(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 697 30 657 T2** 2005.02.10

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 817 101 B1**
(21) Deutsches Aktenzeichen: **697 30 657.7**
(96) Europäisches Aktenzeichen: **97 110 830.3**
(96) Europäischer Anmeldetag: **01.07.1997**
(97) Erstveröffentlichung durch das EPA: **07.01.1998**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **15.09.2004**
(47) Veröffentlichungstag im Patentblatt: **10.02.2005**

(51) Int Cl.[7]: **G06F 17/30**

(30) Unionspriorität:
674318 **01.07.1996** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
Microsoft Corp., Redmond, Wash., US

(74) Vertreter:
Grünecker, Kinkeldey, Stockmair & Schwanhäusser, 80538 München

(72) Erfinder:
Chan, Chuck Y., Bellevue, Washington 98006, US; Ganugapati, Krishna, Bellevue, Washington 98007, US; Johnson, Margaret K., Kirkland, Washington 98033, US; Judd, Steven G., Redmond, Washington 98053, US; Kwan, Stuart L. S., Bellevue, Washington 98007, US; Watson, Colin, Issaquah, Washington 98027, US

(54) Bezeichnung: **Verfahren und System für gleichmässigen Zugriff auf mehrere Verzeichnisdienste**

Beschreibung

Technisches Gebiet

[0001] Diese Erfindung betrifft Computer-basierte Verzeichnisdienste und ganz besonders ein Verfahren und System zum gleichmäßigen Zugreifen auf die Verzeichnisdienste.

Hintergrund der Erfindung

[0002] Computersysteme speichern und unterhalten typischerweise eine große Menge an Daten, die das Computersystem und seine Benutzer betreffen. Zum Beispiel kann ein Computersystem Namen, E-Mail-Adressen und Telefonnummern der Benutzer des Computersystems bewahren. Das Computersystem kann auch Information bezüglich der verschiedenen mit dem Computersystem verbundenen Drucker unterhalten. Diese Information kann Druckermerkmale (z.B. Laser und Farbe) und Namen von Benutzern enthalten, die berechtigt sind, jeden Drucker zu benutzen. Mehrere verschiedene Arten von Computerprogrammen, bekannt als "Clients", können Zugang zu dieser Information benötigen. Zum Beispiel kann ein Client ein Computerprogramm sein, das festzustellen hat, welche Drucker benutzt werden können, um ein Dokument in Farbe zu drucken. Ein anderer Client kann ein Computerprogramm sein, das einem Systemverwalter erlaubt, die Information zu aktualisieren, um z.B. Information bezüglich eines neu hinzugefügten Druckers widerzuspiegeln. Computersysteme haben traditionell solche Information in einer Vielfalt von Stellen und Formaten gespeichert. Zum Beispiel kann Information bezüglich eines Druckers in einer Konfigurationsdatei gespeichert werden, und Information bezüglich eines Benutzers kann in einer Datenbank gespeichert werden. Typischerweise war es daher für einen Programmierer, der einen Client entwickelte, sehr schwer, auch nur zu wissen, wo nach dieser Information zu suchen ist, geschweige denn zu wissen, wie darauf zuzugreifen ist.

[0003] Einige Computersysteme stellen einen Verzeichnisdienst bereit, um beim Speichern und Unterhalten dieser Information zu helfen. Ein Verzeichnisdienst stellt einen Speicher dieser Information bereit, auf den ein Client über das Netzwerk zugreifen kann. Verzeichnisdienste verweisen gewöhnlich auf die Dinge (z.B. Benutzer, Drucker oder Zugriffsgruppen), für die Information unterhalten wird, als "Objekte". Verzeichnisdienste gliedern solche Objekte hierarchisch in ein Verzeichnis. Das heißt, ein Objekt, auf das als ein Behälterobjekt verwiesen wird, kann verschiedene andere Objekte enthalten, auf die als enthaltene Objekte verwiesen wird. Zum Beispiel ist eine Zugriffsgruppe, die Benutzer umfasst, die sich in gemeinsame Zugriffsrechte auf eine Ressource teilen, ein Behälterobjekt, das Benutzer enthält, die enthaltene Objekte sind. Jedes Objekt, für das ein Verzeichnisdienst Information untehält, hat einen eindeutigen Identifikator (z.B. ein Name), mit dem ein Client das Objekt identifizieren kann. Obwohl Verzeichnisdienste ursprünglich entwickelt wurden, um Information bezüglich Computersystemen zu unterhalten, können sie benutzt werden, um Information ohne Bezug auf Computersysteme zu unterhalten. Zum Beispiel kann ein Verzeichnisdienst von einem Flugzeughersteller verwendet werden, um eine Teileliste für die Komponenten und Unterkomponenten eines Flugzeugs zu unterhalten.

[0004] **Fig.** 1 veranschaulicht eine Musterhierarchie eines Verzeichnisdienstes. Jeder Block stellt ein Objekt einer bestimmten "Objektklasse" dar. Zum Beispiel, ein Objekt, das eine Firma darstellt, würde eine Objektklasse mit Namen "Firma" haben. Jede Objektklasse definiert die Merkmale von Objekten dieser Objektklasse. Zum Beispiel kann die Objektklasse Firma die Merkmale "Name" und "Adresse" definieren. Jedes Objekt hat einen Merkmalwert für jedes für die Objektklasse definierte Merkmal. Zum Beispiel kann ein Objekt der Objektklasse Firma den Merkmalwert "MS" für das Namensmerkmal haben. Block **101** entspricht einem Objekt der Objektklasse Firma und hat einen Wert für den Namen und Adressmerkmale. Block **102** entspricht einem Objekt der Objektklasse Abteilung und hat einen Wert für den Namen der Abteilung (z.B. "System"). Block **103** entspricht einem Objekt der Objektklasse Benutzer und hat einen Wert für die E-Mail-Adresse des Benutzers. Weil der Verzeichnisdienst hierarchisch gegliedert ist, kann jedes Objekt durch einen Pfad von der Wurzel zu dem Objekt eindeutig identifiziert werden. Zum Beispiel wird Objekt **102** durch den Pfad "Firma=MS\Abtei-lung=System" eindeutig identifiziert. Der Verzeichnisdienst stellt einen begrifflichen endlichen Raum, bezeichnet als Namensraum, bereit, in dem ein gegebener Name aufgelöst werden kann. Der Verzeichnisdienst löst Pfade zu Objekten in dem Verzeichnis unzweideutig auf.

[0005] Verschiedene Lieferanten stellen Verzeichnisdienstsysteme bereit. Jeder Lieferant entwirft und implementiert typischerweise eine Anwendungsprogrammschnittstelle (API), um Clients zu erlauben, auf sein Verzeichnisdienstsystem zuzugreifen. Jedes Verzeichnisdienstsystem kann sehr verschiedene API-Sätre haben, was in der Regel der Fall ist. Wenn ein Client Verzeichnisdienstsysteme benutzen muss, weil z.B. der Client in einer Umgebung arbeitet, die Personal Computer und Mainframe Computer enthält, müsste der Programmierer des Client über die AIP-Sätre jedes Verzeichnisdienstes Bescheid wissen und den Client entwerfen, um

jeden der API-Sätze zu unterstützen.

[0006] Obwohl jeder API-Satz tieferantenspezifisch ist, stellen API-Sätze im Allgemeinen ähnliche Funktionalität bereit. Die API-Sätze enthalten gewöhlich Funktionen zum Zugreifen auf Merkmalwerte der Objekte und Funktionen zum Definieren neuer Objektklassen. Die Funktionen zum Manipulieren von Objekten sind z.B. OpenObject, ReadObject, WriteObject, ListObjects, CloseObject, CreateObject und DeleteObject. Der Funktion OpenObject wird die Identifikation (z.B. Pfad) zu einem Objekt übergeben, und sie gibt eine Handhabe zurück, die dieses Objekt identifiziert. Die Handhabe wird später von dem Client benutzt, um das offene Objekt gegenüber dem Verzeichnisdienst zu identifizieren. Die Funktion OpenObject benutzt die Identifikation, um das Objekt zu identifizieren, und lokalisiert dann die Merkmale des identifizierten Objekts. Der Funktion ReadObjekt werden eine Handhabe zu einem offenen Objekt und eine Liste der Namen der Merkmale dieses Objekts, die zurückzugewinnen sind, übergeben. Die Funktion ReadObjekt gibt die aktuellen Merkmalwerte für diese Merkmale zurück. Der Funktion WriteObject wird eine Handhabe zu einem offenen Objekt und einen Liste von Merkmalnamen und Merkmalwertpaaren übergeben. Die Funktion WriteObject setzt jedes genannte Merkmal in dem offenen Objekt auf den Merkmalwert des Paares. Der Funktion ListObjects wird eine Handhabe zu einem offenen Objekt übergeben, und sie gibt eine Liste zurück, die die Identifikation jedes Objekts enthält, das in dem offenen Objekt enthalten ist. Der Funktion CloseObject wird eine Handhabe zu einem offenen Objekt übergeben, und sie schließt das Objekt, sodass darauf mit dieser Handhabe nicht mehr zugegriffen werden kann. Der Funktion CreateObject wird eine Objektklasse übergeben, sie erzeugt ein Objekt der Objektklasse, die in dem offenen Objekt enthalten ist, und gibt eine Handhabe zu dem enthaltenen Objekt zurück. Der Funktion DeleteObject wird eine Handhabe zu dem offenen Objekt übergeben, und sie entfernt das offene Objekt aus seinem Behälterobjekt.

[0007] Die Funktionen zum Definieren der Objektklassen sind CreateObjectClass, DeleteObjectClass, CreateProperty, DeleteProperty, AddPropertyToObjectClass, DeletePropertyFrom-ObjectClass und ListPropertiesOfObjectClass. Der Funktion CreateObjectClass wird der Name einer neuen Objektklasse übergeben, und sie erzeugt eine neue Objektklasse. Sobald eine Objektklasse erzeugt ist, kann ein Client Objekte dieser Objektsklasse erzeugen. Der Funktion CreateProperty werden der Name eines Merkmals und der Merkmaltyp (z.B. Ganzzahl oder Zeichenkette) übergeben, und sie erzeugt ein Merkmal dieses Merkmaltyps (z.B. Merkmalname von "Adresse" mit Merkmaltyp "Zeichenkette"). Der Funktion AddProperty-ToObjectClass werden der Name einer Objektklasse und der Name eines Merkmals übergeben, und sie fügt das genannte Merkmal als ein Merkmal der genannten Objektklasse hinzu. Der Funktion ListPropertiesOfObjectClass wird der Name einer Objektklasse übergeben, und sie gibt eine Liste der Merkmale zurück, die dieser Objektklasse hinzugefügt wurden. Die Funktionen DeleteObjectClass, DeleteProperty und DeletePropertyFromObjectClass führen das durch ihre Namen angedeutete Verhaften durch.

[0008] Die vorliegende Efindung wird im Folgenden unter Verwendung einiger objektorientierter Verfahren beschrieben, und daher wird eine Übersicht von wohl bekannten objektorientierten Programmiertechniken bereitgestellt. (Der Begriff "Objekt" hat viele verschiedene Bedeutungen, wenn er in verschiedenen Kontexten gebraucht wird. Im Rest des Hintergrunds wird der Begriff "Objekt" in einem objektorientierten Sinn gebraucht, um auf eine In-Speicher-Datenstruktur zu verweisen.) Zwei übliche Eigenschaften von objektorienterten Programmiersprachen sind die Unterstützung für Dateneinkapselung und Daten-Hinterlassenschaft. Dateneinkapselung betrifft das Binden von Funktionen und Daten. Hinterlassenschaft betrifft die Fähigkeit, einen Datentyp in Form von anderen Datentypen zu deklarieren. In der Sprache C++ werden Einkapselung und Hinterlassenschaft durch die Verwendung von Klassen unterstützt. Eine Klasse ist ein benutzerdefinierter Typ. Eine Klassendeklaration beschreibt die Datenelemente und Funktionselemente der Klasse. Ein Funktionselement wird auch als ein Verfahren einer Klasse bezeichnet. Die Datenelemente und Funktionselemente einer Klasse werden zusammengebunden, insofern als die Funktion auf einer Instanz der Klasse arbeitet. Ein Instanz einer Klasse wird auch ein Objekt der Klasse genannt. Eine Klasse liefert daher eine Definition für eine Gruppe von Objekten mit ähnlichen Merkmalen und gemeinsamem Verhaften.

[0009] Um Speicher für ein Objekt einer bestimmten Klasse (Typ) zuzuteilen, wird ein Objekt eingerichtet. Sobald eingerichtet, können den Datenelementen des einzelnen Objekts Daten zugewiesen werden. Ebenfalls können, sobald eingerichtet, die Funktionselemente des einzelnen Objekts aufgerufen werden, um auf die Datenelemente zuzugreifen und sie zu manipulieren. Auf diese Weise implementieren daher die Funktionselemente das Verhalten des Objekts, und dasObjekt liefert eine Struktur zum Einkapseln von Daten und Verhaften in eine einzige Wesenheit.

[0010] Um das Konzept der Hinterlassenschaft zu unterstützen, können Klassen aus (basierend auf der Deklaration von) anderen Klassen abgeleitet werden. Eine abgeleitete Klasse ist eine Klasse, die die Eigenschaf-

ten, Datenelemente und Funktionselemente ihrer Grundklassen erbt. Eine Klasse, die die Eigenschaften einer anderen Klasse erbt, ist eine abgeleitete Klasse. Eine Klasse, die nicht die Eigenschaften einer anderen Klasse erbt, ist eine Primär(Wurzel) Klasse. Eine Klasse, deren Eigenschaften von einer anderen Klasse geerbt werden, ist eine Grundklasse. Ein abgeleitete Klasse kann die Eigenschaften von mehreren Klassen erben; das heißt, eine abgeleitete Klasse kann mehrere Grundklassen haben. Dies wird als mehrfache Hinterlassenschaft bezeichnet.

[0011] Eine Klasse kann auch spezifizieren, ob ihre Funktionselemente virtuell sind. Deklarieren, dass ein Funktionselement virtuell ist, bedeutet, dass die Funktion durch eine Funktion des gleichen Namens und Typs in einer abgeleiteten Klasse umgestoßen werden kann. Wenn eine virtuelle Funktion deklariert wird, ohne Implementierung bereitzustellen, wird sie als eine reine virtuelle Funktion bezeichnet. Eine reine virtuelle Funktion ist eine Funktion, die mit dem reinen Spezifizierer, "=0", deklariert wird. Wenn eine Klasse eine reine virtuelle Funktion spezifiziert, dann muss jede abgeleitete Klasse eine Implementierung für dieses Funktionselement spezifizieren, bevor das Funktionselement aufgerufen werden kann. Eine Klasse, die wenigstens ein reines virtuelles Funktionselement enthält, ist eine abstrakte Klasse.

[0012] **Fig.** 2 ist ein Blockdiagramm, das typische Datenstrukturen veranschaulicht, die benutzt werden, um ein Objekt darzustellen. Ein Objekt umfasst Instanzdaten (Datenelemente) und Funktionselemente, die das Verhalten des Objekts implementieren. Die Datenstrukturen, die zur Darstellung eines Objekts benutzt werden, umfassen die Instanzdatenstruktur **201**, die Tabelle virtueller Funktionen **202** und die Funktionselemente **203**, **204**, **205**. Die Instanzdatenstruktur **201** enthält einen Zeiger auf die Tabelle virtueller Funktionen **202** und enthält Datenelemente. Die Tabelle virtueller Funktionen **202** enthält einen Eintrag für jedes für das Objekt definierte virtuelle Funktionselement. Jeder Eintrag enthält einen Verweis auf den Code, der das entsprechende Funktionselement implementiert. Das Layout dieses Musterobjekts entspricht Modellen, die im US Patent Nr. 5,297,284, betitelt "A Method for Implementing Virtual Functions and Virtual Bases in a Compiler for an Object Oriented Programming Language" beschrieben sind. Im Folgenden wird ein Objekt als ein Fall einer Klasse beschrieben, wie durch die Programmiersprache C++ definiert. Eine in der Technik erfahrene Person würde erkennen, dass andere Objektmodelle unter Verwendung anderer Programmiersprachen definiert werden können.

[0013] Ein Vorteil der Verwendung von objektorientierten Techniken ist, dass diese Techniken benutzt werden können, um das Teilen von Objekten zu erleichtern. Zum Beispiel kann ein Programm, das die Funktionselemente eines eingerichteten Elements implementiert (ein "Server-Programm"), das Objekt mit einem anderen Programm (ein "Client-Programm") teilen. Um einem Objekt einer beliebigen Klasse zu erlauben, mit einem Client-Programm geteilt zu werden, werden Schnittstellen definiert, durch die auf ein Objekt zugegriffen werden kann, ohne dass des Client-Programm zum Zeitpunkt der Compilation Zugriff auf die Klassendefinitionen haben muss. Eine Schnittstelle ist ein benannter Satz von logisch verwandten Funktionselementen. In C++ ist eine Schnittstelle eine abstrakte Klasse ohne Datenelemente, und deren virtuelle Funktionen sind alle rein. Eine Schnittstelle stellt daher ein veröffentlichtes Protokoll zum Kommunizieren von zwei Programmen bereit. Schnittstellen werden typischerweise zur Ableitung benutzt: Ein Programm definiert (implementiert) Klassen, die Implementierungen für die Schnittstellen, von denen die Klassen abgeleitet werden, bereitstellen. Danach werden Objekte als Instanzen dieser abgeleiteten Klassen erzeugt. Objekte, die aus abgeleiteten Klassen, die bestimmte Schnittstellen implementieren, eingerichtet werden, sollen die Schnittstellen "unterstützen". Ein Objekt unterstützt, abhängig von der gewünschten Funktionalität, eine oder mehrere Schnittstellen.

[0014] Wenn ein Client-Programm ein Objekt zu teilen wünscht, muss das Client Programm auf den Code zugreifen, der die Schnittstellen für das Objekt (der abgeleitete Klassencode) implementiert. In der OLE 2.01 Umgebung, gegründet von Microsoft Corporation in Redmont, Washington, wird, um auf den abgeleiteten Klassencode (auch Klassencode genannt) zuzugreifen, jeder Klassenimplementierung ein einmaliger Klassenidentifikator (ein "CLSID") gegeben. OLE 2.01 wird in "Inside OLE", 2. Ausgabe, Microsoft Press, 1995 von Kraig Brock schmidt beschrieben. Zum Beispiel kann Code, der ein von Microsoft Corporation entwickeltes Spreadsheet-Objekt implementiert, einen Klassenidentifikator "MSSpreadsheet" haben, während Code, der ein von einer anderen Firma entwickeltes Spreadsheet-Objekt implementiert, einen Klassenidentifikator "LTSSpreadsheet" haben kann. Eine ständige Registrierung wird in jedem Computersystem bewahrt, die jeden CLSID in den Code abbildet, der die Klasse implementiert. Typischerweise wird, wenn ein Spreadsheet-Programm auf einem Computersystem installiert wird, die ständige Registrierung aktualisiert, um die Verfügbarkeit dieser Klasse von Spreadsheet-Objekten widerzuspiegeln. Solange der Spreadsheet Entwickler jedes Funktionslement, das durch die von Spreadsheet-Objekten zu unterstützenden Schnittstellen definiert wird, implementiert, und solange die ständige Registrierung aufrechterhalten wird, kann das Client-Programm auf die Funktionselemente von geteilten Spreadsheet Objekten ohne Rücksicht darauf zugreifen, welches Server-Programm sie

implementiert hat oder wie sie implementiert wurden.

[0015] Da ein Objekt einige Schnittstellen unterstützen kann und andere nicht, kann ein Client Programm zur Laufzeit feststellen müssen, ob ein bestimmtes Objekt eine bestimmte Schnittstelle untersützt. Um diese Feststellung zu ermöglichen, unterstützt jedes COM-Objekt die IUnknown-Schnittstelle, die ein Funktionselement enthält, und QueryInterface, die angibt, welche Schnittstellen für das Objekt implementiert sind. Das QueryInterface-Verfahreen ist wie folgt definiert:
virtual HRESULT QueryInterface (REFIID iid, void**ppv) = 0;

[0016] Dem QueryInterface-Verfahren wird ein Schnittstellenidentifikator in Parameter iid (von Typ REFIID) übergeben, und es gibt in Parameter ppv einen Zeiger auf die Implementierung der designierten Schittstelle "iid" zurück. Das QueryInterface-Verfahren ist typischerweise codiert, um über alle verfügbaren Schnittstellen des Objekts, zu dem es gehört, Bescheid zu wissen. Wenn das Objekt die Schnittstelle nicht unterstützt, gibt das QueryInterface-Verfahren unwahr zurück. Der Typ HRESULT gibt einen vordefinierten Status an.

[0017] Die (Unknown-Schnittstelle definiert auch die Verfahren AddRef und Release, die zum Implementieren einer Verweiszählung benutzt werden. Wann immer einer neuer Verweis auf eine Schnittstelle erzeugt wird, wird das AddRef-Verfahren aufgerufen, um einen Verweiszählwert des Objekts zu erhöhen. Wenn ein Verweis nicht mehr gebraucht wird, wird das Release-Verfahren aufgerufen, um den Verweiszählwert zu dekrementieren, und wenn der Verweiszählwert auf null geht, wird das Objekt freigesetzt.

Zusammenfassung der Erffindung

[0018] Die Erfindung wird durch die Gegenstände der unabhängigen Ansprüche bereitgestellt.

[0019] Bevorzugte Ausführungen werden in den abhängigen Ansprüchen definiert.

[0020] Die vorliegende Erfindung stellt eine Definition für OLE-Schnittstellen und ein Modell für Anbieter-Software zum Zugreifen auf eine Vielzahl von Verzeichnisdiensten in einer gleichmäßigen Weise bereit. Jeder Verzeichnisdienstanbieter verwaltet Information bezüglich Objekten dieses Verzeichnisdienstes. Die Art der Information, die ein Verzeichnisdienst für ein Objekt verwaltet, wird durch die Objektklasse des Objekts definiert. Eine Objektklasse definiert die Merkmale (d.h. Information), die ein Verzeichnisdienst für Objekte dieser Objektklasse verwaltet. Jedes Merkmal hat einen Merkmalnamen und Merkmaltyp. Ein Verzeichnisdienst hat einen Merkmalnamen für jedes durch die Objektklasse jedes Objekts definierte Merkmal. Das Verzeichnisdienstsystem umfasst eine Schema-Browsing-Komponente, eine Namenauflösungskomponente, eine Bindungskomponente und eine Erweiterungskomponente. Die Schema-Browsing-Komponente steuert das Rückgewinnen des Merkmalnamens und Merkmalstyps jedes Merkmals jeder Objektklasse jedes Verzeichnisdienstens. Ein Client eines Verzeichnisdienstsystems benutzt die Schema-Browsing-Komponente, um Merkmalnamen und Merkmaltypen der Objektklassen zurückzugewinnen. Die Namenauflösungskomponente steuert die Rückgewinnung eines eindeutigen Identifikators eines Objekts innerhalb eines Verzeichnisdienstes und das Auffinden des Objekts innerhalb des Verzeichnisdienstes. Die Bindungskomponente steuert das Binden an ein In-Speicher-Objekt, das ein lokalisiertes Objekt in einem Verzeichnisdienst darstellt. Die Erweiterungskomponente steuert das Definieren neuer Objektklassen und neuer Merkmale für jeden Verzeichnisdienst. Ein Client des Verzeichnisdienstsystems benutzt die Erweiterungskomponente, um neue Objektklassen und neue Merkmale zu definieren.

Kurzbeschreibung der Zeichnungen

[0021] **Fig.** 1 veranschaulicht eine Musterhierarchie eines Verzeichnisdienstes.

[0022] **Fig.** 2 veranschaulicht typische Datenstrukturen, die zum Darstellen eines Objekts benutzt werden.

[0023] **Fig.** 3A veranschaulicht die Hierarchie von OleDs.

[0024] **Fig.** 3B veranschaulicht die Beziehung zwischen OleDs-Objekten und ihren entsprechenden Objekten in einem Verzeichnisdienst.

[0025] **Fig.** 4 ist ein Blockschaltbild eines zum Implementieren der vorliegenden Erfindung konfigurierten Computersystems.

[0026] **Fig.** 5 ist ein Flussdiagramm der Funktion OleDsGetObject, die die Bindefunktion von OleDs ist.

[0027] **Fig.** 6 ist ein Flussdiagramm des Verfahrens IOleDs::GetInfo.

[0028] **Fig.** 7 ist ein Flussdiagramm des Verfahrens IOleDS::Get.

[0029] **Fig.** 8 ist ein Flussdiagramm des Verfahrens IOleDsContainer::CopyHere.

[0030] **Fig.** 9 ist ein Flussdiagramm des Verfahrens IOleDs::Create.

[0031] **Fig.** 10 ist ein Flussdiagramm der Prozedur zum Anzeigen aller Namensräume.

Ausführliche Beschreibung der Efindung

[0032] Die vorliegende Erfindung stellt ein Rahmenwerk zum gleichmäßigen Zugriff auf verschiedene unterschiedliche Verzeichnisdienste bereit. In einer bevorzugten Ausführung stellt das Verzeichnisdienstsystem, bezeichnet als OleDs, eine Architektur bereit, die Clients erlaubt, auf den Inhalt dieser verschiedenen Verzeichnisdienste unter Verwendung eines einzigen, gemeinsamen Satzes von OleDs-Objekten, -Attributen und -Schnittstellen zuzugreifen. Ein OleDs-Objekt ist eine In-Speicher-Struktur, die einem Objekt eines Verzeichnisdienstes oder einer Objektklasse des Verzeichnisdienstes entspricht. Wenn ein OleDs-Objekt einem Objekt entspricht, stellt es eine Schnittstelle zum Zugreifen auf die Merkmalwerte und Verfahren dieses Objekts bereit. Wenn ein OleDs-Objekt einer Objektklasse entspricht, stellt es eine Schnittstelle zum Zugreifen auf die Definition dieser Objektklasse bereit. Jeder Anbieter eines Verzeichnisdienstes liefert eine Implementierung dieser Schnittstellen für seinen Verzeichnisdienst, die das Verhalten seines API-Satzes in das Verhalten dieser OleDs-Schnittstellen abbildet. Auf diese Weise kann ein Client, der entwickelt ist, um die OleDs-Architektur zu verwenden, auf jeden dieser Verzeichnisdienste ungeachtet der Unterschiede in ihren API-Sätzen zugreifen.

[0033] OleDs modelliert alle Verzeichnisdienste als innerhalb eines Wurzelbehälterobjekts, bezeichnet als ein "Namensraum"-Behälter, befindlich. **Fig.** 3 veranschaulicht den Namensraum-Behälter. Die Objekte in dem Namensraum-Behälter stellen die verschiedenen Verzeichnisdienste dar. OleDs stellt eine Schnittstelle zum Aufzählen der in dem Namensraum-Behälter enthaltenen Verzeichnisdienste bereit. Jeder enthaltene Verzeichnisdienst wird durch ein OleDs-Namensraum-Behälterobjekt dargestellt. Ein Client kann die Schnittstelle verwenden, um das Vorhandensein eines Verzeichnisdienstes zu ermitteln, und kann auf sein Schema zugreifen, um die Definition der Objektklassen des Verzeichnisdienstes zu ermitteln. Das Schema enthält eine Definition der Objektklassen des Verzeichnisdienstes. Durch die vom OleDs bereitgestellten Schnittstellen kann ein Client zur Laufzeit das Vorhandensein und den Inhalt von Verzeichnisdiensten ermitteln, die zur Kompilationszeit nicht bekannt waren.

[0034] Da jeder Verzeichnisdienst typischerweise seine eigene Konvention zur Benamung seiner Objekte hat, stellt das OleDs eine gleichförmige Benamungskonvention bereit, die jedes Objekt in den verschiedenen Verzeichnisdiensten ohne Konflikt eindeutig identifiziert. Dieser eindeutige Identifikator wird als ein OleDS-Pfad bezeichnet Der OleDs-Pfad ist eine Zeichenkette, die den Verzeichnisdienst und den Satz von eingepackten Behälterobjekten identifiziert, die geöffnet werden müssen, um auf das identifizierte Objekt zuzugreifen. Das OleDs hat das folgende Format:

@Namespaceldentifier!<namespace path tu object>

oder

Namespaceldentifier://<namespace path to object>

[0035] Der "Namespaceldentifier" identifiziert den Verzeichnisdienst eindeutig. Die Zeichenfolge nach dem "1" oder ":" ist in einem vom Verzeichnisdienst unabhängigen Format. Das heißt, jeder Verzeichnisdienst kann sein eigenes Format definieren.

[0036] OleDs stellt eine Bindefunktion zum Binden eines OleDs-Pfades an ein OleDs-Objekt bereit, das dem durch den OleDs-Pfad identifizierten Objekt entspricht. Dieser Bindefunktion wird ein OleDs-Pfad übergeben, und sie gibt einen Zeiger auf das dem Objekt eintsprechende OleDs-Objekt zurück. Um das OleDs-Objekt zu binden, teilt die Bindefunktion den Namensraum-Identifikator von dem OleDs-Pfad. Die Bindefunktion benutzt diesen Namensraum-Identifikator, um Code zurückzugewinnen (z.B. in einer dynamisch verbundenen Bibliothek), der den Zugriff auf den identifizierten Namensraum implementiert. (OleDs unterhält eine Registrierung

für Verzeichnisdienstanbieter, um die Stelle ihrer Implementierung von OleDs-Schnittstellen zu registrieren.) Die Bindefunktion ruft dann den rückgewonnenen Code durch Übergeben des OleDs-Pfades auf. Der aufgerufene Code benutzt den API-Satz des Verzeichnisdienstes, um auf das identifizierte Objekt zuzugreifen und das durch durch die OleDs-Schnittstelle definierte Verhalten zu implementieren.

[0037] OleDs modelliert dieObjekte jedes Verzeichnisdienstes als entweder OleDs-Behälterobjekte oder OleDs-Blattobjekte. Um auf ein Behälterobjekt eines Verzeichnisdienstes zuzugreifen, wird ein OleDs-Behälterobjekt eingerichtet, das eine durch den Anbieter des Verzeichnisdienstes bereitgestellte Implementierung hat. Desgleichen wird, um auf ein Blattobjekt des Verzeichnisdienstes zuzugreifen, ein OleDs-Blattobjekt eingerichtet, das eine durch den Anbieter des Verzeichnisdienstes bereitgestellte Implementierung hat. Jedes OleDs-Objekt enthüllt eine Schnittstelle zum Zugreifen auf die Merkmale des entsprechenden Objekts des Verzeichnisdienstes. Jedes OleDs-Behälterobjekt enthüllt außerdem eine Schnittstelle, über die auf seine enthaltenen Objekt durch einen Client zugegriffen werden kann.

[0038] **Fig.** 4 veranschaulicht die Beziehung zwischen OleDs-Objekten und ihren entsprehenden Objekten in einem Verzeichnisdienst. Jeder Verzeichnisdienstanbieter stellt Implementierungen jeder durch die OleDs-Objekte enthüllten Schnittstelle bereit. Diese Implementierungen bewirken ein Abbilden des Verhaltens des API-Satres des Anbieters in das Verhalten der OleDs-Schnittstellen. Jeder Client kann daher gleichmäßig auf vielfache Verzeichnisdienste unter Verwendung der Anbieter-Implementierungen der OleDs-Schnittstellen zugreifen.

[0039] Um Clienten beim Zugreifen auf die Objekte eines Verzeichnisdienstes zu helfen, definiert OleDs ein Modell zur Darstellung des Schemas jedes Verzeichnisdienstes. Begrifflich wird jeder Verzeichnisdienst als ein Schemabehälterobjekt besitzend angesehen, das ein Schemaobjekt für jede Objektklasse enthält. Das Schemaobjekt definiert die Merkmale der Objektklasse. Dem Schemabehälter wird ein vordefinierter Name in dem Namensraum, z.B. "Schema" zugewiesen. Um auf einen Schemabehälter zuzugreifen, benutzt ein Client die OleDs-Bindefunktion durch Übergeben des OleDs-Pfades des Schemas (z.B. "@WinNTDS! Schema"). Die OleDs-Bindefunktion gibt einen Zeiger auf ein OleDs-Behälterobjekt zurück, das dem Schemabehälterobjekt entspricht. Ein Client kann Schnittstellen des OleDs-Behälterobjekts benutzen, um die enthaltenen Schemaobjekte aufzuzählen. In dieser Weise kann ein Client die Definition der verschiedenen Objektklassen zur Laufzeit ermitteln. Außerdem kann ein Client zusätzliche Objektklassen durch Hinzufügen von Schemaobjekten zu dem Schemabehälterobjekt definieren.

[0040] OleDs definiert im Voraus mehrere Objektklassen, um die Implementierung des Abbildens des Verhaltens eines Verzeichnisdienst-API-Satres in das Verhalten der OleDs-Schnittstellen zu erleichtern. Insbesondere stellt OleDs Objekklassendefinitionen für gemeinsame Behälterobjekte, z.B. Organisationseinheit und Land, und für gemeinsame Blattobjekte, z.B. ein Benutzer oder eine Ressource, bereit.

[0041] OleDs definiert das Konzept eines Vorgabe-Namensraumes oder Verzeichnisdienstes. Ein Vorgabe-Namensraum ist der Namensraum, den ein Client zu benutzen wünscht, wenn in dem OleDs-Pfad kein Namensraum spezifiziert ist. Die Identifikation des Vorgabe-Namensraumes für einen Client oder Benutzer würde typischerweise in der durch den Client oder Benutzer indexierten Registrierung gespeichert werden.

[0042] OleDs definiert auch das Konzept eines Vorgabe-OleDs-Objekts, das durch jeden Anbieter implementiert wird, um einem Client zu erlauben, Objekte einer neu definierten Objektklasse dem Verzeichnisdienst hinzuzufügen. Ein Verzeichnisdienstanbieter würde typischerweise eine Implementierung eines OleDs-Objekts für jede Objektklasse in dem Verzeichnisdienst bereitstellen. Zum Beispiel kann ein Anbieter verschiedene Implementierungen für die Objektklassen für ein Computerobjekt und Firmenobjekt bereitstellen. Jede Implementierung hat Kenntnis davon, welche Merkmale für die entsprechende Objektklasse definiert sind, und kann den API-Satz auffordern, die Werte der Merkmale zurückzugewinnen. Wenn jedoch ein Client ein neues Objekt zur Laufzeit definiert, wird es keine entsprechende Implementierung geben. Folglich stellt jeder Anbieter, der es erlaubt, dass neue Objektklassen für seinen Verzeichnisdienst definiert werden, auch eine Implementierung eines Vorgabe-OleDs-Objekts bereit. Wann immer ein Client verlangt, ein OleDs-Objekt für ein Objekt einzurichten, für das es keine Implementierung gibt, wird ein Vorgabe-OleDs-Objekt eingerichtet.

[0043] Das Vorgabe-OleDs-Objekt enthält den Namen der Objektklasse, der es entspricht, den es benutzt, um die Definition der Objektklasse aus dem Schema zurückzugewinnen. Mittels dieser Definition kann das Vorgabe-OleDs-Objekt auf das entsprechende Objekt des Verzeichnisdienstes zugreifen.

[0044] **Fig.** 3A ist ein Blockdiagramm, das die Hierarchie des OleDs veranschaulicht. Die Hierarchie enthält

einen als "Namensräume" bezeichneten Wurzelbehälter. Der Namensraum-Behälter enthält folgerichtig den Namensraum für jeden Verzeichnisdienst. **Fig.** 3B ist ein Blockdiagramm, das die Architektur des OleDs veranschaulicht. Jeder Anbieter eines Verzeichnisdienstes implementiert ein Abbilden des Verhaltens des Anbieter-API-Satres in das Verhalten der OleDs-Schnittstellen. Jedes OleDs-Behälterobjekt enispricht einem Behälterobjekt in dem Verzeichnisdienst, und jedes OleDs-Blattobjekt entspricht einem Blattobjekt in dem Verzeichnisdienst.

[0045] OleDs sorgt für frühes und spätes Binden eines Client an die OleDs-Objekte. Frühes Binden bedeutet, dass der Entwickler eines Client zur Kompilationszeit den Namen der Merkmale jedes Objekts kennt. Der Client kann daher programmiert werden, um auf die Werte von diesen Merkmalen unter Verwendung von Verfahren zum "Holen" und "Ablegen" von Merkmalen (z.B. "get_Address" oder "put_Name") direkt zuzugreifen. Jedes OleDs-Objekt enthüllt die IDispatch-Schnittstelle (definiert in Ole 2.01), die es erlaubt, zur Laufzeit einen Client an ein Merkmal zu binden (d.h. spätes Binden).

[0046] Jdes OleDs-Objekt enthüllt die IOleDs-Schnittstelle. Die IOleDs-Schnittstelle definiert verschiedene Attribute des OleDs-Objekts zusammen mit Verfahren zum Zugreifen auf die Merkmale des entsprechenden Objekts in dem Verzeichnisdienst. (Im Folgenden betrifft der Begriff "Merkmal" Daten in einem Verzeichnisdienst, und der Begriff "Attribut" betrifft Daten, die durch OleDs für die OleDs-Objekte definiert werden.) Jedes OleDs-Objekt, das ein Behälter ist, stellt auch die IOleDsContainer-Schnittstelle bereit. Die IOleDsContainer-Schnittstelle definiert verschiedene Attribute, die mit Behälterobjekten und Verfahren für manipulierte enthaltene Objekt in Beziehung stehen. Außerdem kann jedes OleDs-Objekt optional eine Schnittstelle benannt IOleDs<class> oder IOleDs<class>operations bereitstellen, wo <class> dem Namen der Objektklasse des entsprechenden Objekts in dem Verzeichnisdienst entspricht, und <class>operations den Operationen entspricht, die auf einem Objekt durchgeführt werden können. Zum Beispiel hat ein Punktqueue-Objekt Operationen wie Pausieren, Starten und Löschen eines Druckauftrags. Im Folgenden werden diese und andere OleDs-Schnittstellen definiert, zusammen mit einer Beschreibung einer Beispielimplementierung von bestimmten Verfahren der Schnittstellen.

[0047] **Fig.** 4 ist ein Blockschaltbild eines zum Implementieren der vorliegenden Erfindung konfigurierten Computersystems. Das Computersystem **400** umfasst einen Speicher **401**, eine zentrale Verarbeitungseinheit **402**, eine E/A-Schnittstelle **403**, Speichereinrichtungen **404** und eine Anzeigeeinrichtung **405**. Verschiedene Implementierungen der OleDs-Objekte werden in einem computerlesbaren Speicher, z.B. einer Platte, gespeichert. Diese Implementierungen werden zum Ausführen in den Computerspeicher geladen. Der Speicher enthält auch eine Implementierung der OleDs-Objekte **408**, **411**, **414** für drei Verzeichnisdienste. Jede Implementierung der OleDs-Objekte greift auf den entsprechenden API-Satz der Verzeichnisdienste **409**, **412**, **415** zu. Diese API-Sätze stellen Zugriff auf die Merkmale der Objekte **410**, **413**, **416** für den zugrunde liegenden Verzeichnisdienst bereit. Außerdem stellt das OleDs selbst Implementierungen der Bindefunktion, des Namensraum-Behälters und verschiedener OleDs-Objekte bereit, die typischen Objekten von Verzeichnisdiensten entsprechen.

OleDsGetObiect-Funktion

[0048] **Fig.** 5 ist ein Flussdiagramm der Funktion OleDsGetObject, die die Bindefunktion des OleDs ist. Der Funktion OleDsGetObject werden ein OleDs-Pfad zu einem Objekt und ein Schnittstellen-Identifikator übergeben, und sie gibt einen Zeiger auf die identifizierte Schnittstelle für ein OleDs-Objekt, das dieses Objekt darstellt, zurück. In Schritt **501** teilt die Funktion den Namensraum-Identifikator von dem OleDs-Pfad. Die Funktion benutzt den Namensraum-Identifikator, um die Implementierung einer von dem Anbieter des identifizierten Namensraumes bereitgestellten Parsing-Schnittstelle zu lokalisieren. In Schritt **502** ruft die Funktion eine Funktion CoGetClassObject (definiert durch OLE **2**.01) auf, um einen Zeiger zu einem Objekt einer Klasse Fabrik für eine von dem Anbieter des identifizierten Namensraumes implementierte ParseDisplayName-Klasse zurückzugewinnen. Die Funktion erzeugt dynamisch den Klassenidentifikator der ParseDisplayName-Klasse für den identifizierten Namensraum. Wenn z.B. der geparste Namensraum "WinNTDS" ist, kann der Klassenidentfikator "CLSID_ PNDWinNTDS" sein. Das Klassefabrik-Objekt richtet ein Objekt der ParsedDisplayName-Klasse ein, das die einzurichtende IParseDisplayName-Schnittstelle enthüllt. Die IParseDisplayName-Schnittstelle liefert ein Verfahren zum Zurückgeben eines Monikers an das OleDs-Objekt für das durch den OleDs-Pfad identifizierte Objekt. In Schritt **503** ruft die Funktion das Verfahren IClassFactory::CreateInstance durch Übergeben des Identifikators der IParseDisplayName-Schnittstelle auf und empfängt einen Zeiger auf die IParseDisplayName-Schnittstelle. Das Verfahren CreateInstance erzeugt eine Instanz der durch den dynamisch erzeugten Klassenidentifikator identifizierten Klasse. In Schritt **504** ruft die Funktion das Verfahren IParseDisplayName::ParseDisplayName durch Übergeben des OleDs-Pfades auf. Das Verfahren ParseDisplayName gibt ei-

nen Zeiger auf einen Moniker (IMoniker-Schnittstelle) für das durch den OleDs-Pfad identifizierte Objekt zurück. In Schritt **505** ruft die Funktion das Verfahren IMoniker::BindToObject auf, um einen Zeiger auf das OleDs-Objekt, das das durch den OleDs-Pfad identifizierte Objekt darstellt, zurückzugewinnen. Die Funktion gibt dann den Zeiger auf das OleDs-Objekt zurück.

IOleDs-Schnittstelle

[0049] Die IOleDs-Schnittstelle definiert Attribute und Verfahren, die jedem OleDs-Objekt gemeinsam sind. Das Folgende ist die Schnittstellendefinition zusammen mit einer Beschreibung ihrer Attribute und Verfahren:

```
Interface IOleDs : IDispatch
    {   HRESULT get_Name (string *pName)
        HRESULT get_Class (string *pClass);
        HRESULT get_GUID (string *pGUID);
        HRESULT get_OleDsPath (string *pOleDsPath);
        HRESULT get_Parent (string *pParentContainer);
        HRESULT get_Schema (string *pSchemaClassObject);

        HRESULT Access (IOleDsAccess **ppComponentAccessControl);
        HRESULT PropAccess (string PropName, IOleDsAccess **ppComponentAccessControl);
        HRESULT GetInfo (variant vHints);
        HRESULT SetInfo (void);
        HRESULT Get (string Name, variant *pProp);
        HRESULT Put (string Name, variant Prop);
    }
```

Attribute

[0050] Name Beschreibung.

[0051] Name Relativer Name dieses Objekts in dem Behälterobjekt

[0052] OleDsPath OleDs-Pfad dieses Objekts

[0053] Class Name der Objektklasse dieses Objekts

[0054] GUID Eindeutiger Identifikator für Objekte dieser Objektklasse

[0055] Parent OleDs-Pfad des Behälterobjekts dieses Objekts

[0056] Schema OleDs-Pfad des Schemaobjekts, das diese Objektklasse darstellt

Verfahren

[0057] Name Beschreibung

[0058] Access Erlangt die IOleDsAccess-Schnittstelle (unten beschreiben) des OleDs-Zugriffssteuerobjekts, das den Sicherheitserlaubnissen dieses Objekts entspricht.

[0059] PropAccess Gibt einen Zeiger auf die OleDsAccess-Schnittstelle des abhängigen OleDs-Zugriffssteuerobjekts auf ein Merkmal davon darstellt, zurück.

[0060] GetInfo Gewinnt die Merkmalwerte dieses Objektes aus dem Verzeichnisdienst zurück und speichert sie in dem OleDs-Objekt. Der vHints-Parameter erlaub einem Client, anzugeben, welche Funktionssätze zurückgewonen werden sollten, sodass das Verfahren den Netzwerkzugriff optimieren kann.

[0061] SetInfo Macht Änderungen an diesen Objekt. Wenn Merkamle dieses Objekts geändert wurden, schreibt das Verfahren die Merkmale aus dem OleDs-Objekt in den Verzeichnisdienst.

[0062] Get Gewinnt den Wert für ein benanntes Merkaml aus dem OleDs-Objekt zurück.

[0063] Put Setzt den Wert für ein benanntes Merkaml in das OleDs-Objekt zurück.

[0064] **Fig.** 6 und 7 veranschaulichen Implementierungen von Verfahren der IOleDs-Schnittstelle. **Fig.** 6 ist ein Flussdiagramm des Verfahrens IOleDs::GetInfo. Dieses Verfahren lädt die Merkmale für das Objekt aus dem Verzeichnisdienst in das OleDs-Objekt. OleDs definiert einen Übergebungs- oder Transaktionsprozess, durch den Aktualisierungen an Merkmalen des Objekts nur in dem OleDs-Objekt gespeichert (mit dem Verfahren Put) werden, und nur in dem Verzeichnisdienst gespeichert werden, wenn ein Client es verlangt (mit dem Verfahren SetInfo). Wenn ein OleDs-Objekt erstmals eingerichtet wird, z.B., wenn die Funktion OeDsGetObject aufgerufen wird, würde das eingerichtete Objekt typischerweise die Handhabe enthalten, die das Objekt für den API-Satz des Verzeichnisdienstes identifiziert. In Schritt **601** ruft das Verfahren die Funktion ReadObject durch Übergeben einer Handhabe zu dem Objekt und einer Liste von Merkmalnamen für das Objekt auf. Die Funktion Read-Object gewinnt die Werte für die benannten Merkmale aus dem Verzeichnisdienst zurück. In einer Implementierung des Verfahrens GetInfo für ein OleDs-Vorgabeobjekt gewinnt das Verfahren eine Liste der Merkmalnamen aus dem Schemaobjekt für die Objektklasse dieses Objekts zurück. In Schritten 60204 durchläuft das Verfahren eine Schleife, um die rückgewonnenen Werte in dem OLeDs-Objekt zu speichern. In Schritt **602** wählt das Verfahren den nächsten Wert beginnend mit dem ersten aus. Wenn in Schritt **703** alle Werte bereits ausgewählt wurden, kehrt das Verfahren zurück, ansonsten fährt das Verfahren bei Schritt **604** fort. In Schritt **604** speichert das Verfahren den ausgewählten Wert in dem OleDs-Objekt und springt zurück zu Schritt **602**, um den nächsten Wert auszuwählen.

[0065] **Fig.** 7. ist ein Flussdiagramm des Verfahrens IOleDs::Get. Dem Verfahren wird ein Merkmalname übergeben, es gewinnt den Wert des übergebenen Merkmals aus dem OleDs-Objekt zurück und kehrt zurück.

IOleDsContainer-Schnittstelle

[0066] Die IOleDsContainer-Schnittstelle definiert Attribute und Verfahren, die jedem OleDs-Objekt, das ein Behälter ist, gemeinsam sind. Das Folgende ist die Schnittstellendefinition zusammen mit einer Beschreibung ihrer Attribute und Verfahren.

```
Interface IOleDsContainer: IDispatch
{
        HRESULT get_Count (long *pCount);
        HRESULT get_NewEnum (IUnknown *ppEnum);
        HRESULT get_Filter (Variant *pFilter);
        HRESULT put_Filter (Variant Filter);
        HRESULT GetObject (string Class, String RelativeName, IOleDs **ppNamedObject);
        HRESULT Create (string Class, string RelativeName, IOleDs **ppNewObject);
        HRESULT Delete (string Class, string RelativeName);
        HRESULT CopyHere (string SourceObject, string NewName, IOleDs **ppNewObject);
        HRESULT MoveHere (string SourceObject, string NewName, IOleDs **ppNewObject);
}
```

Attribute

[0067] Name Beschreibung

[0068] Filter Anordnung von Filtern für die Objektlasse, die in einer gegebenen Aufizählung zurückgegeben wird. Wenn Filter leer ist, werden alle Objekte aller Objektklassen zurückgegeben. Jeder Anordnungseintrag hat das folgende Format

<FilterEntry>::=<ClassName>

|<ClassName>.<PropName>

|<ClassName>.<FuncSetName>

|<ClassName>.<FuncSetName>.<PropName>

|<GUID>

|"Provider Specific String"

[0069] Count Anzahl von OleDs-Objekten in dem Behälter, die das Filter passieren.

[0070] NewEnum Aufzähler von enthaltenen Objekten.

[0071] Verfahren

[0072] Name Beschreibung

[0073] GetObject Gibt die IOleDs-Schnittstelle des Objekts in diesem durch die Objektklasse definierten Behälterobjekt und den relativen Namen in dem Behälterobjekt zurück. Wenn die Objektklasse nicht übergeben wird, gibt das Verfahren die Schnittstelle für das erste mit diesem relativen Namen gefundene Objekt zurück.

[0074] Create Erzeugt ein Objekt der spezifizierten Objektklasse und relativen Namen in diesem Behälterobjekt und gibt einen Zeiger auf die IOleDs-Schnittstelle zurück. Das Objekt wird erst wirklich in dem Verzeichnisdienst erzeugt, wenn das Verfahren IOleDs::SetInfo aufgerufen wird, sodass die verbindlichen Merkmale festgelegt werden können.

[0075] Delete Löscht das durch die Objektklasse und den relativen Namen identifizierte Objekt in diesem Behälterobjekt.

[0076] CopyHere Erzeugt ein neues Objekt in diesem Behälterobjekt, das mit dem spezifizierten Objekt identisch ist, und gibt einen Zeiger auf die IOleDs-Schnittstelle zu dem neuen Objekt zurück. NewName ist ein optionaler Parameter, der, wenn vorden, den Namen des neuen Objekts in dem Behälterobjekt enthält.

[0077] MoveHere Wie Verfahren CopyHere, außer dass das Quellenobjekt nach dem Kopieren gelöscht wird.

[0078] **Fig.** 8 ist ein Flussdiagramm des Verfahrens IOleDsContainer::CopyHere. Dieses Verfahren erzeugt eine Kopie des spezifizierten Quellenobjekts innerhalb dieses Behälterobjekts. In Schritt **801** bestimmt das Verfahren die Objektklasse des zu kopierenden Objekts. In Schritt **802** erzeugt das Verfahren ein Objekt in dem Verzeichnisdienst der bestimmten Objektklasse durch Aufrufen der Funktion CreateObject. In Schritt **803** gewinnt das Verfahren den OleDs-Pfad zu dem Schemaobjekt für diese Objektklase zurück. In Schritt **804** ruft das Verfahren die Funktion OleDsGetObject durch Übergeben des OleDs-Pfades zu dem Schemaobjekt auf und gewinnt einen Zeiger auf die IOeDs-Schnittstelle des OleDs-Schemaobjekts zurück. In Schritt **805** ruft das Verfahren das Verfahren IOleDs::QueryIntertace des OleDs-Schemaobjekts auf, um einen Zeiger auf die IOleDsClass-Schnittstelle zurückzugewinnen. Die IOleDsClass-Schnittstelle liefert Verfahren zur Rückgewinnung des Namens der für diese Objektklasse definierten Merkmale. In Schritt **806** ruft das Verfahren das Verfahren IOleDs-Class::GetMandatoryProperties auf, um eine Liste der obligatorischen Merkmale dieser Schemaklasse zurückzugewinnen. In Schritt **807** ruft das Verfahren das Verfahren IOle-Ds::Properties auf, um alle zusätzlichen Merkmale für diese Klasse zurückzugewinnen. In Schritten 808-811 durchläuft das Verfahren eine Schleife, um jeden der Werte der Merkmale aus dem Quellen-OleDs-Objekt in das Ziel-OleDs-Objektzu kopieren. In Schritt **808** wählt dasVerfahren das nächste Merkmal beginnend mit dem ersten aus. Wenn in Schritt **809** allle Merkmale bereits ausgewählt wurden, fährt das Verfahren in Schritt **812** fort, ansonsten fährt das Verfahren in Schritt **810** fort. In Schritt **810** ruft das Verfahren das Verfahren IOleDs::Get des Quellen-OleDs-Objekts auf, um den Wert für das ausgewählte Merkmal zurückzugewinnen. In Schritt **811** speichert das Verfahren den rückgewonnenen Wert in dem Ziel-IOieDs-Objekt und springt zurück zu Schritt 808, um das nächste Merkmal zu wählen. In Schritt **812** kopiert das Verfahren die Attribute des Quellen-OleDs-Objekts in das Ziel-OleDs-Objekt und kehrt zurück.

[0079] **Fig.** 9 ist ein Flussdiagramm des Verfahrens IOleDs::Create. In Schritt **901** ruft das Verfahren die Funktion CreateObject des Verzeichnisdienstes durch Übergeben einer Objektklasse auf und empfängt eine Handhabe zu einem neu erzeugten Objekt dieser Objektklasse.

IOleDsClass-Schnittstelle

[0080] Die IOleDsClass-Schnittstelle wird durch ein OleDs-Schemaobjekt enthüllt, um Zugriff auf die Definition der Objektklasse bereitzustellen. Das Folgende ist die Schnittstellendefintion zusammen mit einer Beschreibung ihrer Attribute und Verfahren.

[0081] Schnittstelle IOleDsClass: IOleDS

```
HRESULT get_PrimaryInterface (string *GUID);
HRESULT get_CLSIC (string CLSID);
HRESULT put_CLSID (string CLSID);
HRESULT get_OID (string *OID);
HRESULT put_OID (string OID);
HRESULT get_Abstract (boolean *Abstract);
HRESULT put_Abstract (boolean Abstract);
HRESULT get_MandatoryProperties (Variant *Mandatory);
HRESULT put_MandatoryProperties (Variant Mandatory);
HRESULT get_DerivedFrom (Variant *pDerivedFrom);
HRESULT put_DerivedFrom (Variant DerivedFrom);
HRESULT get_Containment (Variant *pContainment);
HRESULT put_Containment (Variant Containment);
HRESULT get_Container (boolean *pContainer);
HRESULT put_Container (boolean Container);
HRESULT get_HelpfileName (string *pHelpFile);
HRESULT put_HelpFileName (string HelpFile);
HRESULT get_HelpFileContext (long *pHelpContext);
HRESULT put_HelpFileContext (long HelpContext);
HRESULT Attributes (IOleDsCollection **ppAttributes)
```

Attribute

[0082] Name Beschreibung

[0083] CLISD CLISD des Codes, der das OleDs-Objekt für diese Objektklasse implementiert.

[0084] OID Namensraumspezifischer Identifikator, der diese Objektklasse definiert. Dieser wird bereitgestellt, um Schemaerweiterung über OleDs in Namensräumen zu erlauben, die namensraumspezifische OIDs für Objektklassen bentötigen.

[0085] Abstrakt Boolescher Wert, der angibt, ob diese Objektklasse abstrakt ist.

[0086] Mandatory Properties Liste der Merkmale für diese objektklasse festgelegt werden müssen, um in einen Speicher geschrieben zu werden.

[0087] Primary Interface der Primärschnittstellen-Identifikator für Objekte dieser Objektklasse. Dieser ist der IID für die Schnittstelle, die die Klasse definiert, z.B. die "Benutzer"-Klasse, wenn durch unterstützende IOleDsUser definiert.

[0088] DerivedFrom Anordnung von OleDs-Pfadketten, die die direkten Überklassen angeben, aus denen dieses Objekt abgeleitet wurde.

[0089] Container Merkmal, das bestimmt, dass diese Objektklasse ein Behälter ist.

[0090] HelpFileName Name einer Hilfedatei, die weitere iformation über Objekte dieser Objektklasse enthält.

[0091] HelpFileContext Context-ID in HelpFileName, wo spezifische Information über diese Objektklasse gefunden werden kann.

Verfahren

[0092] Name Beschreibung

[0093] Attributes gibt eine Sammlung von OleDs-objekten zurück die zusatzliche attribute dieses Merkmals beschreiben Attributobjekte sind anbieterspezifisch.

IOleDsProperty-Schnittstelle

[0094] OleDsdefiniert ein OleDs-Merkmalobjekt, das jedem Merkmal einer Objektklasse entspricht. Ein

OleDs-Merkmalobjekt enhüllt eine Schnittstelle zur Rückgewinnung der Definition des Merkmals. Das Folgende ist die Definition der Schnittstelle zusammen mit einer Beschreibung der Attribute und Verfahren.

interface IOleDsProperty:IOleDs

```
{
    HRESULT get_OleDsNames (Variant *pOleDsNames);
    HRESULT get_DsNames (Variant *pDsName);
    HRESULT get_OID (string OID);
    HRESULT put_OID (string *OID);
    HRESULT get_Syntax (string *pSyntax);
    HRESULT put_Syntax (string Syntax);
    HRESULT get_MaxRange (long *pMaxRange);
    HRESULT put_MaxRange (long MaxRange);
    HRESULT get_MinRange (long *pMinRange);
    HRESULT put_MinRange (long MinRange);
    HRESULT get_MultiValued (long *pMultiValued);
    HRESULT put_MultiValued (long MultiValued);
    HRESULT Attributes (IOleDsCollection **ppAttributes)
};
```

Attribute

[0095] Name Beschreibung

[0096] OleDsNames Anordnung von Zeichenketten, die die Namen enthalten, durch die OleDs auf dieses Merkmal zugreifen kann.

[0097] DsNames Anordnung von Zeichenketten, die die Namen enthalten, durch die der zugrunde liegende Namensraum auf dieses Merkmal zugreifen kann.

[0098] OID Der namensraumspezifische Objektidentifikator, der dieses Merkmal definiert.

[0099] Syntax Relativer Pfad des Schema-Syntaxobjekts, das die Syntax dieses Merkmals definiert. Relativ zu dem aktuellen Schemabehälter.

[0100] MaxRange Obergrenze der dem Merkmal zugewiesenen Werte.

[0101] MinRange Untergrenze der dem Merkmal zugewiesenen Werte.

[0102] Normal Wert, der bestimmt, ob dieses Merkmal normal wiederholt werden sollte.

[0103] MultiValued Wert, der bestimmt, ob dieses Merkmal mehrwertig ist.

Verfahren

[0104] Name Beschreibung

[0105] Attribute Gibt eine Sammlung von OleDs-Objekten zurück, die zusätzliche Attribute dieses Merkmals beschreiben.

IOleDs<Class> - Schnittstelle

[0106] OleDs definiert, dass jede Objektklasse eine Schnittstelle mit Namen IOieDs<class> unterstützen kann, wo <class> der Name der Objektklasse ist. Diese Schnittstelle hat ein Verfahren, wobei jeder Funktionssatz durch die Objektklasse unterstützt wird, das einen Zeiger auf ein OleDsFunctionalSet-Objekt für den Funktionssatz zurückgibt. Das Folgende ist eine Beschreibung einer Muster-Funktionssatzschnittstelle.

[0107] Schnittstelle IOleDsUser: IOleDs

```
{
    HRESULT get_Businessinfo
    (IOleDs FS UserBusinessInfo **ppFuncSet);
};
```

OleDs Obiektklassendefinitionen

[0108] OleDs definiert verschiedene Objektklassen für Behälter- und Blattobjekte. Die folgenden Objektklassen werden für Behälterobjekte durch OleDs definiert:

Namensräume
Namensraum
Land
Lokalität
Organisation
Bereich
Organisationseinheit
Computer
Dateiservice

[0109] Die folgenden Objektklassen werden von OleDs für Blattobjekte definiert:

Benutzer
Gruppe
Alias
Service
Druckschlange
Druckauftrag
Druckeinrichtungssitzung
Ressource
Dateisharing

[0110] Das Namensräumeobjekt ist ein Behälter der Namensraumobjekte. Die Implementierung des Namensräumeobjekts ist Teil von OleDs. Das Namensräumeobjekt enthüllt die IOleDs-Namensräume-Schnittstelle. Diese Schnittstelle wird unten gezeigt.

[0111] Schnittstelle IOleDsNamespaces: IolsDs

```
{
    get_DefaultContainer (string * pDefault);
    get_DefaultContainer (string Default);
}
```

[0112] Das Namensraumobjekt ist ein Behälterobjekt, das die Quelle aller OleDs-Objekte für einen gegebenen Namensraum ist. Das Länderobjekt ist ein Behälterobjekt mit Merkmalen, die ein Land betreffen. Das Lokalitätsobjekt ist ein Behälterobjekt mit Merkmalen, die eine Lokalität betreffen (z.B. Staat). Das Folgende ist eine Defintion einer Schnittstelle zum Erlangen und Festlegen der Merkmale eines Lokalitätsobjekts.

[0113] Interface IOleDsFSLocalityGeneralInfo: IDispatch

```
{
        HRESULT get_Description (string *Description);
        HRESULT put_Description (string Description);
        HRESULT get_LocalityName (string LocalityName);
        HRESULT put_LocalityName (string LocalityName);
        HRESULT get_PostalAddress (string *pPostalAddress);
        HRESULT put_PostalAddress (string PostalAddress);
        HRESULT get_SeeAlso (VARIANT *pSeeAlso);
        HRESULT put_SeeAlso (VARIANT Also);
        HRESULT Access (IOleDsAccess **ppFuncSetAccessControl);
        HRESULT PropAccess (string PropName, IOleDsAccess **ppPropAccessControl);
};
```

Verfahren

[0114] Name Beschreibung

[0115] Access Gibt einen Zeiger auf die IOleDsAccess-Schnittstelle auf dem abhängigen Zugriffssteuerobjekt zurück, das die Zugriffssteuerung auf dieser Schnittstelle darstellt.

[0116] PropAccess Gibt einen Zeiger auf die IOleDsAccess-Schnittstelle auf dem abhängigen Zugriffssteuerobjekt zurück, das die Zugriffssteuerung auf ein Merkmal auf dieser Schnittstelle darstellt, wie durch PropName angegeben.

Attribute

[0117] Name Beschreibung

[0118] Description Text, der die Lokalität beschreibt.

[0119] LocalityName Lokalistätsname. Der Lokalistätsname identifiziert ein geographisches Gebiet, in dem sich der Behälter physikalisch befindet.

[0120] PostalAddress Haupt Postadresse der Lokalität.

[0121] SeeAlso Anordnung von Namen anderer Verzeichnisobjekte, die für dieses Objekt relevant sein können.

[0122] Jede der OleDs-definierten Objektklassen definiert Merkmale, die Objekte dieser Objektklasse betreffen. Das Bereichsobjekt entspricht einem Bereich in einem Computersystem und hat einen Funktionssatz mit Merkmalen (z.B. Passwörter), die sich auf das Bereichsobjekt beziehen. Das Organisationseinheitobjekt entspricht einer Wesenheit, z.B. eine Firma. Das Computerobjekt entspricht einem Computer in einem Bereich. Das Benutzerobjekt hat einen Geschäftsinformations-, Kontobeschränkungs-, Kontostatistik- und anderen Informationsfunktionssatz, um einen Benutzer zu beschreiben. Der Geschäftsinformations-Funktionssatz enthält Merkmale bezüglich der Beschreibung eines Geschäftes, z.B. Land, Division, Abteilung, Manager oder Dienststellen. Der Kontobeschränkungs-Funktionssatz enthält Merkmale, die die allgemeinem Eigenschaften des Benutzerkontos beschreiben, z.B. Konto gesperrt, Zahl von Stunden, die dem Benutzer zum Einloggen erlaubt sind, usw. Der Kontostatistik-Funktionssatz hat Merkmale, die Statistiken des Benutzerkontos spezifizieren, z.B. wann das Passwort zum letzten Mal geändert wurde, oder wann sich der Benutzer das letzte Mal in das Konto eingeloggt hat. Der andere Informations-Funktionssatz enthält Merkmale, die verschiedenartige Benutzerinformation betreffen, z.B. E-Mail-Adresse und Heimverzeichnis.

Beispiel-Client von OleDs

[0123] **Fig.** 10 ist ein Flussdiagramm der Prozedur zum Anzeigen aller Namensräume. Diese Prozedur veranschaulicht, wie ein Client von OleDs die Namen aller Namensräume rückgewinnen würde. In Schritt **1001** ruft die Prozedur die Funktion CoGetClassObject durch Übergeben der Klasse für das OleDs-Namensräumeobjekt und des Schnittstellen-Identifikators für die IClassFactory-Schnittstelle auf. Die Funktion gibt einen Zeiger auf eine Klasse-Fabrik-Schnittstelle für das OleDs-Namensräumeobjekt zurück. In Schritt **1002** ruft die Prozedur das Verfahren IClassFactory::CreateInstance durch Übergeben des Schnittstellen-Identifikators für die

IOleDsContainer-Schnittstelle auf. Das Verfahren erzeugt eine Instanz des OleDs-Namensräumeobjekts und gibt einen Zeiger auf die IOleDsContainer-Schnittstelle zurück. In Schritt **1003** ruft die Prozedur das Verfahren IOleDsContainer::get_NewEnum auf, um einen Aufzähler für das Namensräumeobjekt zurückzugewinnen. In Schritten **1004–1008** durchläuft die Prozedur eine Schleife, um jeden Namensraum aufzuzählen, und zeigt den Namensraum an. In Schritt **1004** ruft die Prozedur das Verfahren IEnumerator::Next auf, um einen Zeiger auf die IUnknown-Schnittstelle für ein Namensraumobjekt zurückzugewinnen. Wenn in Schritt **1005** alle Namensräume bereits aufgezählt wurden, kehrt die Prozedur zurück, andernfalls fährt die Prozedur in Schritt **1006** fort. In Schritt **1006** ruft die Prozedur das Verfahren IUnknown::QuenInterface auf, um einen Zeiger auf die IOleDs-Schnittstelle für das Namensraumobjekt zurückzugewinnen. In Schritt **1007** ruft die Prozedur das Verfahren IOleDs:: get narre auf, um den mit dem Namensraumobjekt verbundenen Namen zurückzugewinnen. In Schritt **1008** zeigt die Prozedur den rückgewonnenen Namen an und springt dann zurück zu Schritt **1004**, um das nächste Namensraumobjekt aufzuzählen.

[0124] Obwohl die vorliegende Erfindung in Form einer bevorzugten Ausführung beschrieben wurde, ist nicht beabsichtigt, die Erfindung auf diese Ausführung zu beschränken. Der Umfang der vorliegenden Erfindung wird in den folgenden Ansprüchen definiert.

Patentansprüche

1. Computersystem (**400**) zum Zugreifen auf eine Vielzahl verschiedener Verzeichnisdienste (**409**, **412**, **415**) in einer gleichmäßigen Weise, wobei jeder Verzeichnisdienst Objekte (**410**, **413**, **416**) besitzt, wobei jedes Objekt eine Objekt-Identifizierung, eine Objektklasse und einen Merkmalwert aufweist, wobei das Computersystem umfasst:
für jeden der Vielzahl von Verzeichnisdiensten,
eine Anwendungsprogrammierschnittstelle, wobei die Anwendungsprogrammierschnittstelle Bindefunktionen zum Binden an ein Objekt des Verzeichnisdienstes durch Empfangen einer Objekt Identifizierung und Zurückgeben eine Verweises auf das identifizierte Objekt und Schemafunktionen zum Rückgewinnen einer Definition jeder Objektklasse besitzt, und
eine Implementierung einer ersten und zweiten vordefinierten Schnittstelle, durch die ein Client (**406**, **407**) auf die Objekte des Verzeichnisdienstes in einer Weise zugreifen kann, die unabhängig von der Anwendungsprogrammierschnittstelle des Verzeichnisdienstes ist, wobei die erste Schnittstelle zum Zugreifen auf Merkmalwerte jedes Objekts ist, und die zweite Schnittstelle zum Zugreifen auf die Definition jeder Objektklasse ist;
eine Namensräume-Schnittstelle zum Aufzählen jedes Verzeichnisdienstes, und
einen Verzeichnisdienst-Browser zum Aufzählen der Verzeichnisdienste unter Verwendung der Namensräume-Schnittstelle und zum Verwenden der Implementierung der vordefinierten Schnittstellen für einen aufgezählten Verzeichnisdienst, um auf die Merkmalwerte von Objekten des Verzeichnisdienstes zuzugreifen, und um auf die Definitionen der Objektldassen des Verzeichnisdienstes zuzugreifen, wobei der Verzeichnisdienst-Browser unabhängig von jedem Verzeichnisdienst ist.

2. Computersystem nach Anspruch 1, wobei die Schnittstelle zum Zugreifen auf die Definition einer Objektklasse ein Verfahren zum Hinzufügen einer Definition einer Objektklasse zu dem Verzeichnisdienst einschließt.

3. Computersystem nach Anspruch 1 oder 2, wobei jeder Verzeichnisdienst eine Implementierung eines Vorgabe-In-Speicher-Objekts bereitstellt, um Objekte von neu hinzugefügten Objektklassen zu akkomodieren.

4. Computersystem nach einem der Ansprüche 1 bis 3, wobei jedes Objekt eine eindeutige Identifizierung hat, und das eine Bindefunktion enthält, die, wenn ihr eine Identifizierung eines Objekts übergeben wird, feststellt, welcher Verzeichnisdienst das identifizierte Objekt enthält, und eine In-Speicher-Datenstruktur zum Speichern der Merkmalwerte dieses Objekts erzeugt.

5. Verfahren in einem Computersystem (**400**) zum Zugreifen auf eine Vielzahl von Verzeichnisdiensten (**409**, **412**, **415**) in einer gleichmäßigen Weise, wobei jeder Verzeichnisdienst eine Anwendungsprogrammierschnittstelle besitzt, wobei die Anwendungsprogrammierschnitt stelle ein Verhalten zum Zugreifen auf Objekte (**410**, **413**, **416**) des Verzeichnisdienstes aufweist, wobei das Verfahren umfasst:
für jeden einer Vielzahl von Verzeichnisdiensten, Installieren einer Implementierung einer vordefinierten Schnittstelle, wobei die vordefinierte Schnittstelle ein Verhaften zum Zugreifen auf Objekte des Verzeichnisdienstes definiert, wobei die Implementierung eine Abbildung aus dem Verhalten der Anwendungsprogrammierschnittstelle auf das Verhalten der vordefinierten Schnittstelle bereitstellt;
Empfangen einer Identifikation eines Objekts;
Bestimmen, welcher Verzeichnisdienst das identifizierte Objekt enthält, und Verwenden der installierten Imp-

lementierung der vordefinierten Schnittstelle für den bestimmten Verzeichnisdienst zum Auffinden und Binden an das identifizierte Objekt.

6. Verfahren nach Anspruch 5, wobei jedes Objekt eine Objektklasse besitzt, und wobei die vordefinierte Schnittstelle ein Verhalten zum Hinzufügen einer Definition einer Objektklasse zu einem Verzeichnisdienst definiert.

7. Verfahren nach Anspruch 5 oder 6, wobei jedes Objekt eine Objektklasse besitzt, und wobei die vordefinierte Schnittstelle ein Verhalten zum Rückgewinnen einer Definition einer Objektklasse eines Verzeichnisdienstes definiert.

8. Verfahren nach einem der Ansprüche **5** bis **7**, wobei die vordefinierte Schnittstelle ein Verhalten zum Übergeben von Änderungen an Objekten an den Verzeichnisdienst definiert.

Es folgen 11 Blatt Zeichnungen

Anhängende Zeichnungen

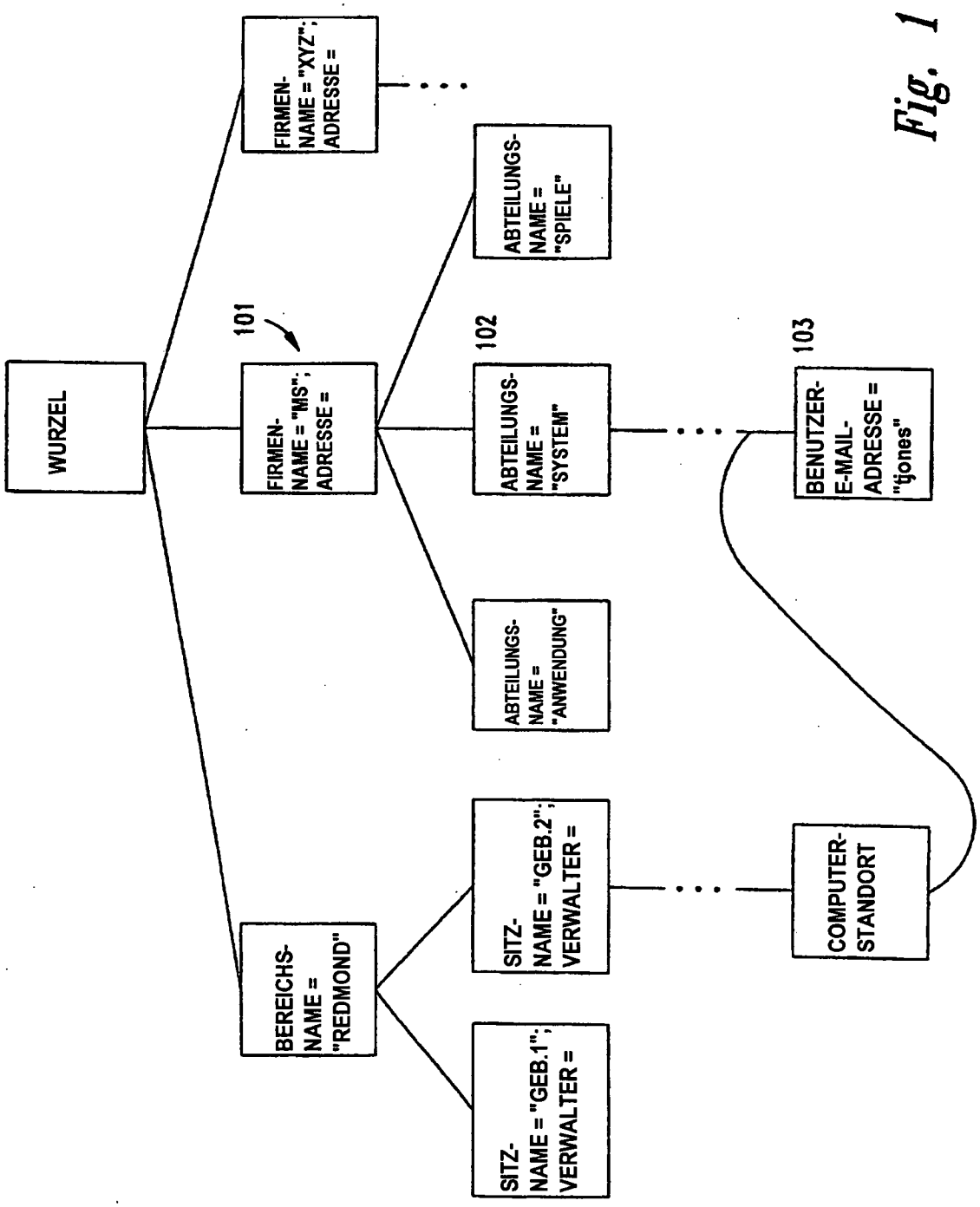

Fig. 1

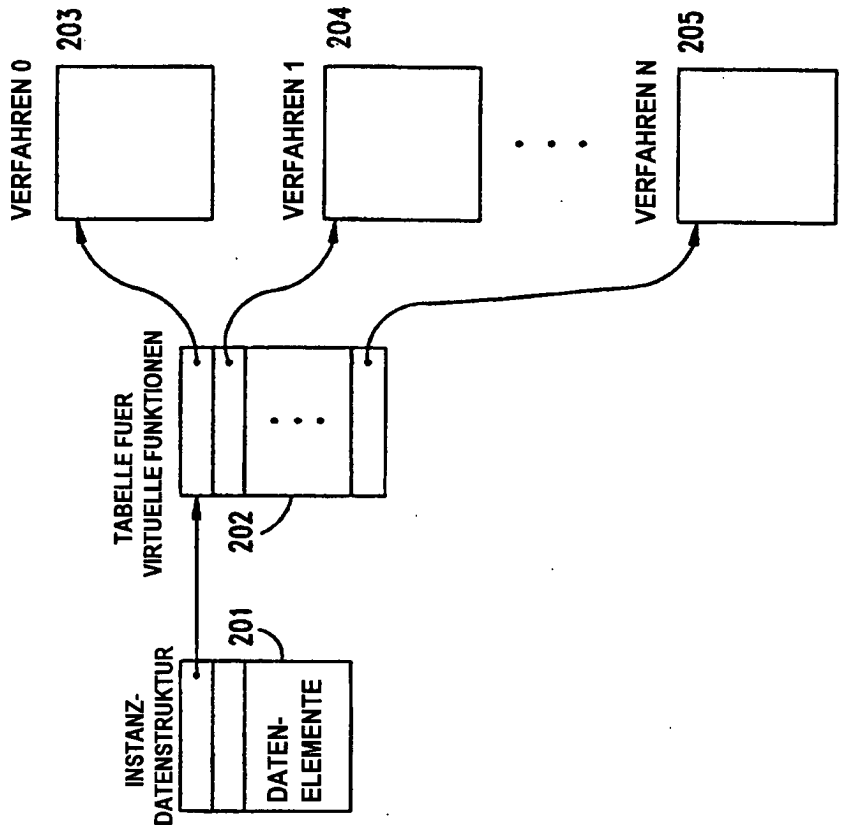
VERFAHREN 0
203
VERFAHREN 1
204
VERFAHREN N
205
TABELLE FUER VIRTUELLE FUNKTIONEN
202
INSTANZ-DATENSTRUKTUR
201
DATEN-ELEMENTE

Fig. 2

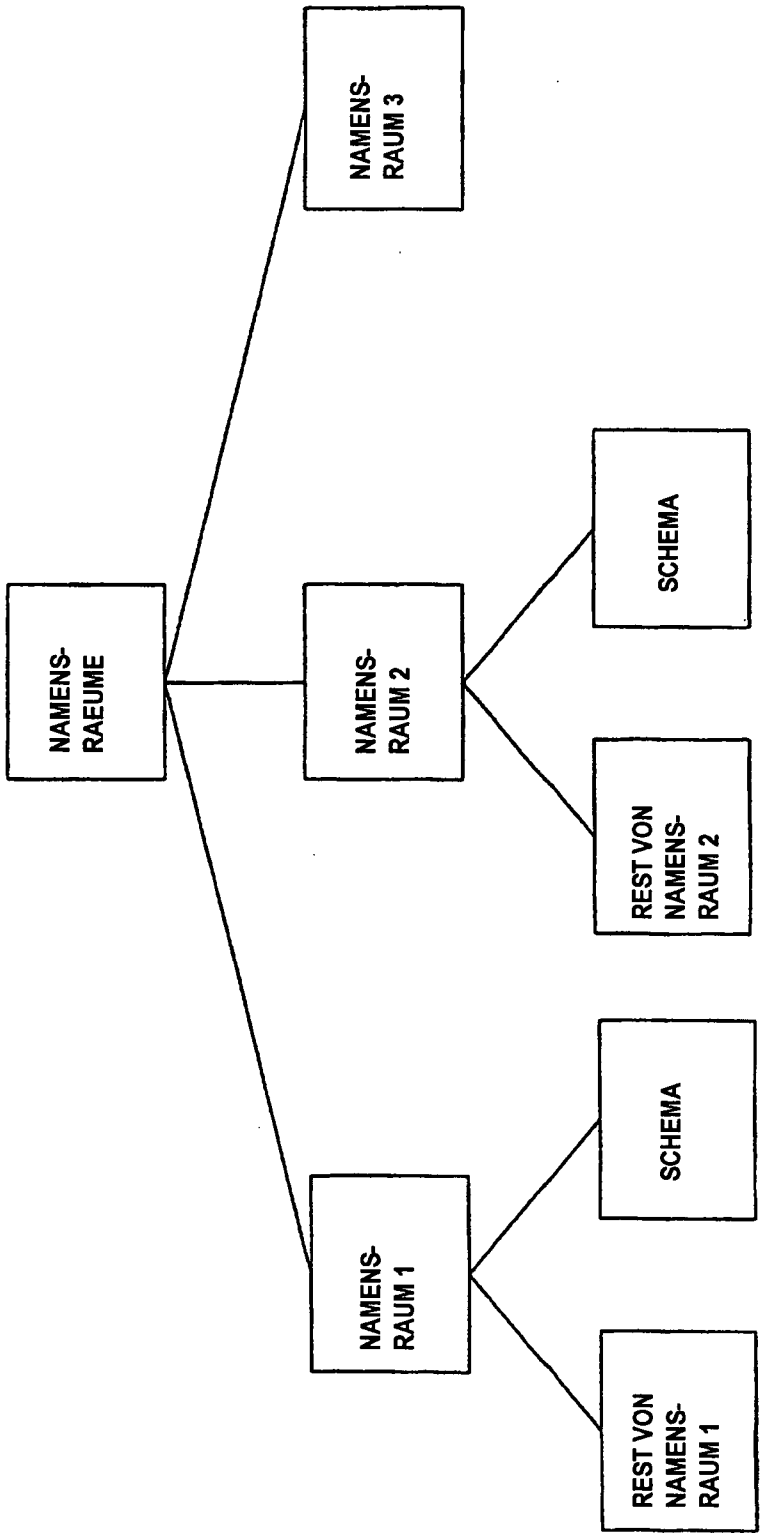
NAMENS-
RAEUME
NAMENS-
RAUM 3
NAMENS-
RAUM 2
SCHEMA
REST VON
NAMENS-
RAUM 2
NAMENS-
RAUM 1
SCHEMA
REST VON
NAMENS-
RAUM 1

Fig. 3A

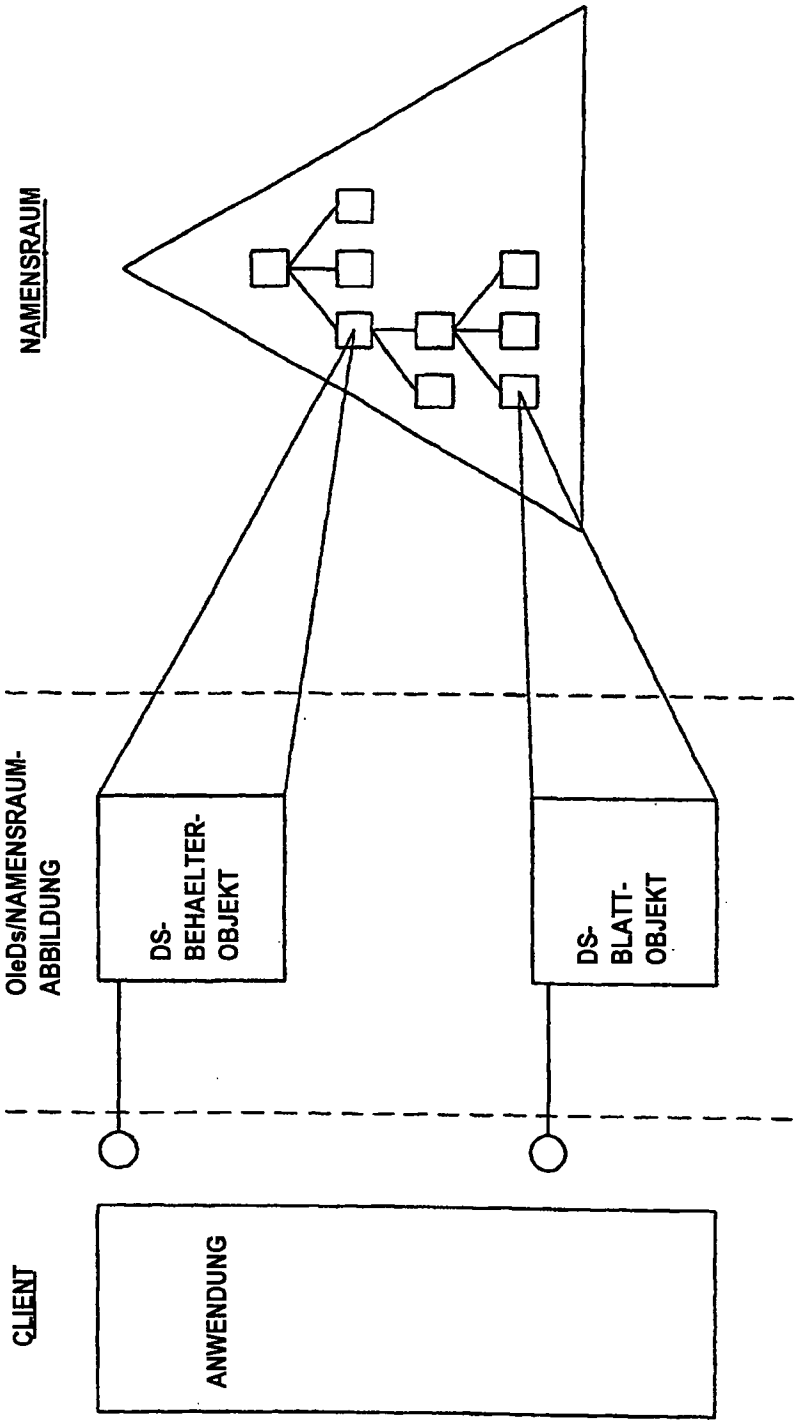
NAMENSRAUM
OleDs/NAMENSRAUM-ABBILDUNG
DS-BEHAELTER-OBJEKT
DS-BLATT-OBJEKT
CLIENT
ANWENDUNG

Fig. 3B

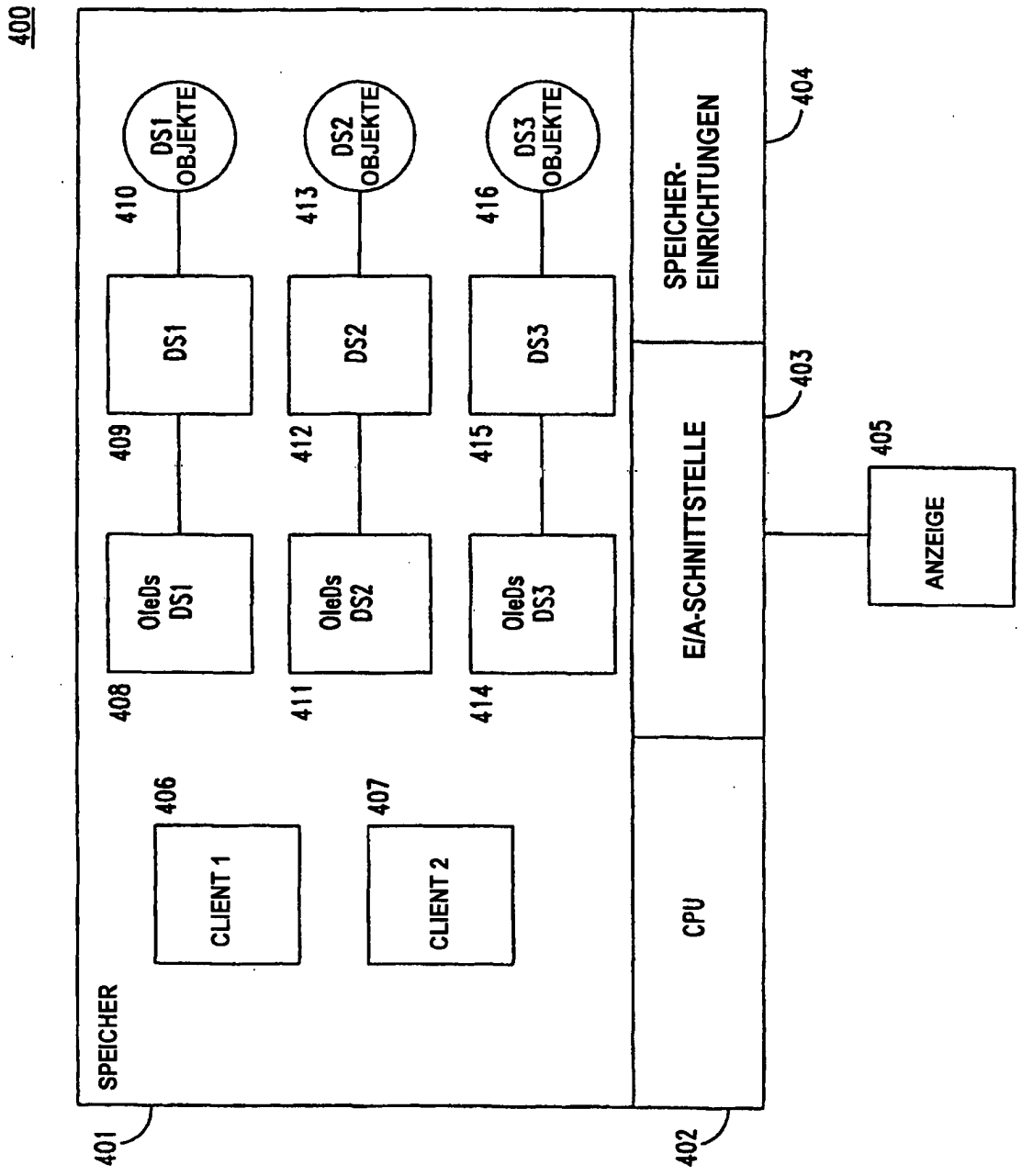
400
SPEICHER
401
CLIENT 1
406
CLIENT 2
407
OleDs DS1
408
DS1
409
DS1 OBJEKTE
410
OleDs DS2
411
DS2
412
DS2 OBJEKTE
413
OleDs DS3
414
DS3
415
DS3 OBJEKTE
416
CPU
402
E/A-SCHNITTSTELLE
403
SPEICHER-EINRICHTUNGEN
404
ANZEIGE
405

Fig. 4

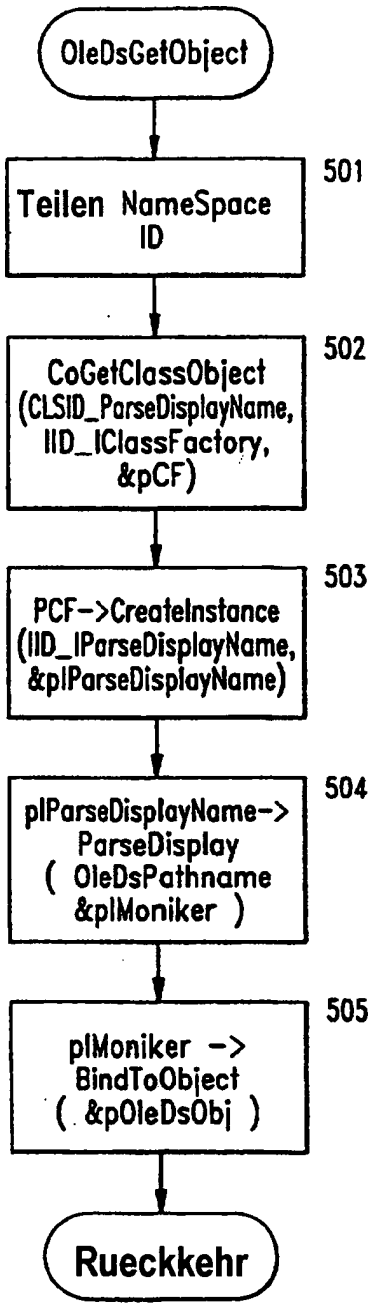
OleDsGetObject
Teilen NameSpace ID
501
CoGetClassObject (CLSID_ParseDisplayName, IID_IClassFactory, &pCF)
502
PCF->CreateInstance (IID_IParseDisplayName, &pIParseDisplayName)
503
pIParseDisplayName-> ParseDisplay (OleDsPathname &pIMoniker)
504
pIMoniker -> BindToObject (&pOleDsObj)
505
Rueckkehr

Fig. 5

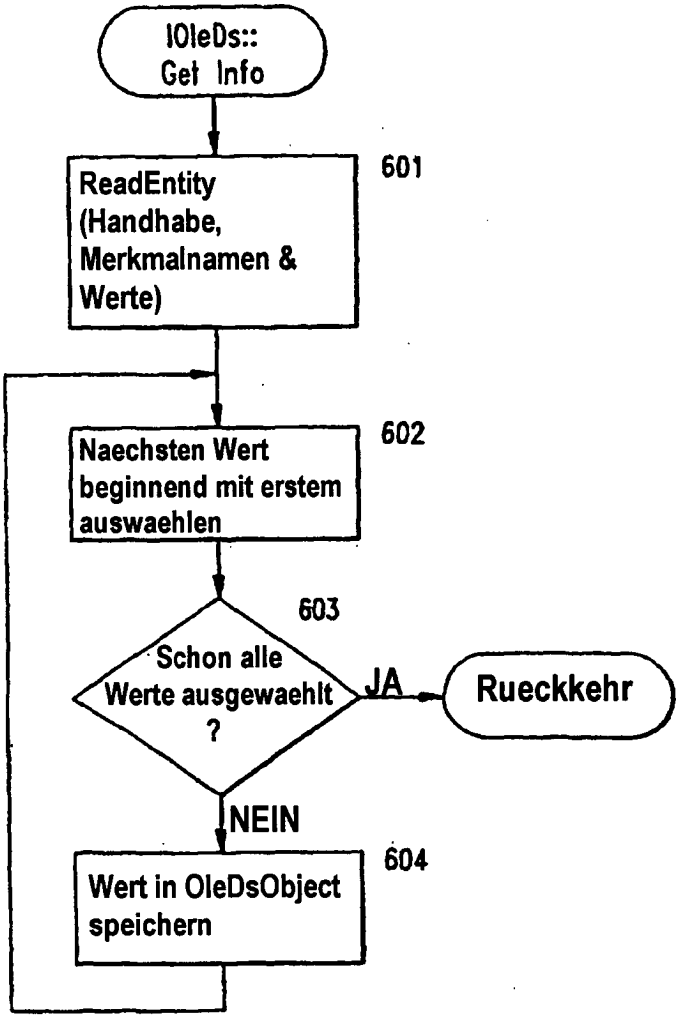
IOleDs::
Get Info
ReadEntity
(Handhabe,
Merkmalnamen &
Werte)
601
Naechsten Wert
beginnend mit erstem
auswaehlen
602
603
Schon alle
Werte ausgewaehlt
?
JA
Rueckkehr
NEIN
Wert in OleDsObject
speichern
604

Fig. 6

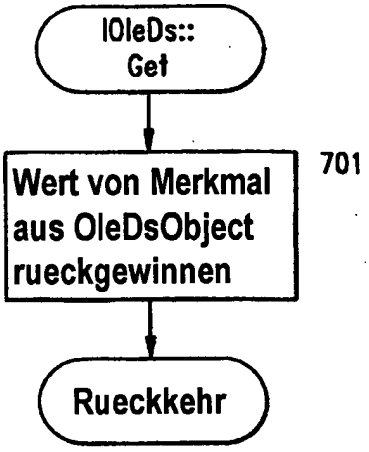
IOleDs::
Get
Wert von Merkmal aus OleDsObject rueckgewinnen
701
Rueckkehr

Fig. 7

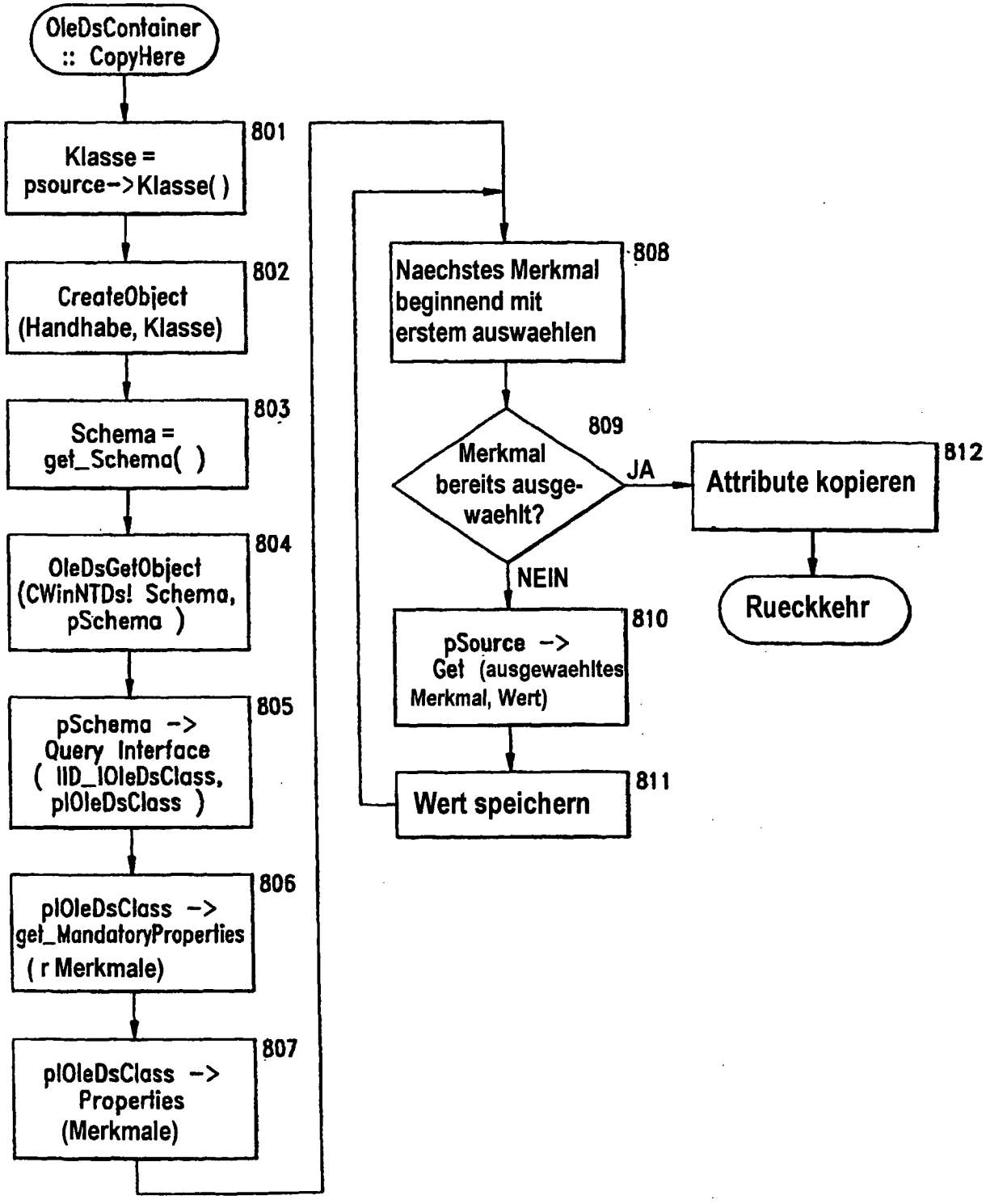
OleDsContainer :: CopyHere
Klasse = psource->Klasse( )
801
CreateObject (Handhabe, Klasse)
802
Schema = get_Schema( )
803
OleDsGetObject (CWinNTDs! Schema, pSchema)
804
pSchema -> Query Interface (IID_IOleDsClass, pIOleDsClass)
805
pIOleDsClass -> get_MandatoryProperties (r Merkmale)
806
pIOleDsClass -> Properties (Merkmale)
807
Naechstes Merkmal beginnend mit erstem auswaehlen
808
Merkmal bereits ausgewaehlt?
809
JA
NEIN
Attribute kopieren
812
Rueckkehr
pSource -> Get (ausgewaehltes Merkmal, Wert)
810
Wert speichern
811

Fig. 8

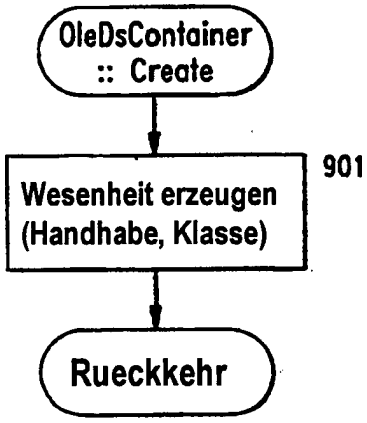
OleDsContainer
:: Create
Wesenheit erzeugen
(Handhabe, Klasse)
901
Rueckkehr

Fig. 9

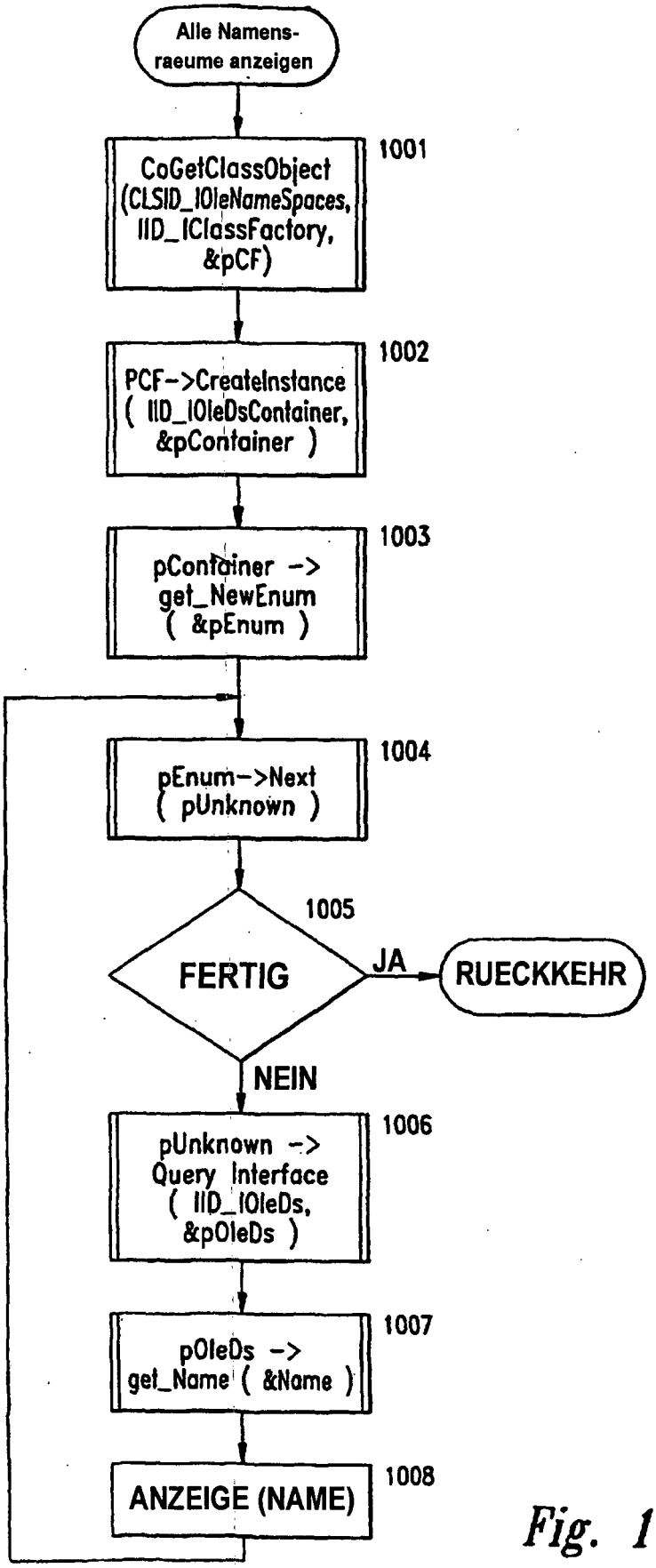
Alle Namens-raeume anzeigen
CoGetClassObject (CLSID_IOleNameSpaces, IID_IClassFactory, &pCF)
1001
PCF->CreateInstance (IID_IOleDsContainer, &pContainer)
1002
pContainer -> get_NewEnum (&pEnum)
1003
pEnum->Next (pUnknown)
1004
FERTIG
1005
JA
RUECKKEHR
NEIN
pUnknown -> Query Interface (IID_IOleDs, &pOleDs)
1006
pOleDs -> get_Name (&Name)
1007
ANZEIGE (NAME)
1008

Fig. 10