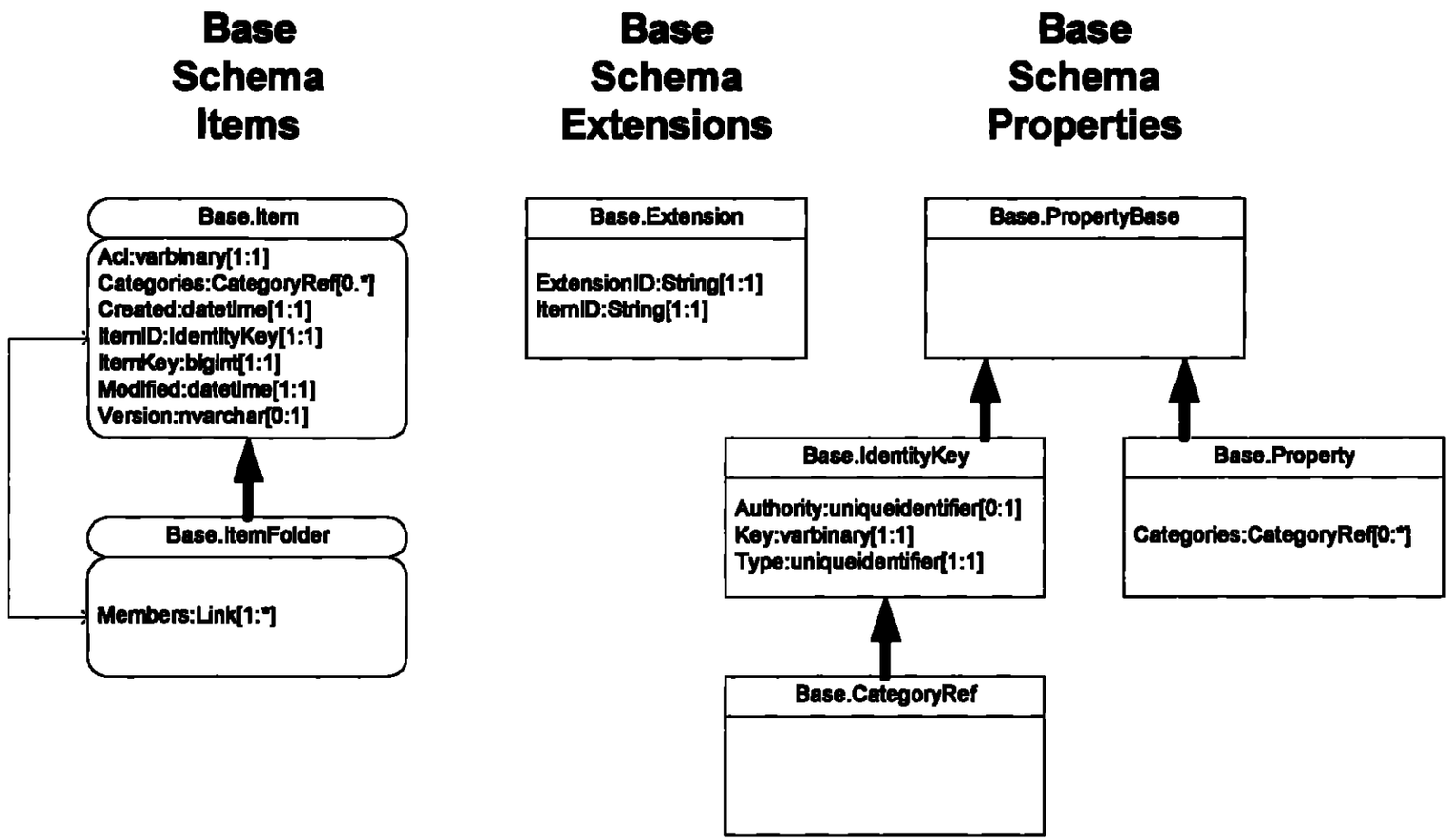
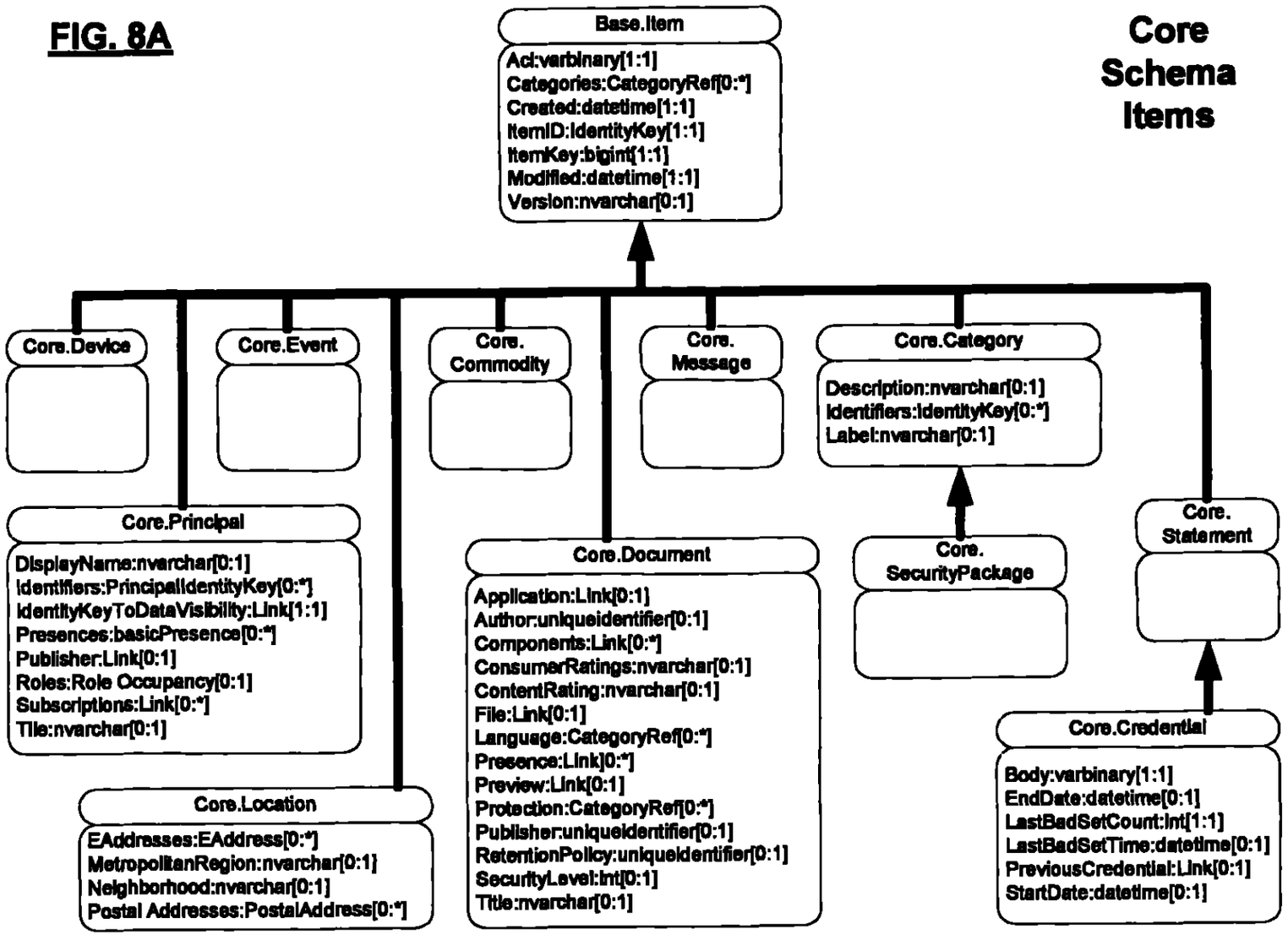

Base Schema Items
Base Schema Extensions
Base Schema Properties
Base.Item
Acl:varbinary[1:1]
Categories:CategoryRef[0.*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Base.ItemFolder
Members:Link[1:*]
Base.Extension
ExtensionID:String[1:1]
ItemID:String[1:1]
Base.PropertyBase
Base.IdentityKey
Authority:uniqueidentifier[0:1]
Key:varbinary[1:1]
Type:uniqueidentifier[1:1]
Base.Property
Categories:CategoryRef[0:*]
Base.CategoryRef

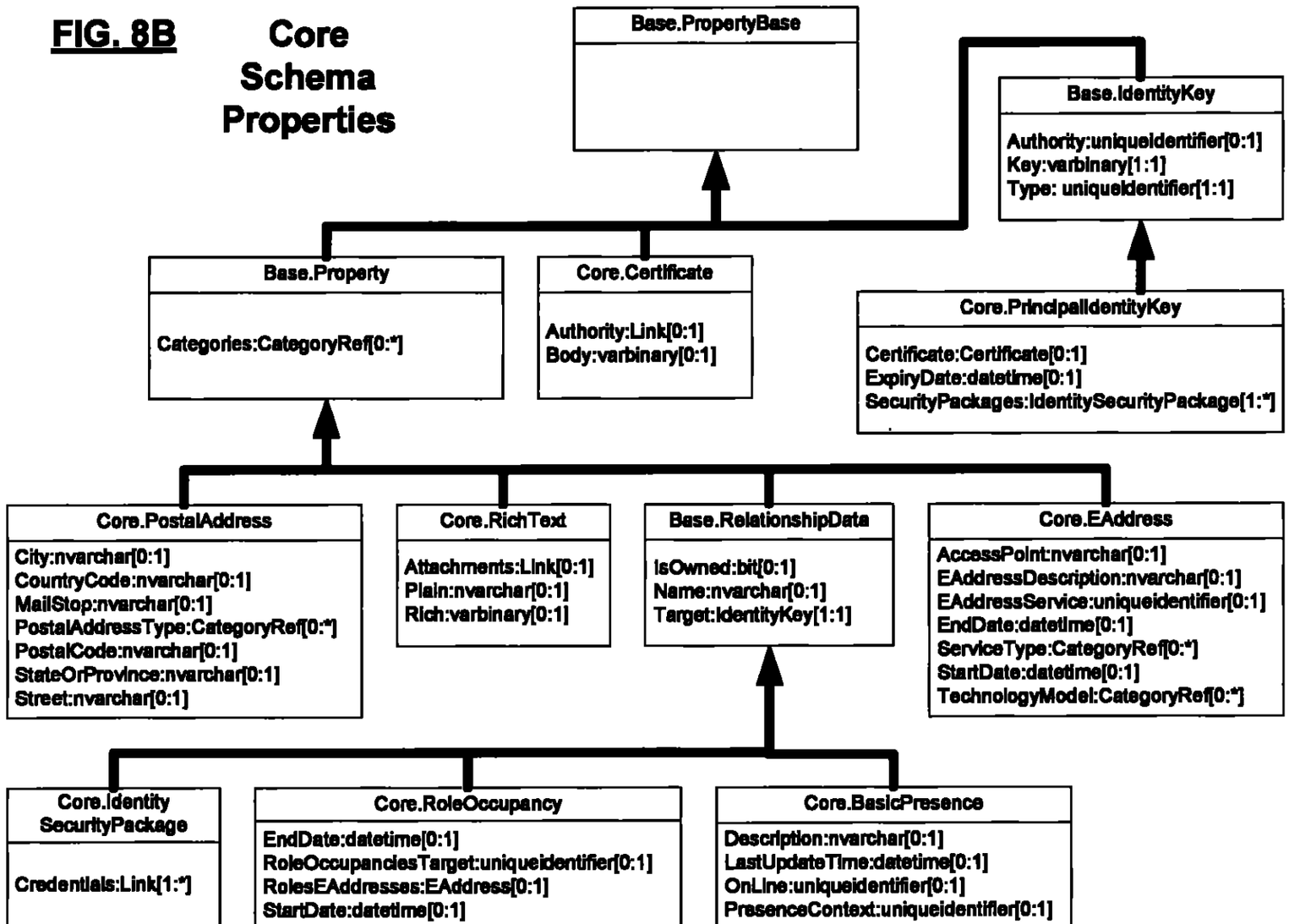
FIG. 7

FIG. 8A


Base.Item
Acl:varbinary[1:1]
Categories:CategoryRef[0:*]
Created:datetime[1:1]
ItemID:IdentityKey[1:1]
ItemKey:bigint[1:1]
Modified:datetime[1:1]
Version:nvarchar[0:1]
Core Schema Items
Core.Device
Core.Event
Core. Commodity
Core. Message
Core.Category
Description:nvarchar[0:1]
Identifiers:IdentityKey[0:*]
Label:nvarchar[0:1]
Core.Principal
DisplayName:nvarchar[0:1]
Identifiers:PrincipalIdentityKey[0:*]
IdentityKeyToDataVisibility:Link[1:1]
Presences:basicPresence[0:*]
Publisher:Link[0:1]
Roles:Role Occupancy[0:1]
Subscriptions:Link[0:*]
Title:nvarchar[0:1]
Core.Document
Application:Link[0:1]
Author:uniqueidentifier[0:1]
Components:Link[0:*]
ConsumerRatings:nvarchar[0:1]
ContentRating:nvarchar[0:1]
File:Link[0:1]
Language:CategoryRef[0:*]
Presence:Link]0:*]
Preview:Link[0:1]
Protection:CategoryRef[0:*]
Publisher:uniqueidentifier[0:1]
RetentionPolicy:uniqueidentifier[0:1]
SecurityLevel:Int[0:1]
Title:nvarchar[0:1]
Core. SecurityPackage
Core. Statement
Core.Location
EAddresses:EAddress[0:*]
MetropolitanRegion:nvarchar[0:1]
Neighborhood:nvarchar[0:1]
Postal Addresses:PostalAddress[0:*]
Core.Credential
Body:varbinary[1:1]
EndDate:datetime[0:1]
LastBadSetCount:Int[1:1]
LastBadSetTime:datetime[0:1]
PreviousCredential:Link[0:1]
StartDate:datetime[0:1]

339

FIG. 8B Core Schema Properties



340

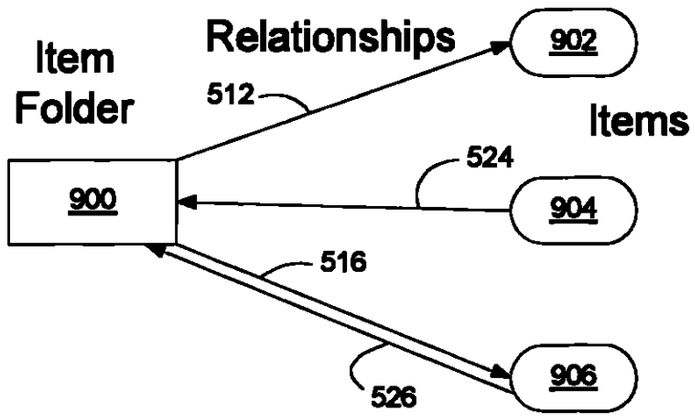
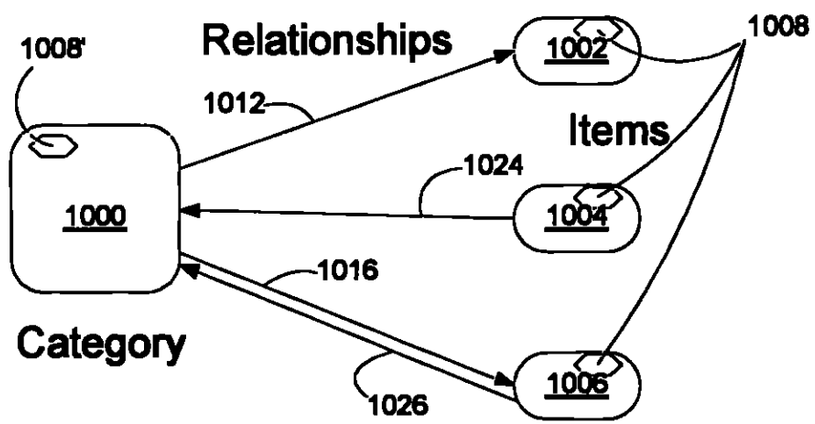

Item
Folder
Relationships
902
512
Items
524
900
904
516
906
526

FIG. 9

FIG. 10


1008'
Relationships
1002
1008
1012
Items
1024
1000
1004
1016
Category
1006
1026

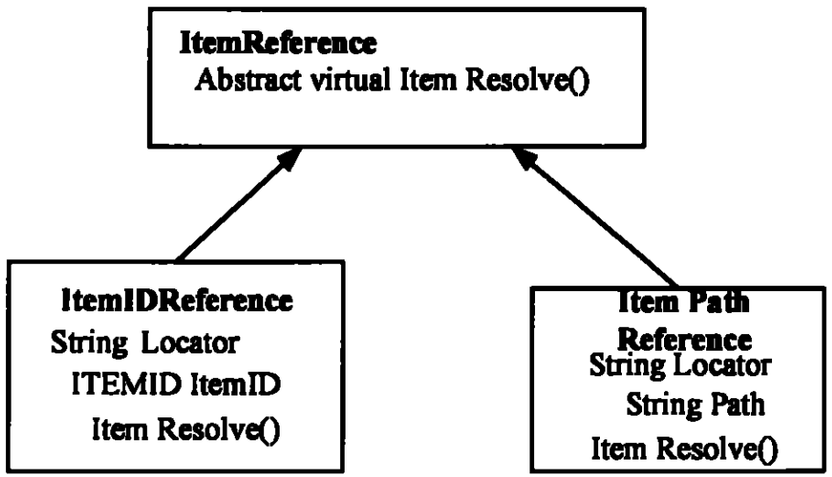


FIG. 11

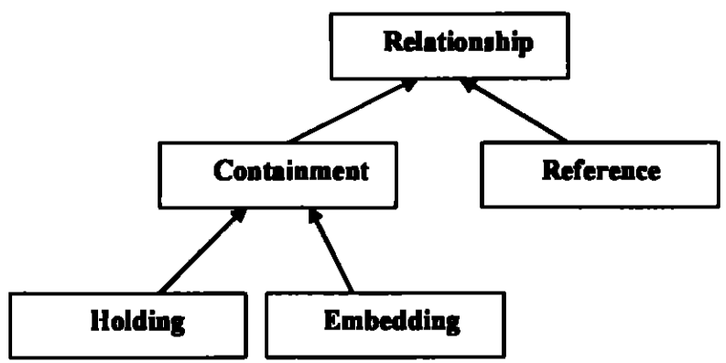


FIG. 12

342

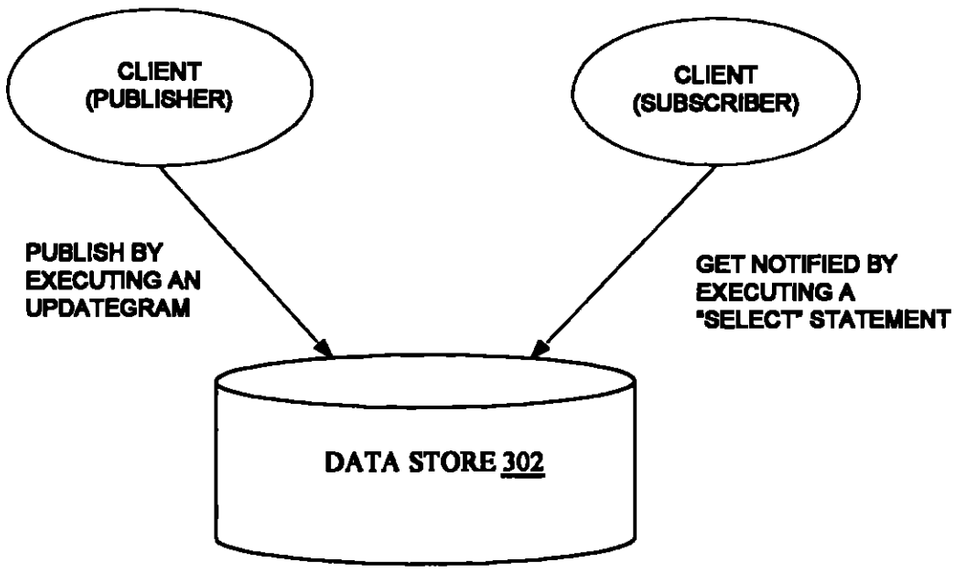

CLIENT
(PUBLISHER)
CLIENT
(SUBSCRIBER)
PUBLISH BY
EXECUTING AN
UPDATEGRAM
GET NOTIFIED BY
EXECUTING A
"SELECT" STATEMENT
DATA STORE 302

FIG. 13

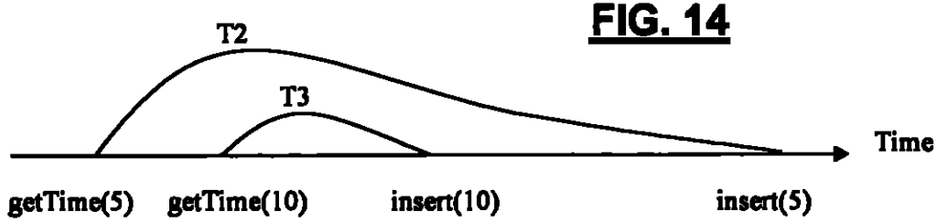

T2
T3
Time
getTime(5)
getTime(10)
insert(10)
insert(5)

FIG. 14

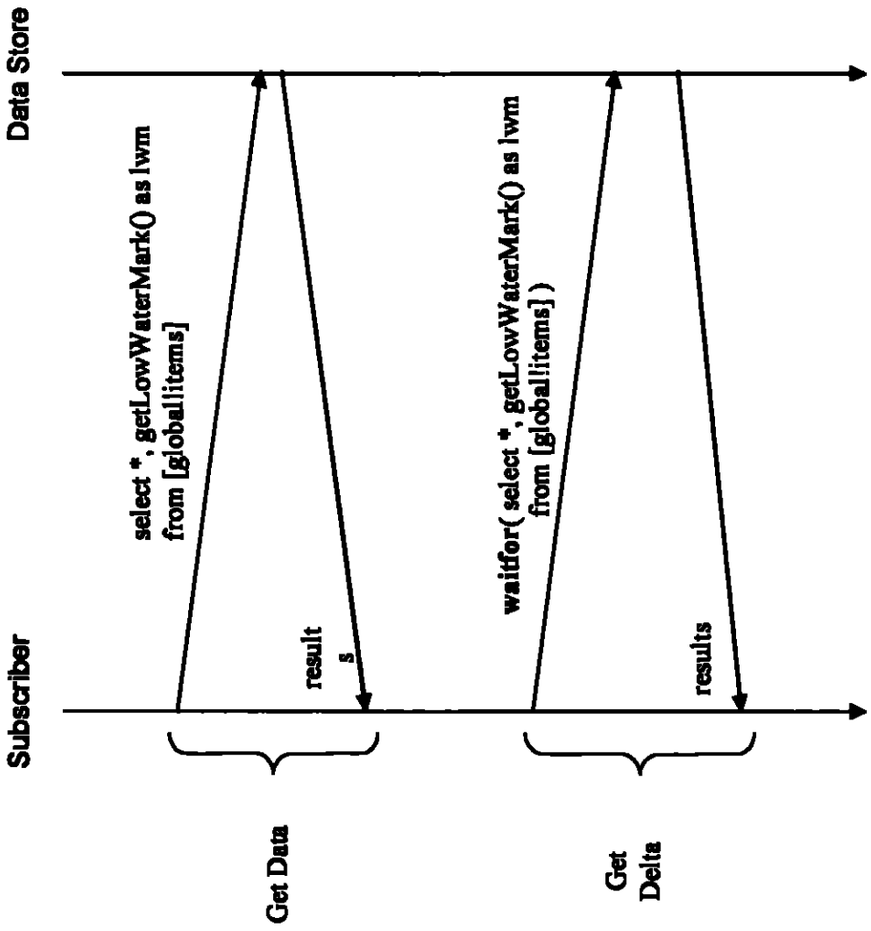

Data Store
Subscriber
select *, getLowWaterMark() as lwm
from [globalitems]
result
s
waitfor(select *, getLowWaterMark() as lwm
from [globalitems])
results
Get Data
Get
Delta

FIG. 15

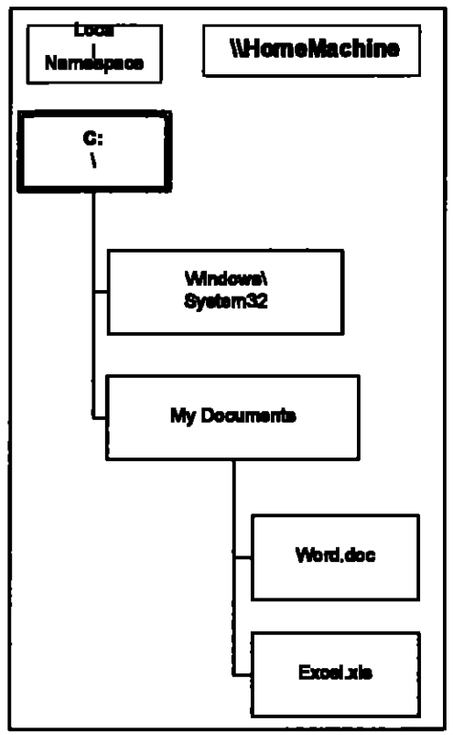

Local
Namespace
\\HomeMachine
C:
\
Windows\
System32
My Documents
Word.doc
Excel.xls

FIG. 16

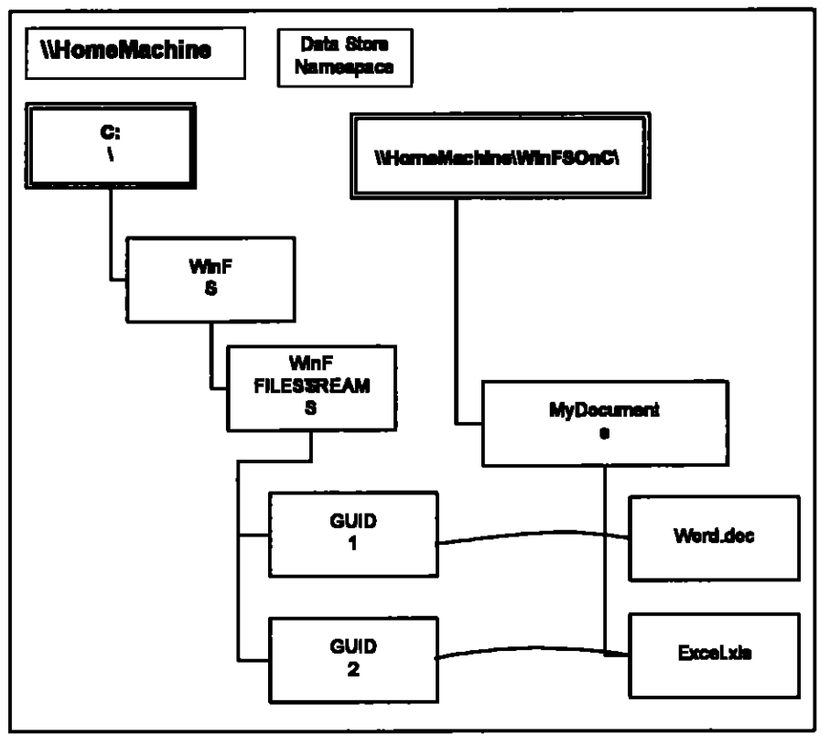

\\HomeMachine
Data Store
Namespace
C:
\
\\HomeMachine\WinFSOnC\
WinF
S
WinF
FILESTREAM
S
MyDocument
s
GUID
1
GUID
2
Word.doc
Excel.xls

FIG. 17

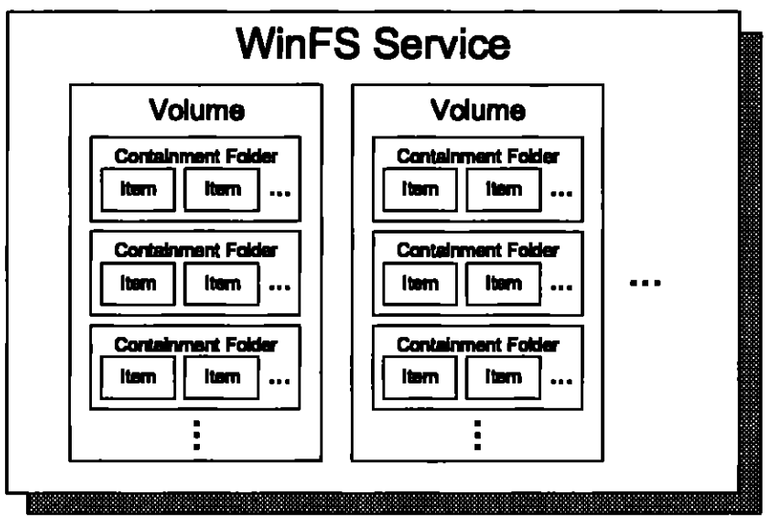

WinFS Service
Volume
Containment Folder
Item
Item
...
Containment Folder
Item
Item
...
Containment Folder
Item
Item
...
Volume
Containment Folder
Item
Item
...
Containment Folder
Item
Item
...
Containment Folder
Item
Item
...
...

FIG. 18

346

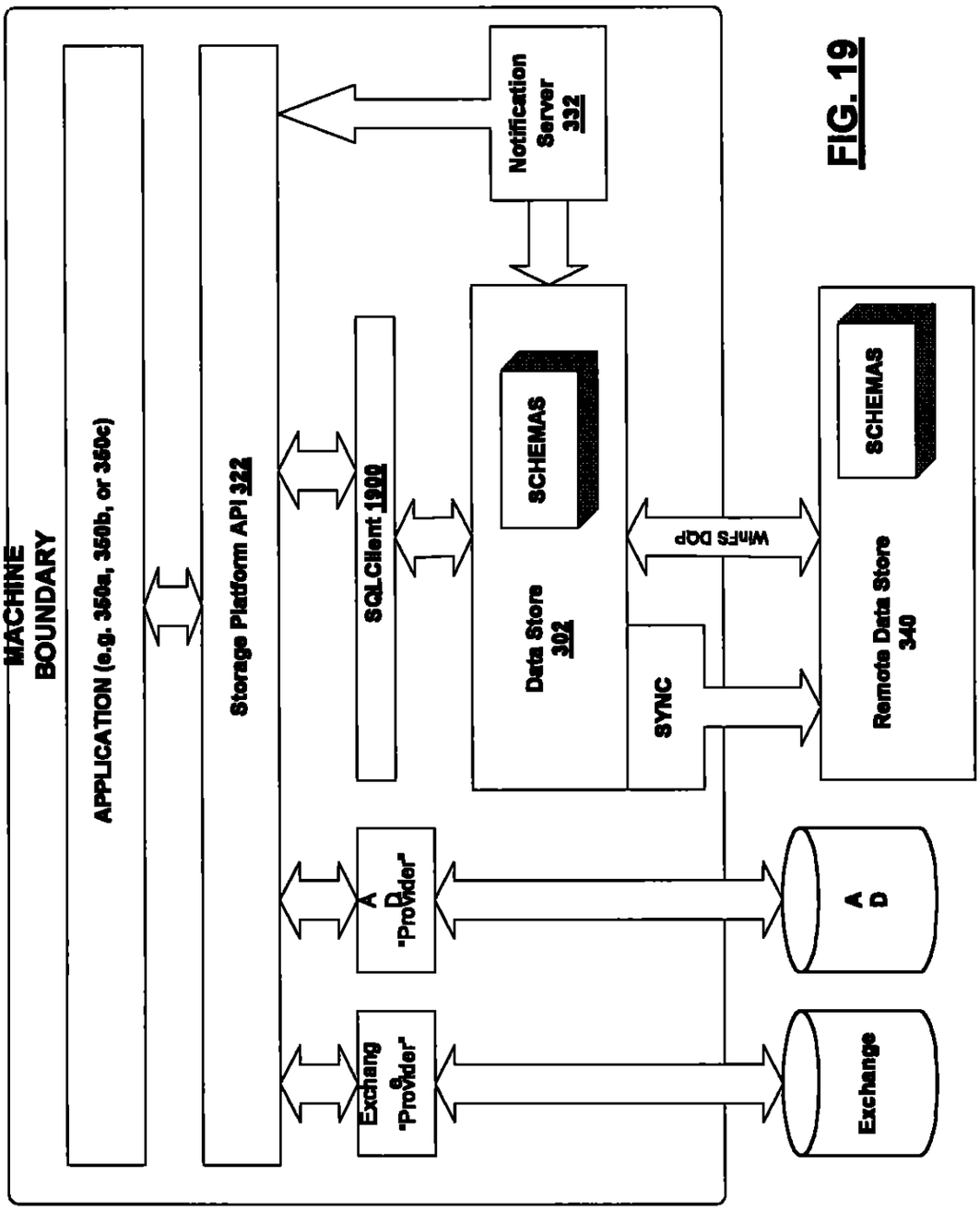

MACHINE BOUNDARY
APPLICATION (e.g. 350a, 350b, or 350c)
Storage Platform API 322
Notification Server 332
SQLClient 1900
SCHEMAS
Data Store 302
WinFS DQP
SYNC
SCHEMAS
Remote Data Store 340
AD "Provider"
Exchange "Provider"
AD
Exchange

FIG. 19

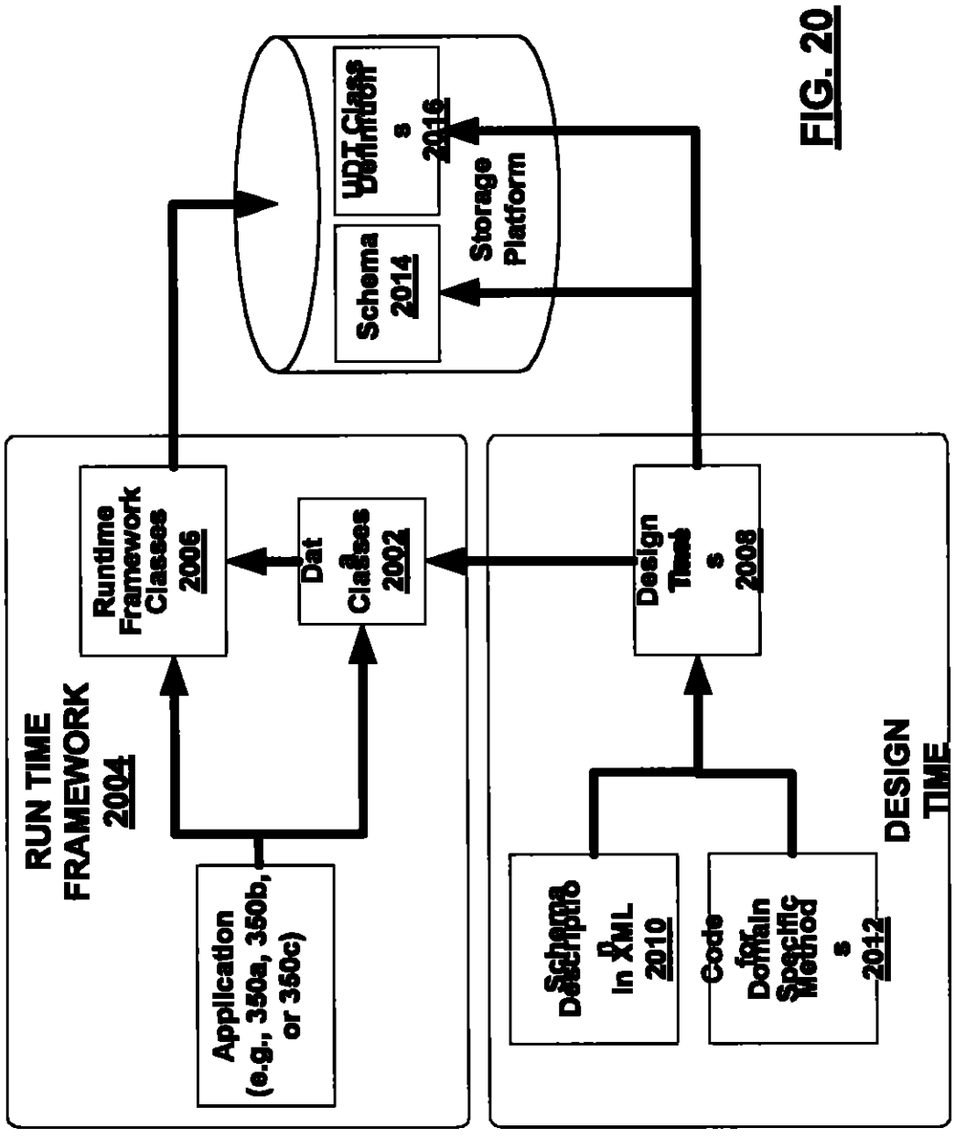

RUN TIME FRAMEWORK 2004
Application (e.g., 350a, 350b, or 350c)
Runtime Framework Classes 2006
Dat Classes 2002
DESIGN TIME
Schemas In XML 2010
Code for Domain Specific Method s 2012
Design Time s 2008
Storage Platform
Schema 2014
UDT Class Definition s 2016

FIG. 20

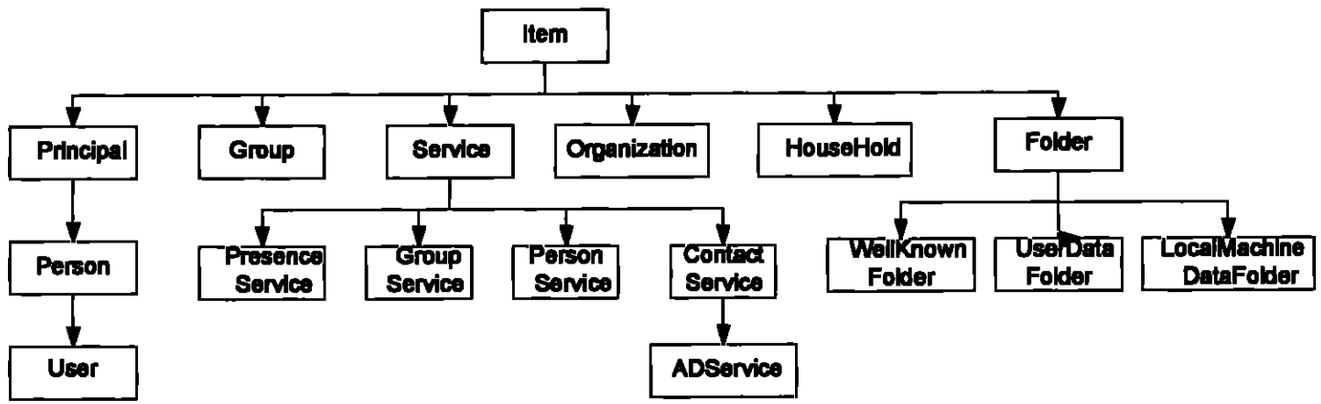

Item
Principal
Group
Service
Organization
HouseHold
Folder
Person
User
Presence Service
Group Service
Person Service
Contact Service
ADService
WellKnown Folder
UserData Folder
LocalMachine DataFolder

FIG. 21A

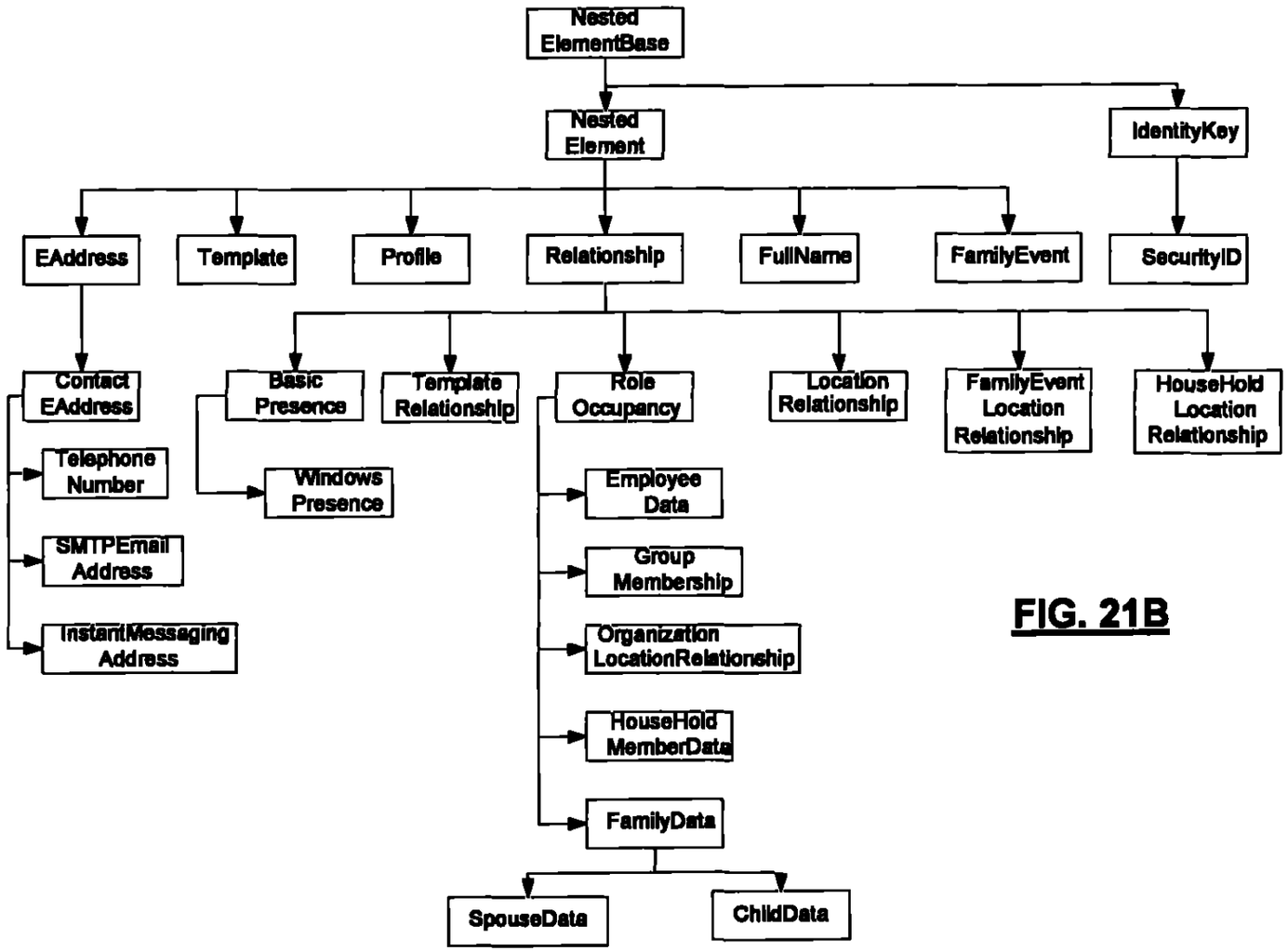

Nested ElementBase
Nested Element
IdentityKey
EAddress
Template
Profile
Relationship
FullName
FamilyEvent
SecurityID
Contact EAddress
Basic Presence
Template Relationship
Role Occupancy
Location Relationship
FamilyEvent Location Relationship
HouseHold Location Relationship
Telephone Number
Windows Presence
SMTPEmail Address
InstantMessaging Address
Employee Data
Group Membership
Organization LocationRelationship
HouseHold MemberData
FamilyData
SpouseData
ChildData

FIG. 21B

350

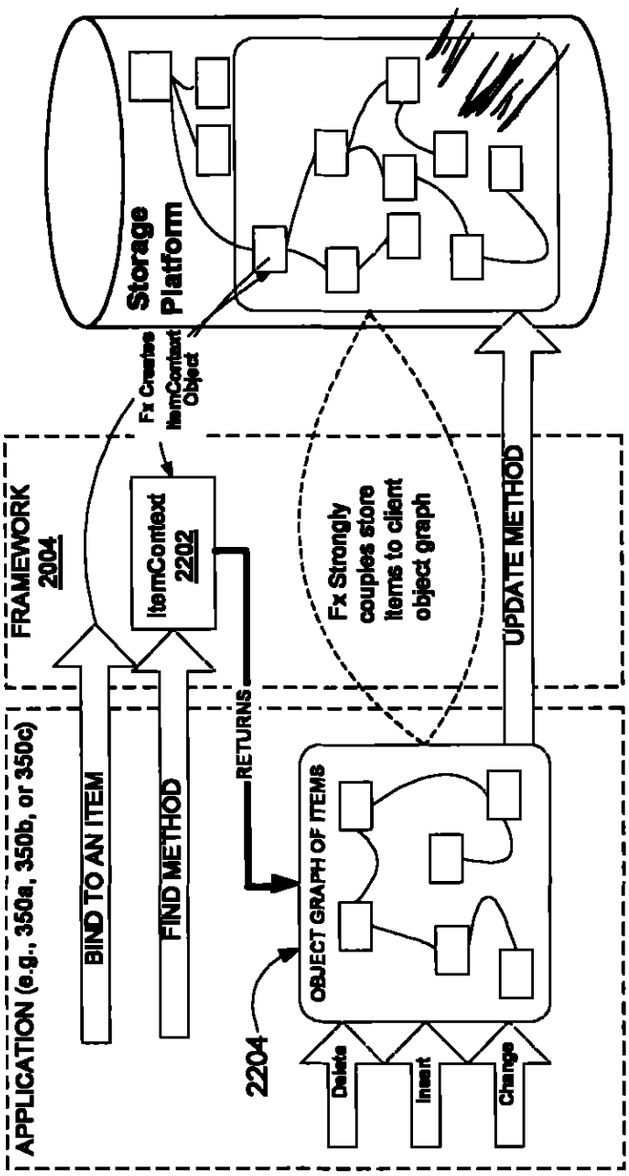

Storage Platform
Fx Creates ItemContext Object
FRAMEWORK 2004
ItemContext 2202
Fx Strongly couples store items to client object graph
UPDATE METHOD
APPLICATION (e.g., 350a, 350b, or 350c)
BIND TO AN ITEM
FIND METHOD
RETURNS
OBJECT GRAPH OF ITEMS
2204
Delete
Insert
Change

FIG. 22

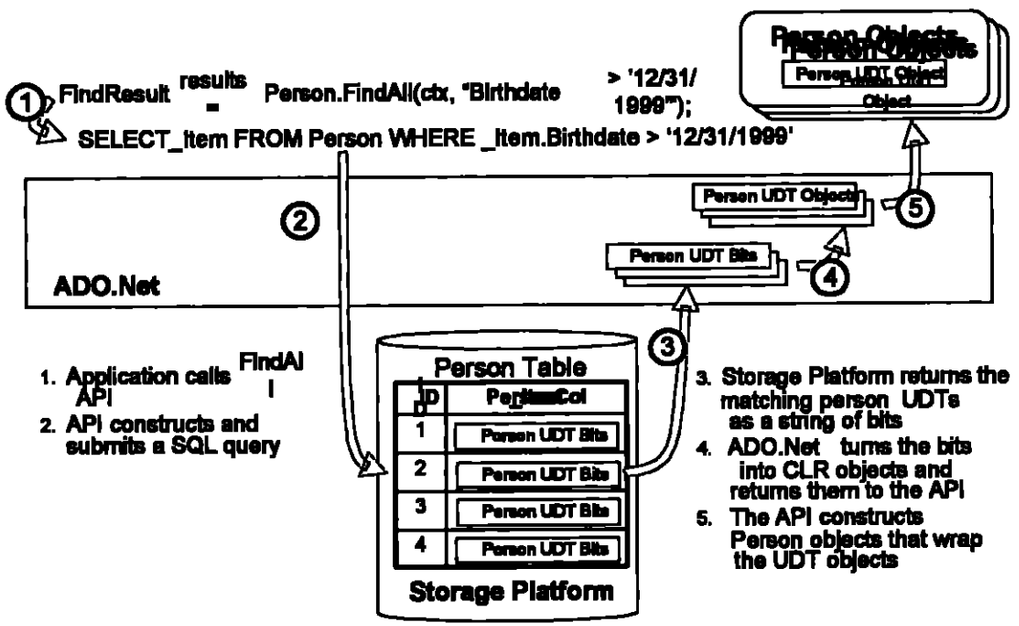

FindResult results = Person.FindAll(ctx, "Birthdate > '12/31/1999");
SELECT_Item FROM Person WHERE _Item.Birthdate > '12/31/1999'
Person Objects
Person UDT Object
Person UDT Objects
Person UDT Bits
ADO.Net
1. Application calls FindAll API
2. API constructs and submits a SQL query
Person Table
ID
PersonCol
1
2
3
4
Person UDT Bits
Storage Platform
3. Storage Platform returns the matching person UDTs as a string of bits
4. ADO.Net turns the bits into CLR objects and returns them to the API
5. The API constructs Person objects that wrap the UDT objects

FIG. 23

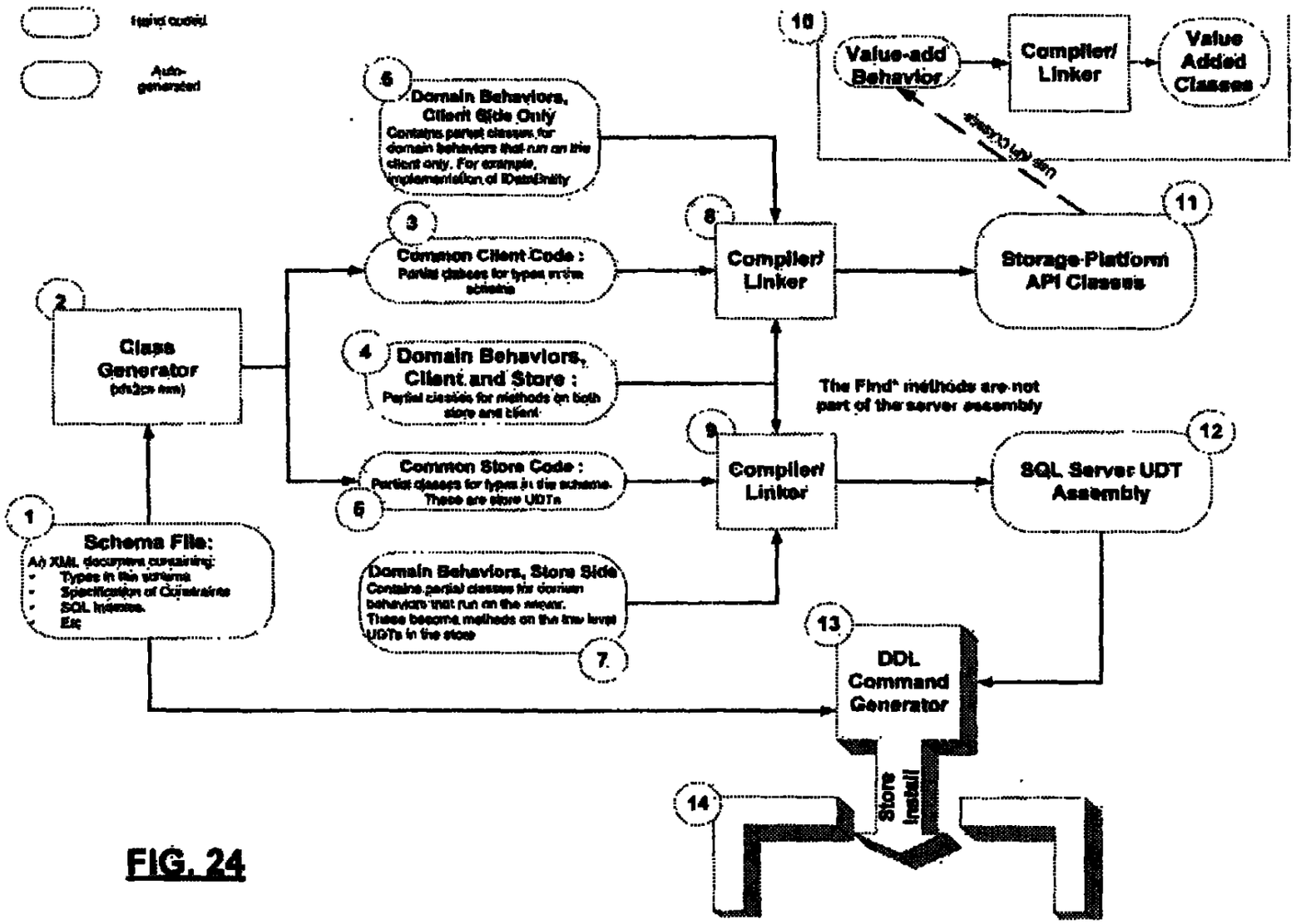
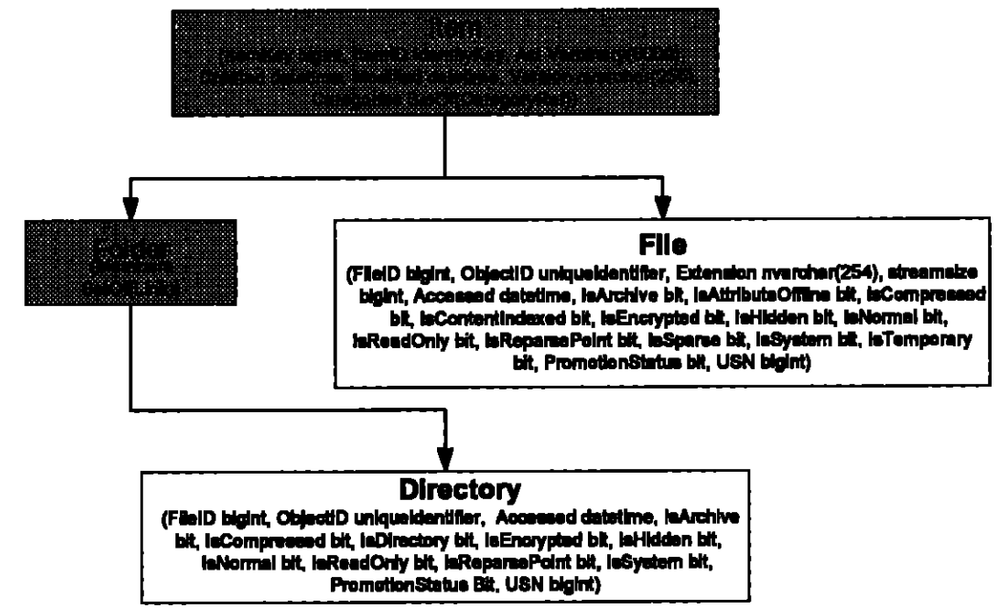

Hand-coded
Auto-generated
10
Value-add Behavior
Compiler/ Linker
Value Added Classes
use API Classes
11
5
Domain Behaviors, Client Side Only
Contains partial classes for domain behaviors that run on the client only. For example, implementation of IDataEntity
3
Common Client Code :
Partial classes for types in the schema
8
Compiler/ Linker
Storage-Platform API Classes
2
Class Generator
4
Domain Behaviors, Client and Store :
Partial classes for methods on both store and client
The Find* methods are not part of the server assembly
9
Compiler/ Linker
12
SQL Server UDT Assembly
Common Store Code :
Partial classes for types in the schema. These are store UDTs
6
1
Schema File:
An XML document containing:
- Types in the schema
- Specification of Constraints
- SQL Indexes
- Etc
Domain Behaviors, Store Side
Contains partial classes for domain behaviors that run on the server. These become methods on the new level UDTs in the store
7
13
DDL Command Generator
Store Install
14

FIG. 24



Note: Grey boxes are types defined in the Base schema, but are shown here for the sake of completeness.

Items Defined in the Files Schema

FIG. 25

354

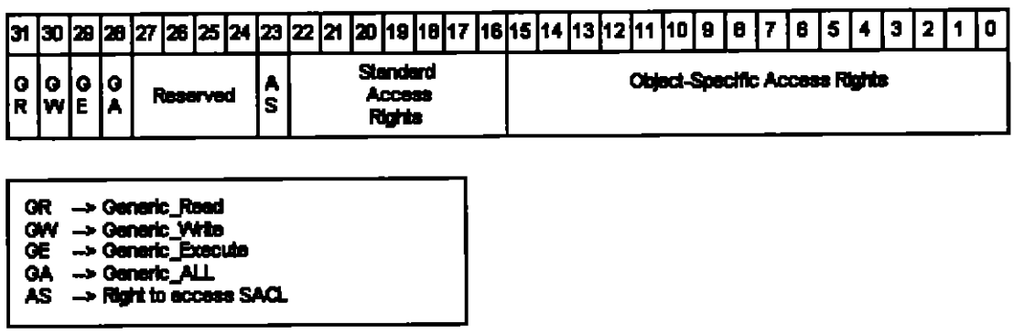

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
GR
GW
GE
GA
Reserved
AS
Standard Access Rights
Object-Specific Access Rights
GR --> Generic_Read
GW --> Generic_Write
GE --> Generic_Execute
GA --> Generic_ALL
AS --> Right to access SACL

FIG. 26

355

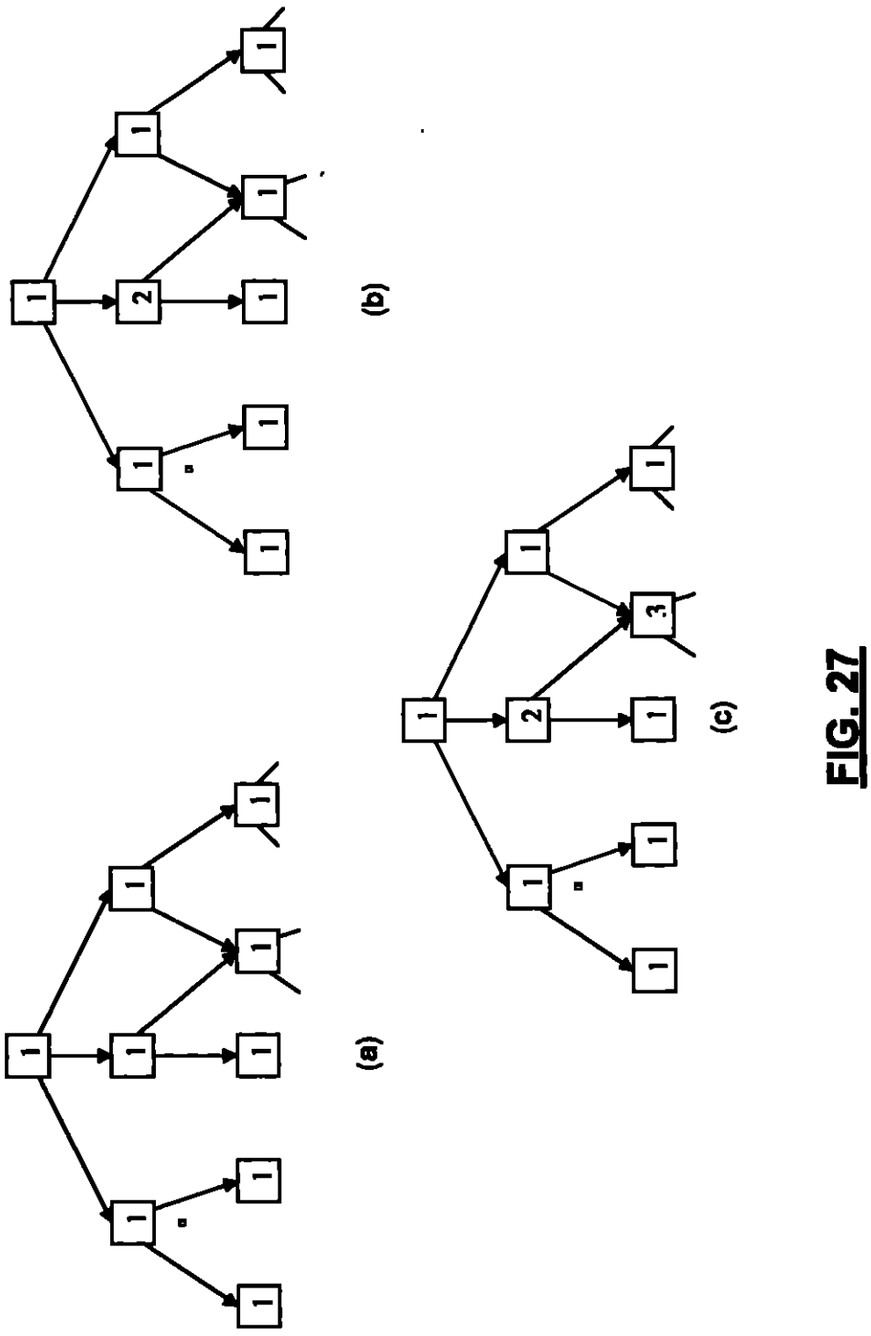


FIG. 27

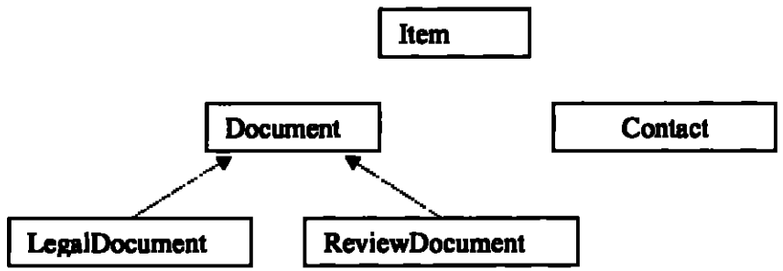

Item
Document
Contact
LegalDocument
ReviewDocument

FIG. 28

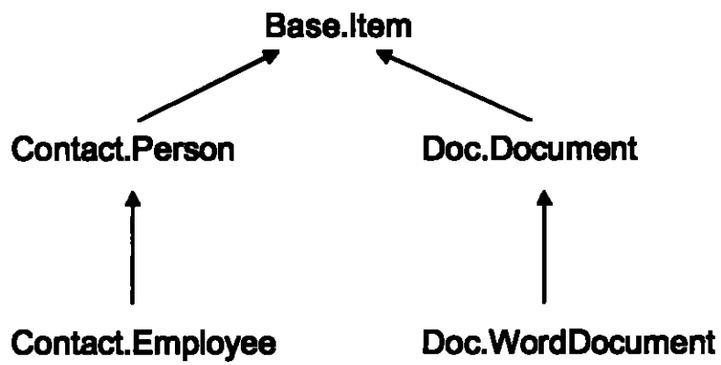

Base.Item
Contact.Person
Doc.Document
Contact.Employee
Doc.WordDocument

FIG. 29

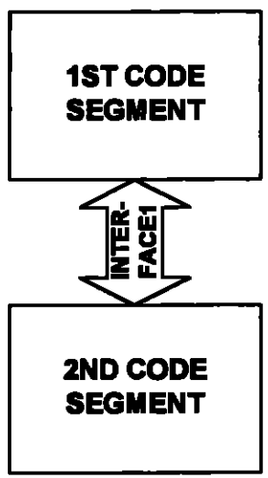


FIG. 30A

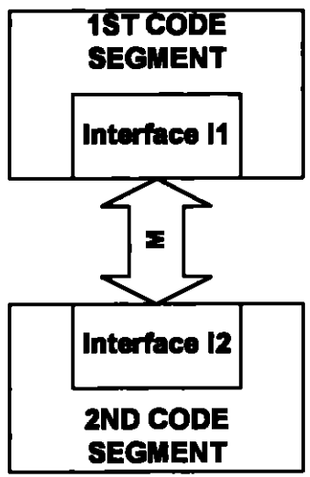


FIG. 30B

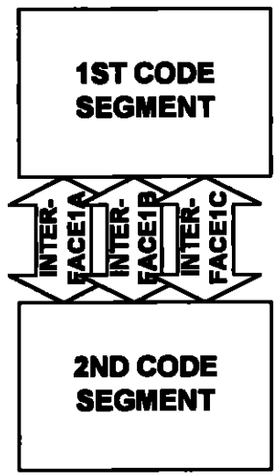


FIG. 31A

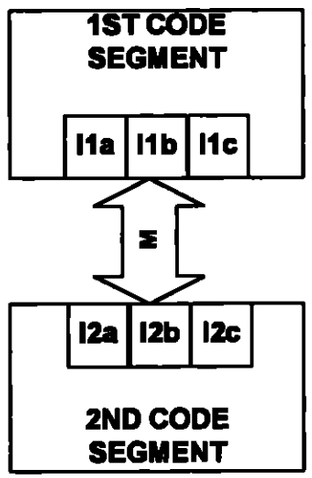


FIG. 31B

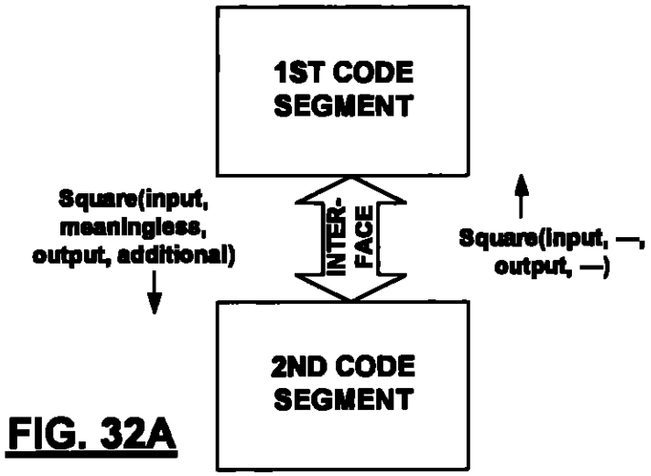


FIG. 32A

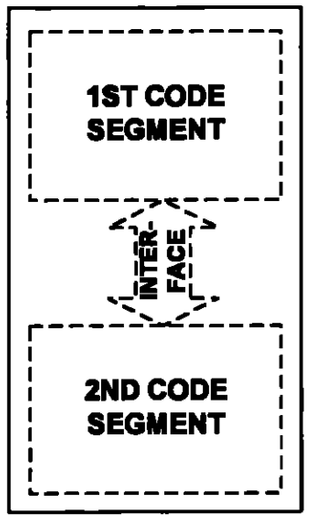


FIG. 33A

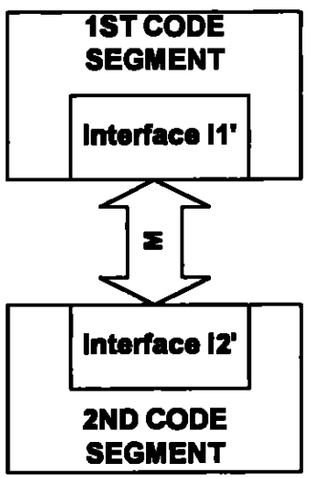


FIG. 32B

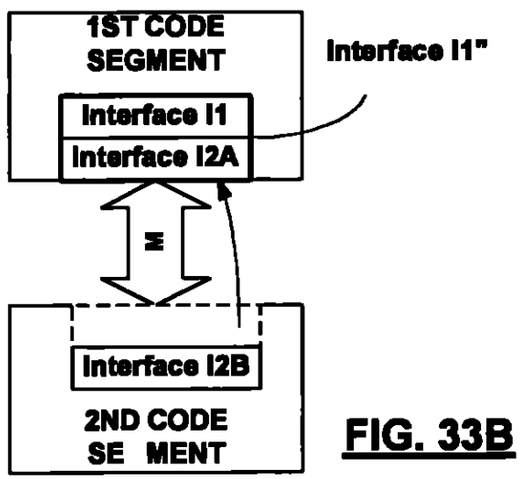


FIG. 33B

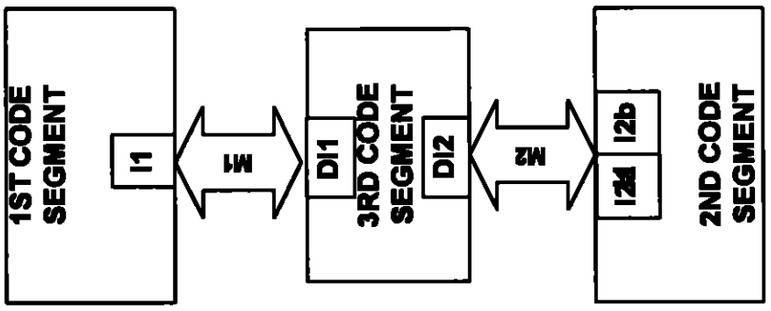


FIG. 34B

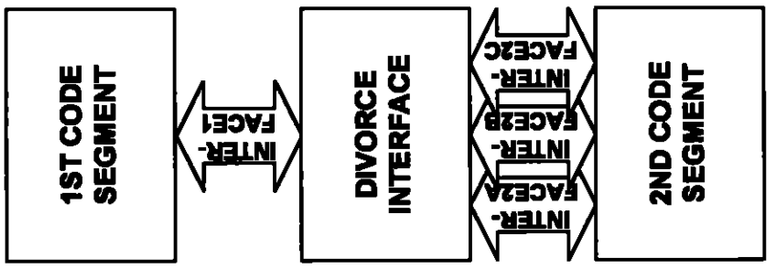


FIG. 34A

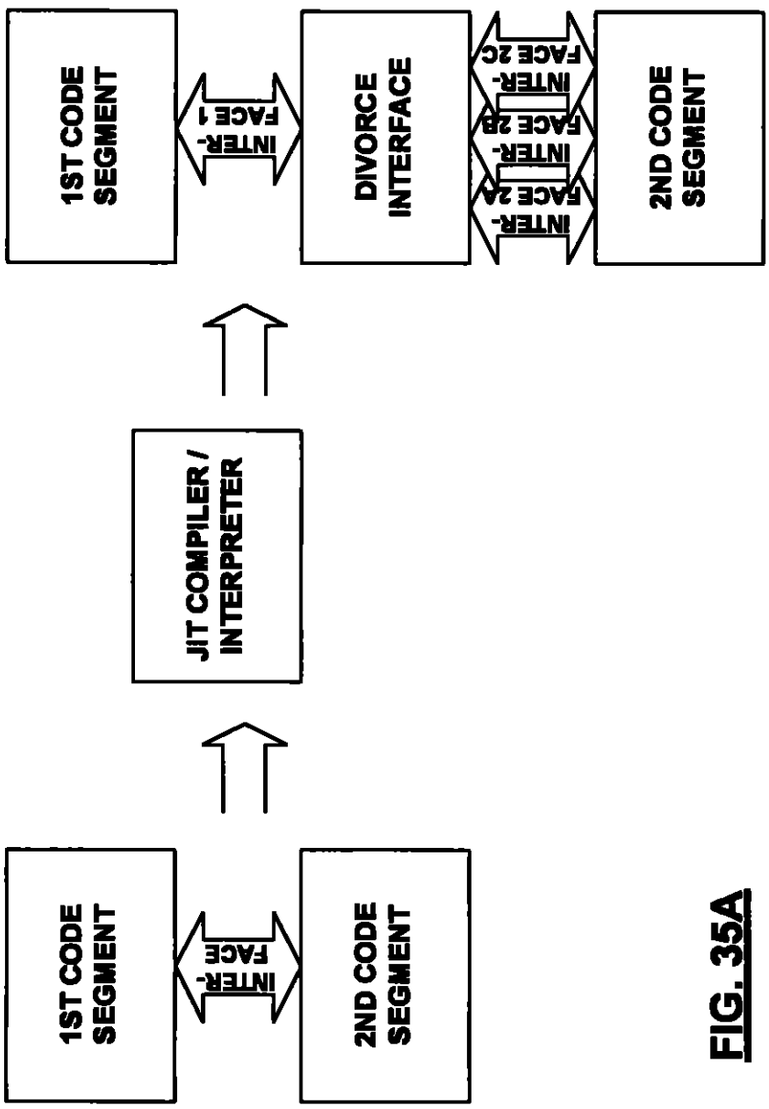

1ST CODE SEGMENT
INTER-FACE
2ND CODE SEGMENT
JIT COMPILER / INTERPRETER
1ST CODE SEGMENT
INTER-FACE 1
DIVORCE INTERFACE
INTER-FACE 2A
INTER-FACE 2B
INTER-FACE 2C
2ND CODE SEGMENT

FIG. 35A

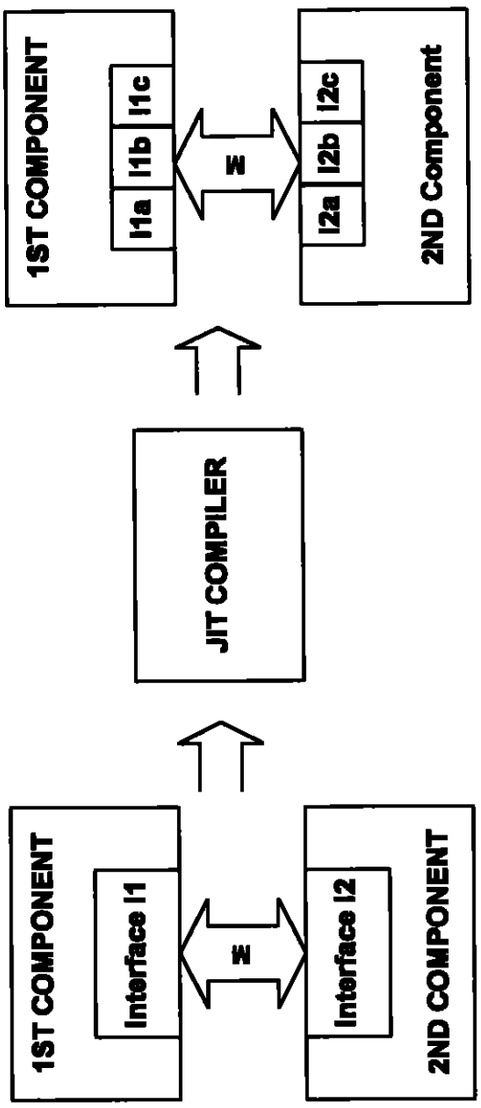

1ST COMPONENT
I1a
I1b
I1c
2ND Component
I2a
I2b
I2c
JIT COMPILER
1ST COMPONENT
Interface I1
Interface I2
2ND COMPONENT

FIG. 35B

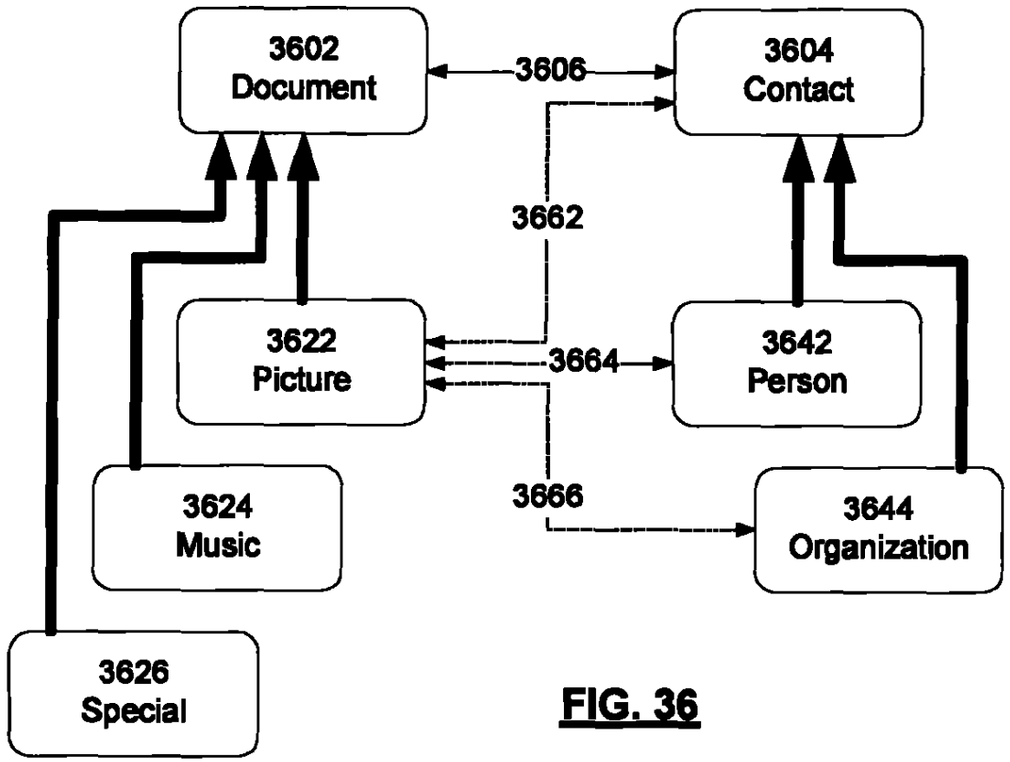
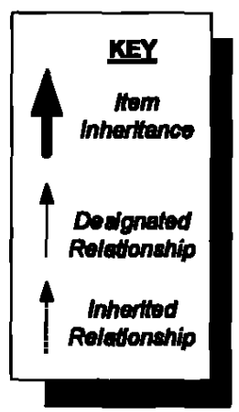

3602
Document
3606
3604
Contact
3662
3622
Picture
3664
3642
Person
3624
Music
3666
3644
Organization
3626
Special

FIG. 36


KEY
Item Inheritance
Designated Relationship
Inherited Relationship

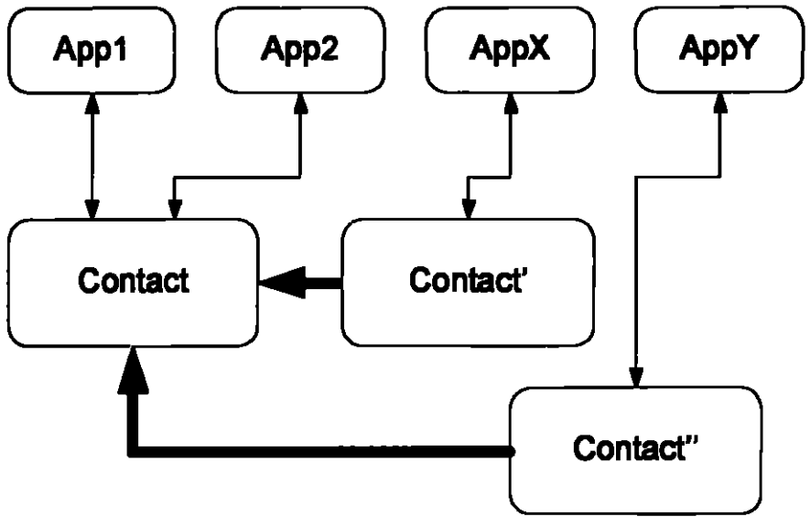


FIG. 37A

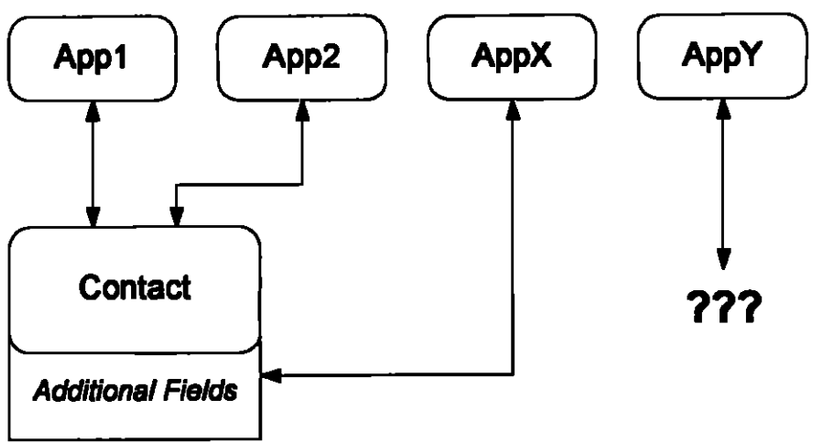


FIG. 37B

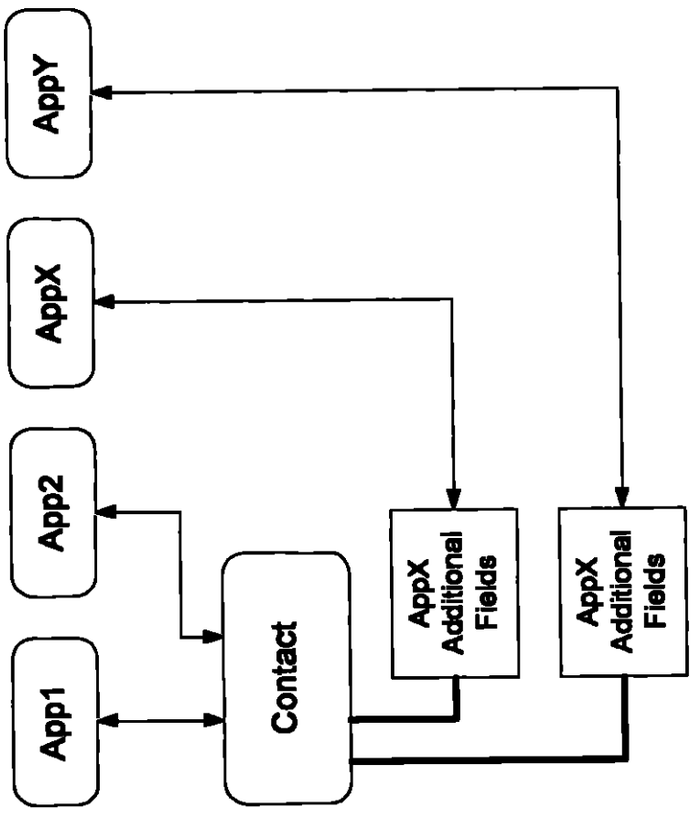

AppY
AppX
App2
App1
Contact
AppX Additional Fields
AppX Additional Fields

FIG. 37C

366

REQUISITOS NECESARIOS PARA PRESENTACION Y AGILIZACION DEL TRAMITE

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

Firma Responsable:

Fecha: