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

(10) **DE 699 36 627 T2** 2008.05.21

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 025 493 B1**
(21) Deutsches Aktenzeichen: **699 36 627.5**
(86) PCT-Aktenzeichen: **PCT/US99/18749**
(96) Europäisches Aktenzeichen: **99 942 278.5**
(87) PCT-Veröffentlichungs-Nr.: **WO 2000/010080**
(86) PCT-Anmeldetag: **17.08.1999**
(87) Veröffentlichungstag der PCT-Anmeldung: **24.02.2000**
(97) Erstveröffentlichung durch das EPA: **09.08.2000**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **25.07.2007**
(47) Veröffentlichungstag im Patentblatt: **21.05.2008**

(51) Int Cl.[8]: ***G06F 9/44*** (2006.01)

(30) Unionspriorität:
135378 **17.08.1998** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB, NL, SE

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

(74) Vertreter:
BOEHMERT & BOEHMERT, 28209 Bremen

(72) Erfinder:
DIEVENDORFF, Richard, Bellevue, WA 98007, US;
HELLAND, Patrick J., Bellevue, WA 98006, US;
CHOPRA, Gagan, Redmond, WA 98052, US;
AL-GHOSEIN, Mohsen, Redmond, WA 98053, US

(54) Bezeichnung: **IN EINER WARTESCHLANGE ANGEORDNETE AUFRUFE VON PROZEDUREN FÜR VERTEILTE AUF KOMPONENTEN BASIERTE ANWENDUNGEN**

Beschreibung

Technisches Gebiet

[0001] Die vorliegende Erfindung betrifft verteilte komponentenbasierte Computersoftware-Anwendungen, insbesondere in einer Warteschlange angeordnete Aufrufe von Prozeduren für solche Anwendungen.

Stand der Technik

[0002] In zahlreichen Informationsverarbeitungsanwendungen stellt eine auf einem Host- oder Server-Computer in einem verteilten Netzwerk laufende Server-Anwendung Verarbeitungsdienste für von einer Vielzahl von Benutzern betriebene, auf Terminal- oder Arbeitsstationscomputern des Netzwerks laufende Client-Anwendungen bereit. Bekannte Beispiele für entsprechende Serveranwendungen umfassen Software zum Verarbeiten von Kursanmeldungen an einer Universität, Reisereservierungen, Geldtransfers und anderen Bankdienstleistungen sowie Verkäufen in Unternehmen. In diesen Beispielen können die von der Serveranwendung bereitgestellten Verarbeitungsdienste Datenbanken von Stundenplänen, Hotelreservierungen, Kontoständen, Lieferaufträgen, Zahlungen oder Bestandsaufnahmen für von den Einzelbenutzern an den jeweiligen Standorten initiierte Aktionen aktualisieren. Dies wird gelegentlich als Client/Server-Datenverarbeitung bezeichnet.

[0003] In einer Art der Client/Server-Datenverarbeitung, die mitunter als "Verteilte Objekte" bekannt ist, wird die Serveranwendung als eine Zusammenstellung von Komponenten entwickelt, die einem Objektorientierten Programmierungsmodell (OOP-Modell), wie beispielsweise dem Komponenten-Objekt-Modell (Microsoft Component Objekt Model (COM)) und dem Verteilte Komponenten-Objekt-Modell (Distributed Component Object Model (DCOM)) von Microsoft, dem System-Objekt-Modell (IBM System Object Model (SOM)) von IBM, der Architektur für Vermittler der Abrufe von gemeinsamen Objekten (Object Management Group's Common Object Request Broker Architecture (CORBA)) der Object Management Group und anderen, entsprechen. Die Vorteile der objektorientierten Programmierung liegen im Allgemeinen in der Einfachheit der Programmierung, der Erweiterbarkeit, der Wiederverwendbarkeit von Code sowie der Möglichkeit der Integration von Software verschiedener Hersteller und (bei einigen objektorientierten Programmierungsmodellen) in verschiedenen Programmiersprachen.

[0004] In der objektorientierten Programmierung werden Programme als eine Zusammenstellung von Objektklassen geschrieben, welche jeweils reale oder abstrakte Elemente durch Kombinieren von Daten zur Abbildung von Eigenschaften des Gegenstands mit Prozeduren (z.B. Programmfunktionen oder -prozeduren) zur Abbildung der Funktionalität des Elements modellieren. Genauer gesagt ist ein Objekt eine als Klasse bezeichnete Instanz eines programmiererdefinierten Typs, welche die Eigenschaften Datenkapselung, Polymorphie und Vererbung darstellt.

[0005] Datenkapselung bezieht sich auf das Kombinieren von Daten (auch als Objekteigenschaften bezeichnet) und Prozeduren, die auf den Daten operieren (auch als Memberfunktionen eines Objekts bezeichnet), zu einer einheitlichen Softwarekomponente (d.h. dem Objekt), so daß das Objekt seine interne Zusammensetzung, Struktur und Operation verbirgt und seine Funktion den das Objekt nutzenden Client-Programmen gegenüber ausschließlich über eine oder mehrere Schnittstellen offenbart. Eine Schnittstelle des Objekts ist eine Gruppe semantisch verwandter Prozeduren des Objekts. Anders formuliert, greifen die Client-Programme nicht direkt auf die Objektdaten zu, sondern sie müssen vielmehr Prozeduren an den Objektschnittstellen abrufen, um auf den Daten operieren zu können.

[0006] Polymorphie bezieht sich auf die Fähigkeit, zwei gleichartige Objekte über eine gemeinsame Schnittstelle zu betrachten (d.h. damit zu interagieren), wodurch sich die Notwendigkeit, zwischen zwei Objekten zu differenzieren, erübrigt. Vererbung bezieht sich auf das Ableiten verschiedener Objektklassen von einer Basisklasse, wobei die abgeleiteten Klassen die Eigenschaften und Besonderheiten der Basisklasse erben.

[0007] Bei Client/Server-Datenverarbeitung mit "verteilten Objekten" nutzt das Client-Programm auf dem Benutzercomputer in der Regel "Echtzeit"- oder synchrone Verarbeitungsmechanismen zum Fernaufrufen von Prozeduren auf den Server-Anwendungs-Objekten, die sich auf dem Server-Computer befinden, wie beispielsweise der Prozedur-Fernaufruf ("RPC"). Bei einem typischen Prozedur-Fernaufruf erstellen Objektdienste des Betriebssystems eine Schnittstellendefinitionssprachbeschreibung eines Serveranwendungsobjekts, um einen lokalen "Proxy" für das Serveranwendungsobjekt auf dem Client-Computer zu generieren. Die Client-Software ruft Prozeduren des entfernten Serveranwendungsobjekts auf, indem sie gewöhnliche lokale Aufrufe von Prozeduren direkt an den Proxy ausgibt. Der Proxy nutzt seinerseits RPC-Dienste, um den Prozedur-Aufruf an das

aktuelle Serveranwendungsobjekt auf dem entfernten Server-Computer weiterzuleiten. Die RPC-Dienste arrangieren Werte für Aufruf-Parameter zu einer Netzwerk-Nachricht und senden die Nachricht über Netzwerkprotokolle an den Server-Computer. Auf dem Server-Computer machen die RPC-Dienste das Arrangieren der Aufruf-Parameter rückgängig und geben den Aufruf an die richtige Serveranwendungsobjekt-Prozedur aus. Außerdem übernehmen die RPC-Dienste das Arrangieren und Rückgängigmachen des Arrangierens von Rückgabewerten von der Serveranwendungsobjekt-Prozedur zurück zum Client-Programm über eine Netzwerk-Nachricht.

[0008] Dementsprechend verarbeiten die RPC-Dienste alle Feinheiten der Netzwerkkommunikation gewissermaßen "hinter den Kulissen", so daß das Client-Programm die entfernte Prozedur auf die gleiche Weise wie einen lokalen Prozedur-Aufruf aufruft. Wie bei einem lokalen Prozedur-Aufruf wird die Ausführung des Client-Programms während des RPC-Prozedur-Aufrufs bis zum Abschluß und zur Rückkehr der Prozedur unterbrochen (auch bekannt als "Blockieren"). Dies bewirkt einen synchronen Ausführungsfluß zwischen dem Client-Programm und den Serveranwendungsobjekten.

[0009] Obzwar sich Echtzeit-Prozedur-Aufruf-Modelle, wie beispielsweise RPC, für viele Anwendungen eignen, können sie den Anforderungen anderer Anwendungen aufgrund einer Anzahl von Einschränkungen in Bezug auf Verfügbarkeit, Netzübertragungskosten, fehlende Möglichkeit der Priorisierung, Lebensdauer des Objekts und Referenzlokalität nicht adäquat entsprechen.

[0010] Bezüglich der Verfügbarkeit fordern Echtzeit-Prozedur-Aufruf-Modelle, daß das Serveranwendungsobjekt zum Zeitpunkt verfügbar ist, zu welchem das Client-Programm einen Aufruf an eine der Prozeduren des Objekts ausgibt. Ist dies nicht der Fall, kann der Echtzeit-Prozedur-Aufruf nicht erfolgen. In der Praxis können Serveranwendungsobjekte jedoch aufgrund von Netzwerkfehlern oder durch eine Arbeitsüberlastung des Server-Computers nicht verfügbar sein. In einigen Fällen kann der Server-Computer über einen bestimmten Zeitraum offline (beispielsweise aufgrund von Aktualisierungs- oder Wartungsmaßnahmen) oder nur zu bestimmten Tageszeiten online sein. Außerdem kann, wenn ein beliebiges Serveranwendungsobjekt nicht in "Echtzeit" verfügbar ist, kein Teil der Arbeit abgeschlossen werden (einschließlich der von verfügbaren Serveranwendungsobjekten). Dieses Problem verstärkt sich bei komplexen Operationen mit multiplen Knoten (z.B. Computer in einem Netzwerk). So ist beispielsweise ein Prozeß, der Zugang zu vier unabhängigen Objekten an separaten Knoten erfordert, wovon jeder ca. 90% der Zeit verfügbar ist, tatsächlich nur etwa 66% der Zeit verfügbar (da $90\%^4 = 65{,}61\%$).

[0011] Außerdem sind einige Client-Computer nur gelegentlich mit einem Netzwerk verbunden und somit einen Großteil der Zeit über nicht in der Lage, Echtzeit-Aufrufe von Prozeduren auszugeben. Ein treffendes Beispiels für solche gelegentlich verbundenen Client-Computer sind Laptops, Notebooks und Handheld-Computer mobiler Benutzer, deren Anteil bei Computerneuanschaffungen heute auf über 50% geschätzt wird.

[0012] Zur Verfügbarkeitsanforderung des Echtzeit-Prozedur-Aufruf-Modells gehört auch, daß der/die Server-Computer mit ausreichender Kapazität konfiguriert sein muß/müssen, um die Anfragen interaktiver Benutzer der Server-Anwendung zu Spitzenzeiten bewältigen zu können. Demzufolge ist das typische System mit Server-Computern konfiguriert, welche die meiste Zeit über nicht ausgelastet, zu bestimmten Zeiten allerdings überlastet sind (wenn z.B. die tatsächliche Belastung die Erwartungen überschreitet).

[0013] Aus den genannten Gründen ist das Echtzeit-Prozedur-Aufruf-Modell für Datenverarbeitungsumgebungen mit eingeschränkter Verfügbarkeit, mobilen Benutzern, hohen Knotenzahlen oder stark variierenden Lasten durch interaktive Benutzer ungeeignet.

[0014] Was die Netzübertragungskosten betrifft, so erfordert jeder Echtzeit-Prozedur-Aufruf über einen RPC eine Hin-und-Rück-Netzwerkkommunikation, die zu beachtlichen Netzübertragungskosten führt, beispielsweise durch die Bearbeitungszeit für das Arrangieren der Aufruf-Parameter und das Rückgängigmachen des Arrangierens, das Verarbeiten im Netzwerkprotokoll-Stack sowie Übertragungszeit. Überdies kann es sein, daß der Client mehrere Prozeduren einer Serveranwendungskomponente aufrufen muß, um verwendbare Arbeit zu leisten, wodurch sich die Netzübertragungskosten wiederum erhöhen. Insbesondere moderne objektorientierte Designtechniken neigen zu einer Vielzahl von Aufrufen von Prozeduren mit relativ wenigen Parametern pro Aufruf. Ein typisches derartiges Objekt ist beispielsweise so gestaltet, daß ein Client zunächst zur Vorbereitung einer Operation verschiedene Eigenschaftsprozeduren ("Property-Set"-Prozeduren) und anschließend eine Prozedur zum Verarbeiten der Operation aufruft. Folglich kann viel Zeit durch Netzwerk-Overhead verwendet werden. Aufgrund dieser Netzübertragungskosten können Echtzeit-Prozedur-Aufruf-Modelle für einige Anwendungen weniger geeignet sein.

[0015] In Bezug auf Priorität werden Aufrufe bei typischen Echtzeit-Prozedur-Aufruf-Modellen nach dem "First-come, first-served"-Prinzip dann bearbeitet, wenn sie eintreffen, ohne jegliche Berücksichtigung von Priorität.

[0016] In Bezug auf die Lebensdauer des Objekts besitzen Serveranwendungsobjekte in Echtzeit-Prozedur-Aufruf-Modellen eine tendenziell lange Lebensdauer. Bei der typischen Client/Server-Datenverarbeitung durch ein verteiltes Objekt existiert das Serveranwendungsobjekt auf dem Server-Computer ab dem Zeitpunkt der Erzeugung durch den Client solange, bis das Objekt freigegeben wird. Vor einem nächsten Prozedur-Aufruf durch den Client verwendet das Objekt einen Großteil dieser Zeit einfach auf das Warten auf den Aufruf der Prozeduren des Objekts durch den Client und auf die Netzübertragungskosten für das Zurückliefern von Prozedurergebnissen. Währenddessen verbraucht das Objekt Ressourcen des Server-Computers, einschließlich Speicher- und Verarbeitungs-Overhead der Ausführungsumgebung. Das Serveranwendungsobjekt "erwacht" und „schläft" gewissermaßen mindestens ein Mal pro Prozedur-Aufruf. Diese "Belegungszeit" des Serveranwendungsobjekts stellt ein Hemmnis für schnelles Durchlaufen von Serverobjekten dar und führt zur Beschränkung von Serveranwendungsskalierbarkeit.

[0017] In Bezug auf Referenzlokalität greifen viele Teile von Computersystemen (z.B. der Prozessor-Cache und der Arbeitsbereich des virtuellen Speichers) auf Referenzlokalität zurück, um Leistungszuwächse zu erreichen. Stark lokale Anwendungen, z.B. Objekte oder Anwendungscode, die/der direkt unter Nutzung lokal verfügbarer Daten ausführen/ausführt, verfügen hierfür über Referenzmuster, die besser Leistungen erzielen. Im Gegensatz dazu besitzen Anwendungen mit über mehrere Server-Computer verteilten Objekten, die Echtzeit-Prozedur-Aufrufe wie das RPC nutzen, nur geringe Referenzlokalität. Beispielsweise wird ein Serverobjekt durch den ersten Echtzeit-Prozedur-Aufruf erzeugt und verbringt häufig einen Großteil seiner Lebensdauer damit, auf nachfolgende Echtzeit-Aufrufe von Prozeduren zu warten. Nach der Verarbeitung jedes Prozedur-Aufrufs wird das Serverobjekt in der Regel aus dem Prozessor-Cache entfernt und fällt manchmal zwischen Aufrufen aus dem Arbeitsbereich des virtuellen Speichers heraus. Auf diese Weise verringern die Echtzeit-Aufrufe von Prozeduren den Transaktions-Verarbeitungsumfang der Anwendung.

[0018] Eine Alternative zu „verteilten Objekten", die das Problem der eingeschränkten Verfügbarkeit eines RPC-Echtzeit-Prozedur-Aufrufs zum Teil bewältigt, ist eine mitunter als "nachrichtenorientierte Middleware" (Message Oriented Middleware) (MOM)) bezeichnete Art der Client/Server-Datenverarbeitung. Bei MOM kommuniziert eine Client-Anwendung mit einer entfernten Server-Anwendung durch Senden von Nachrichten an eine Nachrichtenwarteschlange. Die Server-Anwendung, die zu einem späteren Zeitpunkt als die Client-Anwendung laufen kann, ruft die Nachrichten von ihrer Nachrichtenwarteschlange ab und verarbeitet diese. Die Serveranwendung kann Ergebnisse dieser Verarbeitung an die Client-Anwendung zurücksenden, indem sie Nachrichten an eine gleiche oder separate Nachrichtenwarteschlange zur Verarbeitung durch die Client-Anwendung sendet. Solcher Datentransfer unter Verwendung von Nachrichtenwarteschlangen hat den Vorteil, daß die Client- und die Server-Anwendungen nicht gleichzeitig verfügbar sein und keine übereinstimmenden Lebensdauern haben müssen.

[0019] Herkömmliche MOM-Produkte weisen allerdings auch Anzahl von Einschränkungen auf. Eine Einschränkung besteht darin, daß die Client- und die Serveranwendung die Nachricht selbst als einen linearen Stream formatieren. Genauer gesagt, müssen die Client- und die Serveranwendungen die Daten dieses Streams selbst arrangieren und das Arrangieren selbst rückgängig machen. Die Nachrichtenwarteschlangen-Infrastruktur umfaßt keinen Support für das Arrangieren.

[0020] Eine weitere Einschränkung von MOM besteht darin, daß die Client- und die Server-Anwendung für die Kommunikation mit der Nachrichtenwarteschlangen-Infrastruktur eine herstellerspezifische Programmierschnittstelle (Application Programming Interface (API)) für Nachrichtenwarten-schlangenbildungsanwendungen nutzen. Anders formuliert, senden die Client- und die Serveranwendung Nachrichten an eine Nachrichtenwarteschlange über explizite Aufrufe an die Nachrichtenwarteschlangen bildende API. Für die Entwickler von Client/Serveranwendungen gilt es, noch ein weiteres API-Set zu lernen. So ist beispielsweise die Nachrichtenwarteschlangen bildende Schnittstelle (MQI) (für MQSeries API) die API des nachrichtenorientierten Middlewareprodukts der MQSeries von IBM. Andere Hersteller von MOM-Produkten, wie beispielsweise Microsoft (Microsoft Message Queue (MSMQ), Covia (Communications Integrator), Peerlogic (PIPES), Horizon Strategies (Message Express) oder System Strategies (ezBridge) haben unterschiedliche Nachrichtenwarteschlangen bildende APIs.

[0021] Außerdem erstellen Abnehmer eines MOM-Produkts (wie beispielsweise eine Informationstechnologie-Organisation (IT-Organisation)) aus unterschiedlichen Beweggründen eine separate API-Schicht über die

vom Hersteller bereitgestellte Nachrichtenwarteschlangen bildende API. Ein Beweggrund ist, daß die IT-Organisation verhindert möchte, daß ihre Anwendungsentwickler entscheiden können, welche Teile der Nachrichtenwarteschlangen bildenden API sie nutzen und auf welche Weise sie die API nutzen. Ein zweiter Beweggrund kann sein, daß die Nachrichtenwarteschlangen bildende API zu umfangreich, zu komplex ist oder den Anwendungsentwicklern zu viele Optionen bietet. Die IT-Organisationen erstellen somit ihre eigene API-Schicht in der Absicht, die API für ihre Anwendungsentwickler zu vereinfachen. Folglich müssen die meisten Entwickler sowohl die Nachrichtenwarteschlangen bildende API des Herstellers als auch die von der IT-Organisation bereitgestellte API lernen. Ein dritter Beweggrund kann sein, daß die IT-Organisation vermeiden möchte, daß ihre Anwendungen von einer jeglichen Nachrichtenwarteschlangen bildenden API eines Herstellers oder einem jeglichen MOM-Produkt anhängig sind, und sich die Möglichkeit offenhalten möchte, die Hersteller zu einem späteren Zeitpunkt zu wechseln. Dies hindert häufig die IT-Organisation daran, die Funktionen des MOM-Produkts voll auszuschöpfen. Dieses Phänomen verzögert die Implementierung von Anwendungen mit einem MOM-Produkt und erschwert der IT-Organisation bisweilen die Nutzung neueingeführter Funktionen des MOM-Produkts.

[0022] Eine weitere Einschränkung herkömmlicher MOM-Produkte besteht in der suboptimalen Leistungsfähigkeit in einigen Konfigurationen. Insbesondere können Anwendungen und Anwendungsobjekte lokal über Aufrufe von Prozeduren zu extrem niedrigen Verarbeitungszeitkosten kommunizieren, insbesondere wenn sie in einem gleichen Prozeß sind. Anwendungen und Anwendungsobjekte, die über eine Nachrichtenwarteschlangen bildende API kommunizieren, erfordern die Verarbeitung über einen Warteschlangenmanager, selbst wenn sich die Anwendungen oder Objekte in derselben oder einer ähnlichen Umgebung, wie beispielsweise einem gleichen Prozeß, befinden.

[0023] In ihrem Aufsatz "Design of a Remote Procedure Call System for Object-Oriented Distributed Programming" beschreiben die Autoren Anand R. Tripathi und Terence Noonan das Design eines RPC-Systems zur Bildung objektorientierter verteilter Softwaresysteme. Bei Nutzung dieses Systems kann eine eine Prozedur darstellende Anfragenachricht an einen Server übermittelt werden, auf dem ein Objekt unter Verwendung einer Kernel-Funktion verwaltet wird. Der Kernel sendet den einzigen Identifikator für den Aufruf an die RPC-Laufzeit zurück. Um die Antwort des Aufrufs zu empfangen, führt der Client-Prozeß RPC-Laufzeit-Funktion aus und überprüft anschließend den Status des Aufrufs in einer Aufruftabelle. Wurde der Aufruf aufgrund eines Zeitüberschreitungs- oder eines jeglichen anderen Fehlerzustands abgebrochen, sendet er diesen Status an den Client-Prozeß zurück und entfernt den Eintrag für diesen Auftrag aus der Aufruftabelle. Ist der Aufruf noch zu erledigen, führt er eine Funktion zum Empfangen einer Antwortnachricht vom Kernel aus. Er wartet während der in einer Sync-Funktion vorgegebenen Zeitüberschreitungsperiode auf den Eingang der Antwortnachricht, und wenn während dieser Zeitüberschreitungsperiode keine Nachricht eingeht und der Aufruf im Kernel immer noch anhängig ist, wird ein Zeitüberschreitungsstatus an den Client-Prozeß zurückgesandt. Der Client-Prozeß muß dann die Synchronisierung zu einem späteren Zeitpunkt nochmals versuchen.

Zusammenfassung der Erfindung

[0024] Die vorliegende Erfindung befähigt einen Client eines Objekts zur Ausgabe und das Objekt zum Empfang von Aufrufen von Prozeduren auf der Grundlage einer Warteschlange unter Nutzung normaler Aufrufsemantik eines Objektmodells ohne Verwendung einer Nachrichtenwarteschlangen bildenden API. Genauer gesagt nutzt der Client die normale Semantik des Objektmodells, um das Objekts zu erstellen, die Prozeduren des Objekts aufzurufen und das Objekt freizugeben. Der Rahmen oder die Ausführungsumgebung des Objekts stellt Dienste zum automatischen Anordnen der Aufrufe von Prozeduren in einer Warteschlange und zum möglicherweise späteren Ausgeben der in einer Warteschlange angeordneten Aufrufe von Prozeduren an das Objekt bereit. Unterdessen kann der Client asynchron zu der aufgerufenen Prozedur die Ausführung fortsetzen. Das Objekt wiederum hat die an seine Schnittstelle(n) über normale Aufrufsemantik ausgegebenen, in einer Warteschlange angeordneten Aufrufe von Prozeduren, verarbeitet diese und sendet anschließend einen Wert der Prozedur unter erneuter Nutzung normaler Aufrufsemantik zurück. Auf diese Weise müssen die Anwendungsentwickler den Client und das Objekt nicht für die Nutzung einer Nachrichtenwarteschlangen bildenden API zur Warteschlangenverarbeitung programmieren, wie es bei herkömmlicher nachrichtenorientierter Middleware der Fall ist, und das Erlernen solcher APIs ist nicht erforderlich.

[0025] Gemäß einem Aspekt der Erfindung kann ein Objekt sowohl Echtzeit- als auch in einer Wartschlange angeordnete Aufrufe von Prozeduren über eine gemeinsame Schnittstelle empfangen. Echtzeit-Aufrufe von Prozeduren werden an das Objekt über einen lokalen Prozedur-Aufruf oder einen entfernten Prozedur-Aufruf an die Schnittstelle ausgegeben. In einer Warteschlange angeordnete Aufrufe von Prozeduren werden in einer Vorrichtung für in einer Warteschlange angeordnete Aufrufe von Prozeduren in einer Warteschlange angeord-

net und anschließend über einen lokalen Prozedur-Aufruf von der Vorrichtung an die Objektschnittstelle ausgegeben. Das Objekt unterscheidet nicht zwischen den Echtzeit- und den in einer Warteschlange angeordneten Aufrufen von Prozeduren. Daher kann das Objekt ohne Veränderung sowohl in einer synchronen Echtzeit-Umgebung als auch in einer asynchronen Warteschlangen-Umgebung laufen gelassen werden. In gewisser Weise weiß das Objekt weder, ob es sich in einer Echtzeit- oder in einer Warteschlangen-Umgebung befindet, noch, in welcher Weise die Prozeduren auf seiner Schnittstelle aufgerufen werden.

[0026] Gemäß einem weiteren Aspekt der Erfindung stellt der Rahmen oder die Umgebung des Objekts außerdem Support zum automatischen Arrangieren für in einer Warteschlange angeordnete Aufrufe von Prozeduren bereit. Ein Client gibt in einer Warteschlange angeordnete Aufrufe von Prozeduren an ein Objekt über einen vom System bereitgestellten Proxy und Stub oder Adapter (Wrapper) aus, der aus der Beschreibung einer Schnittstellendefinitionssprache der Schnittstelle des Objekts übersetzt ist, um das geeignete Arrangieren von Aufrufparametern und zugehörigen Daten zu und von in einer Warteschlange angeordneten Nachrichten zu implementieren.

[0027] Gemäß einem weiteren Aspekt der Erfindung umfaßt die Vorrichtung zum Anordnen von Aufrufen von Prozeduren in einer Warteschlange eine Aufzeichnungseinrichtung für Aufrufe von Prozeduren auf einer Client-Seite und eine Abspieleinrichtung für Aufrufe von Prozeduren auf der Objekt-Seite der Client-Objekt-Interaktion. Die Aufzeichnungseinrichtung für Prozedur-Aufrufe empfängt eine Anzahl von möglicherweise mehr als einem Prozedur-Aufruf eines Clients, die als ein Stapel an die Abspieleinrichtung für Prozedur-Aufrufe weitergeleitet werden soll. Beispielsweise kann die Aufzeichnungseinrichtung eine Menge von in einer Warteschlange angeordneten, vom Client an das Objekt als Teil einer Transaktion ausgegebenen Prozedur-Aufrufen sammeln und die Prozedur-Aufrufe erst nach Abschluß der Transaktion weitergeben. Die Abspieleinrichtung für Prozedur-Aufrufe ruft die in einer Warteschlange angeordneten Prozedur-Aufrufe nacheinander von einer Warteschlange für Prozedur-Aufrufe ab und gibt die Aufrufe von Prozeduren – möglicherweise als Teil einer anderen, das Objekt umfassenden Transaktion – an das Objekt aus.

[0028] Gemäß einem weiteren Aspekt der Erfindung entspricht ein Objekt, das in einer Warteschlange angeordnete Aufrufe von Prozeduren unterstützt, der Einschränkung, daß dessen Prozeduren ausschließlich Eingabeparameter aufweisen können und keine anwendungsspezifischen Informationen zurücksenden können. Die in einer Warteschlange angeordneten Aufrufe von Prozeduren können dann als unidirektionale Kommunikationen vom Client zum Objekt ausgegeben werden, bei denen der Client nicht in Echtzeit- oder synchroner Ausführung auf Ergebnisse des Prozedur-Aufrufs warten muß. Das Objekt kann Ergebnisse der Verarbeitung der in einer Warteschlange angeordneten Prozedur-Aufrufe durch Ausgabe von Echtzeit-Prozedur-Aufrufen oder von in einer Warteschlange angeordneten Prozedur-Aufrufen an ein durch einen Eingabeparameter vom Client beschriebenes Ergebnisobjekt liefern.

[0029] Weitere Eigenschaften und Vorteile der Erfindung werden anhand der nachfolgenden detaillierten Beschreibung einer Ausführungsform unter Bezugnahme auf die beigefügten Zeichnungen ersichtlich.

Kurze Beschreibung der Zeichnungen

[0030] Fig. 1 ist ein Blockdiagramm eines Computersystems, das zur Implementierung eines Verfahrens und einer Vorrichtung genutzt werden kann, welche die Erfindung für in einer Warteschlange angeordnete Aufrufe von Prozeduren verkörpern.

[0031] Fig. 2 ist ein Blockdiagramm einer Ausführungsumgebung und einer Laufzeitarchitektur für in einer Warteschlange angeordnete Aufrufe von Prozeduren gemäß der dargestellten Ausführungsform der Erfindung.

[0032] Fig. 3 ist ein Blockdiagramm einer Struktur einer in einer Warteschlange angeordneten Komponente in der Ausführungsumgebung von Fig. 2.

[0033] Fig. 4 ist ein Blockdiagramm einer Aufzeichnungseinrichtung und eines Proxy in der Laufzeitarchitektur von Fig. 2.

[0034] Fig. 5 ist ein Blockdiagramm einer Abspieleinrichtung und eines Stub in der Laufzeitarchitektur von Fig. 2.

[0035] Fig. 6 ist ein Blockdiagramm einer Vorrichtung zur Überwachung von Warteschlangen für unzustellba-

re Nachrichten (Dead-Letter-Queue) und eines Ausnahmezustandsbehandlers in der Laufzeitarchitektur von Fig. 2.

[0036] Fig. 7 ist eine Programmliste von beispielhaften Objekt-Instanziierungs-Aufrufen zur Aktivierung der in einer Warteschlange angeordneten Komponente in der Laufzeitarchitektur von Fig. 2.

[0037] Fig. 8 ist eine Programmliste einer „IPlaybackControl"-Schnittstelle eines Ausnahmezustandsbehandlers in Fig. 5 und Fig. 6.

[0038] Fig. 9 ist eine Programmliste eines Formats einer Nachricht von in einer Warteschlange angeordneten Aufrufen von Prozeduren in der Laufzeitarchitektur von Fig. 2.

[0039] Fig. 10 ist ein Blockdiagramm des Formats einer in einer Warteschlange angeordnete Prozedur-Aufrufe enthaltenden Nachricht in der Laufzeitarchitektur von Fig. 2.

DETAILLIERTE BESCHREIBUNG DER ERFINDUNG

[0040] Die vorliegende Erfindung betrifft ein Verfahren und ein System für in einer Warteschlange angeordnete Aufrufe von Komponentenprozeduren. In einer hier erläuterten Ausführungsform wird die Erfindung in eine als „COM+" bezeichnete Objektdienste-Komponente eines als „Microsoft Windows NT Server 5.0" bezeichneten, durch die Microsoft Corporation in Redmond, Washington vertriebenen Betriebssystems integriert. Kurz beschrieben ist diese Software ein skalierbares Hochleistungsnetzwerk- und Computerbetriebssystem, das verteilte Client/Server-Datenverarbeitung unterstützt und eine Objekt-Ausführungsumgebung für dem Komponentenobjektmodell (COM) von Microsoft entsprechende Komponentenanwendungen bereitstellt. Die COM+-Komponente umfaßt Objektdienste von vorbekannten Objektsystemen, einschließlich des Microsoft-Komponentenobjektmodells (COM), der Microsoft-Objektverlinkung und -einbettung (Microsoft Object Linking and Embedding (OLE)), des Verteilten Komponentenmodells (DCOM) von Microsoft und des Microsoft-Transaktionsservers (Microsoft Transaction Server (MTS)).

Beispielhafte Betriebsumgebung

[0041] Fig. 1 und die folgende Darstellung dienen einer kurzen, allgemeinen Beschreibung einer geeigneten Datenverarbeitungsumgebung, in welcher die Erfindung implementiert werden kann. Obwohl die Erfindung im allgemeinen Kontext computerausführbarer Anweisungen eines auf einem Computer laufenden Computerprogramms beschrieben wird, wird der Fachmann feststellen, daß die Erfindung gleichfalls in Kombination mit anderen Programmmodulen implementiert werden kann. Programmmodule umfassen im Allgemeinen Routinen, Programme, Komponenten, Datenstrukturen usw., die bestimmte Aufgaben ausführen oder bestimmte abstrakte Datentypen implementieren. Außerdem wird der Fachmann erkennen, daß die Erfindung mit anderen Computersystemkonfigurationen, wie beispielsweise Handheld-Vorrichtungen, Multiprozessorsystemen, Mikroprozessor-basierter oder programmierbarer Unterhaltungselektronik, Minicomputer, Großrechenanlagen u. a., betrieben werden kann. Die dargestellte Ausführungsform der Erfindung wird ebenfalls in verteilten Datenverarbeitungsumgebungen betrieben, wo Aufgaben über entfernte, durch ein Kommunikationsnetzwerk verbundene Verarbeitungsvorrichtungen ausgeführt werden. Einige Ausführungsformen der Erfindung können allerdings auch auf Einzelrechnern betrieben werden. In einer verteilten Datenverarbeitungsumgebung können sich Programmmodule sowohl in lokalen als auch in entfernten Speichervorrichtungen befinden.

[0042] Bezugnehmend auf Fig. 1 umfaßt ein beispielhaftes System zur Implementierung der Erfindung einen konventionellen Computer **20** (wie beispielsweise Personalcomputer, Laptops, Palmtops, Set-Tops, Server, Großrechenanlagen und andere Computerarten) mit einer Verarbeitungseinheit **21**, einem Systemspeicher **22** und einem Systembus **23**, der verschiedene Systemkomponenten einschließlich des Systemspeichers mit der Verarbeitungseinheit **21** verbindet. Die Verarbeitungseinheit kann ein jeglicher von verschiedenen im Handel erhältlichen Prozessoren sein, wie beispielsweise Intel x86-, Pentium- und kompatible Mikroprozessoren von Intel und anderen Herstellern, wie beispielsweise Cyrix, AMD und Nexgen; Alpha von Digital, MIPS von MIPS Technology, NEC, IDT, Siemens u. a. sowie der PowerPC von IBM und Motorola. Duale Mikroprozessoren und andere Multiprozessor-Architekturen können ebenfalls als Verarbeitungseinheit **21** genutzt werden.

[0043] Der Systembus kann ein jeglicher von verschiedenen, einen Speicherbus oder eine Speicher-Steuereinheit, einen Peripheriebus und einen lokalen Bus umfassenden Busstruktur-Typen sein, die eine von unterschiedlichen konventionellen Bus-Architekturen nutzen, wie beispielsweise PCI, VESA, AGP, Microchannel, ISA und EISA, um nur einige zu nennen. Der Systemspeicher umfaßt Nur-Lese-Speicher (ROM) **24** und Spei-

cher mit wahlfreiem Zugriff (RAM) **25**. In ROM **24** befindet sich ein einfaches Eingabe/Ausgabe-System (BIOS), welches die Basisprogramme zur Übertragung von Informationen zwischen Elementen innerhalb des Computers **20**, wie beispielsweise während des Hochfahrens, umfaßt.

[0044] Der Computer **20** umfaßt weiterhin ein Festplattenlaufwerk **27**, ein Magnetplattenlaufwerk **28** beispielsweise zum Lesen einer oder zum Schreiben auf eine Wechselplatte **29** und ein Optische-Medien-Laufwerk **30** beispielsweise zum Lesen einer CD-ROM-Platte **31** oder zum Lesen eines oder zum Schreiben auf ein anderes optisches Medium. Das Festplattenlaufwerk **27**, das Magnetplattenlaufwerk **28** und das Optische-Medien-Laufwerk **30** sind jeweils über eine Festplattenlaufwerk-Schnittstelle **32**, eine Magnetplattenlaufwerk-Schnittstelle **33** bzw. eine Schnittstelle für ein Optische-Medien-Laufwerk **24** mit dem Systembus **23** verbunden. Die Laufwerke und ihre dazugehörigen computerlesbaren Medien stellen nichtflüchtigen Speicher von Daten, Datenstrukturen, Computer-ausführbaren Anweisungen usw. für den Computer **20** bereit. Obwohl sich die obige Beschreibung computerlesbarer Medien auf eine Festplatte, eine magnetische Wechselplatte und eine CD bezieht, sollte der Fachmann feststellen, daß auch andere computerlesbaren Medientypen, wie beispielsweise Magnetbandkassetten, Flash-Speicherkarten, digitale Videodisks, Bernoulli-Laufwerke u. a., in der beispielhaften Betriebsumgebung eingesetzt werden können.

[0045] In den Laufwerken und dem RAM **25** kann eine Anzahl von Programmmodulen, einschließlich des Betriebssystems **35**, eines oder mehrerer Anwendungsprogramme **36**, weiterer Programmmodule **37** und Programmdaten **38**, gespeichert werden.

[0046] Ein Benutzer kann über eine Tastatur **40**, und eine Steuervorrichtung, wie beispielsweise eine Maus **42**, Befehle und Informationen in den Computer **20** eingeben. Weitere (nicht abgebildete) Eingabevorrichtungen können ein Mikrophon, ein Joystick, ein Gamepad, eine Parabolantenne, ein Scanner oder ähnliches sein. Diese und andere Eingabevorrichtungen sind an die Verarbeitungseinheit **21** häufig über einen seriellen Schnittstellenport **46** angeschlossen, der mit dem Systembus verbunden ist, aber auch an andere Schnittstellen, wie beispielsweise einen Parallel-Port, einen Game-Port oder einen universellen seriellen Bus (USB) angeschlossen sein kann. Außerdem ist ein Bildschirm **47** oder eine andere Anzeigevorrichtung über eine Schnittstelle, wie beispielsweise einen Video-Adapter **48**, an den Systembus **23** angeschlossen. Zusätzlich zum Bildschirm umfassen Computer in der Regel andere periphere (nicht abgebildete) Ausgabevorrichtungen, wie beispielsweise Lautsprecher und Drucker.

[0047] Der Computer **20** kann in einer vernetzten, logische Verbindungen zu einem oder mehreren entfernten Computer, wie beispielsweise einem entfernten Computer **49**, nutzenden Umgebung operieren. Der entfernte Computer **49** kann ein Server, ein Router, eine Peer-Vorrichtung oder ein anderer gemeinsamer Netzknoten sein und umfaßt in der Regel viele oder alle der in Bezug auf den Computer **20** beschriebenen Elemente, auch wenn in Fig. 1 nur eine Speichervorrichtung **50** dargestellt ist. Die in Fig. 1 dargestellten logischen Verbindungen umfassen ein lokales Netz (LAN) **51** und ein Weitbereichsnetz (WAN) **52**. Entsprechende vernetzende Umgebungen sind gewöhnlich in Büros, Computer-Netzwerken in Unternehmen, internen Netzwerken und im Internet anzutreffen.

[0048] Bei Einsatz in einer LAN-vernetzenden Umgebung wird der Computer **20** über eine Netzwerkschnittstelle oder einen -adapter **53** mit dem lokalen Netz **51** verbunden. Bei Einsatz in einer WAN-vernetzenden Umgebung umfaßt der Computer **20** in der Regel ein Modem **54** oder andere Mittel zum Aufbau von Kommunikation (z. B. mittels LAN, eines Gateway- oder eines Proxy-Servers **55**) über das Weitbereichsnetz **52**, wie beispielsweise das Internet. Das Modem **54**, das ein internes oder externes sein kann, wird über den seriellen Schnittstellenport **46** an den Systembus **23** angeschlossen. In einer vernetzten Umgebung können in Bezug auf den Computer **20** dargestellte Programmodule oder Teile davon in einer entfernten Speichervorrichtung gespeichert werden. Man erkennt, daß die dargestellten Netzwerkverbindungen Beispiele sind und andere Mittel zum Aufbau einer Kommunikationsverbindung zwischen Computern verwendet werden können.

[0049] In Übereinstimmung mit den Gepflogenheiten von Fachleuten aus dem Bereich der Computerprogrammierung erfolgt die nachstehende Beschreibung der vorliegenden Erfindung, falls nicht anders angegeben, unter Bezugnahme auf durch den Computer **20** ausgeführte Handlungen und symbolische Darstellungen von Operationen. Entsprechende Handlungen und Operationen werden gelegentlich als „durch einen Computer ausgeführt" bezeichnet. Man erkennt, daß die Handlungen und symbolisch dargestellten Operationen die durch die Verarbeitungseinheit **21** erfolgende, Datentbits darstellende Bearbeitung von elektrischen Signalen, welche eine resultierende Transformation oder Reduktion der Darstellung elektrischer Signale hervorruft, sowie die Verwaltung der Datenbits an im Speichersystem (mit Systemspeicher **22**, Festplattenlaufwerk **27**, Magnetplatten-Speicher **29** und CD-ROM **31**) befindlichen Speicherorten umfassen, um so die Operation des

Computersystems sowie andere Signalverarbeitung zu rekonfigurieren oder anderweitig zu ändern. Bei den Speicherorten, an denen Datenbits verwaltet werden, handelt es sich um physische Speicherplätze, die bestimmte elektrische, magnetische oder optische, den Datenbits entsprechende Eigenschaften aufweisen.

Ausführungsumgebung der Komponentenanwendung

[0050] Wie aus Fig. 2 ersichtlich, liefert die oben erwähnte COM+-Komponente des Microsoft Windows NT 5.0 Betriebssystems Laufzeit- oder Systemdienste zur Erstellung einer Ausführungsumgebung für Laufzeitobjekte **80** auf einem Server-Computer **84**, welche in einer Warteschlange angeordnete Aufrufe von Prozeduren an ein Objekt **86** (nachstehend als „Warteschlangenkomponente" bezeichnet) automatisch bereitstellt. Die COM+-Komponente ist als dynamische Linkbibliothek (Dynamic Link Library (DLL)) implementiert. (Eine DLL ist ein bekanntes ausführbares Dateiformat, welches dynamische Verlinkung oder Laufzeitverlinkung von ausführbarem Code in den Prozeß eines Anwendungsprogramms ermöglicht). Die COM+-DLL wird direkt in Anwendungsserverprozesse (z. B. „ASP" **90**) geladen, die Komponentenanwendungsobjekte hosten, und läuft transparent im Hintergrund dieser Prozesse.

[0051] Der dargestellte ASP **90** ist ein Systemprozeß, der die Ausführung von Komponentenanwendungsobjekten einschließlich der Warteschlangenkomponente **86** hostet. Jeder ASP **90** kann mehrere Komponentenanwendungsobjekte hosten, die zu einer als „COM+-Anwendung" bezeichneten Sammlung (in der Objektausführungsumgebung des vorbekannten Microsoft-Transaktionsservers auch als „Package" bezeichnet) gruppiert werden. Ferner können mehrere ASP **90** auf dem Server-Computer **84** unter einem Multithreaded-, Multitasking-Betriebssystem (z. B. Microsoft Windows NT in der dargestellten Ausführungsform) ablaufen. Jeder ASP **90** stellt eine separate Vertrauensgrenze (separate trust boundary) und einen separaten Fehlereingrenzungsbereich (fault isolation domain) für die Serveranwendungsobjekte bereit. Anders formuliert, kann sich bei getrennten ASPs ein durch ein Serveranwendungsobjekt hervorgerufener Fehler, der eine Beendigung seines ASP bewirkt, in der Regel nicht auf die Serveranwendungsobjekte in einem anderen ASP auswirken. In der dargestellten Ausführungsform sind die Komponentenanwendungsobjekte als eine COM+-Anwendung gruppiert, um mit Hilfe eines als „COM+-Explorer" bezeichneten Verwaltungsprogramms gemeinsam in einem ASP zu laufen. Dieses Programm stellt eine grafische Benutzerschnittstelle zur Steuerung von mit den Komponentenanwendungsobjekten verbundenen Attributen einschließlich der Gruppierung von Objekten in COM+-Anwendungen bereit.

[0052] In einer typischen, in Fig. 2 dargestellten Installation befindet sich die Ausführungsumgebung **80** auf dem Server-Computer **84** (der ein Beispiel des oben beschriebenen Computers **20** sein kann), der in einem verteilten Computernetzwerk angeschlossen ist, welches eine große Menge an auf die Komponentenanwendungsobjekte in der Ausführungsumgebung **80** zugreifenden Client-Computern **92** umfaßt. Alternativ kann sich die Ausführungsumgebung **80** auf einem Einzelcomputer befinden und Komponentenanwendungsobjekte hosten, auf die ebenfalls auf diesem Computer befindliche Client-Prozesse zugreifen.

Überblick über Komponentenanwendungsobjekte

[0053] Unter Bezugnahme auf Fig. 2 führt der Computer **84** als eine COM+-Anwendung entwickelte, eine Gruppe von Komponentenanwendungsobjekte umfassende Komponentenanwendungen aus. Beispielsweise können die in der Ausführungsumgebung **80** des ASP **90** gehosteten Komponentenanwendungsobjekte (wie z. B. die Warteschlangenkomponente **86**) die Geschäftslogik einer Client/Server-Anwendung, wie z. B. den Code zur Steuerung von Einschreibungen in einer Registrierungsanwendung einer Universität oder von Aufträgen in einer Online-Verkaufsanwendung, implementieren. In der Regel umfaßt jede Komponentenanwendung eine Mehrzahl von Komponenten, wovon jede Programmcode für einen Teil der Anwendungsarbeit enthält.

[0054] Bezugnehmend auf Fig. 3 entsprechen die Komponentenanwendungsobjekte in der dargestellten Ausführungsumgebung **80** (Fig. 2) der Anforderung des Microsoft-Komponentenobjektmodells ("COM") (d.h., daß sie als ein "COM-Objekt" **100** implementiert sind) und werden, wie oben beschrieben, unter Nutzung der COM+-Dienste des Microsoft Windows NT Server 5.0 Betriebssystems ausgeführt; alternativ können sie allerdings auch nach anderen Objektstandards implementiert werden (wie beispielsweise den Anforderungen der CORBA (Common Objekt Request Broker Architecture (Architektur für Vermittler der Abrufe von gemeinsamen Objekten)) der Object Management Group oder Java Beans von der Sun Microsystems Inc.) und unter Objektdiensten eines anderen Betriebssystems ausgeführt werden. Die COM-Anforderung definiert binäre Standards für Objekte und deren Interfaces, welche die Integration von Softwarekomponenten in Anwendungen erleichtern. (Für eine detaillierte Erörterung von COM und OLE siehe Kraig Brockschmidt: "Inside OLE", 2. Auflage,

Microsoft Press, Redmond, Washington 1995).

[0055] Gemäß COM wird das COM-Objekt **100** im Computer **84** (Fig. 2) durch eine Instanzdatenstruktur **102**, eine Tabelle **104** der virtuellen Funktionen und Prozeduren oder Memberfunktionen **106**–**108** dargestellt. Die Instanzdatenstruktur **102** umfaßt einen Zeiger **110** auf die Tabelle **104** der virtuellen Funktionen und Daten **112** (auch als Datenmember oder Eigenschaften des Objekts bezeichnet). Ein Zeiger ist ein Datenwert, der die Adresse eines Gegenstands im Speicher hält. Die Tabelle **104** der virtuellen Funktionen umfaßt Einträge **116**–**118** für die Prozeduren **106**–**108**. Jeder dieser Einträge **116**–**118** umfaßt einen Verweis auf den die entsprechende Prozedur implementierenden Code **106**–**108**.

[0056] Der Zeiger **110**, die Tabelle **104** der virtuellen Funktionen und die Prozeduren **106**–**108** implementieren eine Schnittstelle des COM-Objekts **100**. Üblicherweise werden die Schnittstellen eines COM-Objekts grafisch als eine Steckbuchse dargestellt, wie für die Warteschlangenkomponente **86** in Fig. 2 gezeigt. Außerdem erhalten Schnittstellen üblicherweise Bezeichnungen, die mit dem Großbuchstaben "I" beginnen. Gemäß COM kann das COM-Objekt **100** mehrere Schnittstellen umfassen, die mit einer oder mehreren Tabellen der virtuellen Funktionen implementiert werden. Die Prozedur einer Schnittstelle wird mit „IInterfaceName::FunctionName" bezeichnet.

[0057] Die Tabelle **104** der virtuellen Funktionen und die Prozeduren **106**–**108** des COM-Objekts **100** werden von einem Objektserverprogramm **120** (nachstehend als „Objektserver-DLL" bezeichnet) bereitgestellt, welches in dem Computer **20** (Fig. 1) als eine Datei einer dynamischen Linkbibliothek (mit der Dateinamenerweiterung ".dll" versehen) gespeichert ist. Gemäß COM umfaßt die Objektserver-DLL **120** Code für die Tabelle **104** der virtuellen Funktionen und die Prozeduren **106**–**108** der Klassen, die sie unterstützt, sowie weiterhin eine Klassenfabrik **122**, welche die Instanzdatenstruktur **102** für ein Objekt der Klasse generiert.

[0058] Andere Objekte und Programme (als „Client" des COM-Objekts **100** bezeichnet) greifen auf die Funktionalität des COM-Objekts durch Aufrufen der Prozeduren über die Schnittstellen des COM-Objekts zu. Zunächst muß das COM-Objekt jedoch instanziiert werden (z. B. indem die Klassenfabrik zur Erstellung der Instanzdatenstruktur **102** des Objekts veranlaßt wird), und der Client muß einen Schnittstellenzeiger auf das COM-Objekt erhalten.

[0059] Bevor das COM-Objekt **100** instanziiert werden kann, wird das Objekt zunächst auf dem Computer **20** installiert. In der Regel umfaßt die Installation das Installieren einer Gruppe verwandter, in einer COM+-Anwendung enthaltener Objekte. Die Installation des COM-Objekts **100** erfolgt durch das Speichern der Objektserver-DLL-Datei(en), welche das Objekt in dem Computer **20** zugänglichem Datenspeicher (in der Regel die in Fig. 1 dargestellte Festplatte) bereitstellt/bereitstellen, und das Registrieren von COM-Attributen (z. B. Klassenidentifikatoren, Pfad und Bezeichnung der Objektserver-DLL-Datei **120** usw.) des COM-Objekts in einem Systemregister, einem Katalog oder einer ähnlichen Konfigurationsdatenbank.

[0060] Ein Client fordert die Instanziierung des COM-Objekts unter Einsatz vom System bereitgestellter Dienste und einer Menge systemdefinierter Standard-Komponentenschnittstellen, die auf den Klassen und Schnittstellen des COM-Objekts zugewiesenen Klassen- und Schnittstellenidentifikatoren basieren, an. Genauer gesagt sind die Dienste für die Client-Programme als Funktionen von Schnittstellen zur Anwendungsprogrammierung (API) verfügbar, die in der COM+-Bibliothek bereitgestellt werden, welche eine Komponente des Microsoft Windows NT Server 5.0 Betriebssystems in einer Datei mit dem Namen „OLE32.DLL." ist. Außerdem werden in COM+ Klassen von COM-Objekten ausschließlich mit Klassenidentifikatoren („CLSIDs") assoziiert und von ihrem CLSID in einer als „Registry" bezeichneten Systemkonfigurationsdatenbank registriert.

[0061] Der Registrierungseintrag für eine Klasse von COM-Objekten assoziiert den CLSID der Klasse mit Informationen, die eine ausführbare, die Klasse bereitstellende Datei (z.B. eine DLL-Datei, die eine Klassenfabrik zur Erstellung einer Instanz der Klasse aufweist) identifizieren. Klassenidentifikatoren sind 128-Bit global eindeutige Identifikatoren (Globally Unique Identifiers ("GUIDs")), die der Programmierer mittels eines als "CoCreateGUID" bezeichneten COM+-Dienstes (oder einer/einem von verschiedenen anderen APIs und Programmen zur Erstellung universeller eindeutiger Identifikatoren) programmiert und den entsprechenden Klassen zuweist. Zusätzlich werden die Schnittstellen einer Komponente mit Schnittstellenidentifikatoren ("IIDs") verbunden.

[0062] Insbesondere stellt die COM+-Bibliothek API-Funktionen, z.B. "CoCreateInstance()" und "CoGetObject()" bereit, die das Client-Programm aufrufen kann, um die Erstellung einer seine zugewiesene CLSID und einen IID einer gewünschten Schnittstelle nutzenden Komponente anzufordern. Als Reaktion auf eine Instan-

ziierungsanforderung des Clients sucht die "CoCreateInstance()"-API den Registrierungseintrag der angeforderten CLSID im Register, um die ausführbare Datei für die Klasse zu identifizieren. Die "CoCreateInstance()"-API-Funktion lädt anschließend die ausführbare Datei der Klasse und nutzt die Klassenfabrik in der ausführbaren Datei, um eine Instanz des COM-Objekts **100** zu erstellen. Abschließend sendet die "CoCreateInstance()"-API-Funktion einen Zeiger der angeforderten Schnittstelle an das Client-Programm zurück. Die "CoCreateInstance()"-API-Funktion kann die ausführbare Datei entweder in den Prozeß des Client-Programms oder in einen Serverprozeß laden, der, je nach den für das COM-Objekt **100** in der System-Registrierdatenbank registrierten Attributen, lokal oder entfernt sein kann (z.B. auf dem selben Computer oder auf einem entfernten Computer in einem verteilten Computernetzwerk). Die "CoGetObject()"-API andererseits nutzt die COM-Moniker-Architektur, um eine die Serverobjektklasse identifizierende Zeichenkette zu analysieren und ein Moniker-Objekt zu erstellen, welches anschließend zur Erstellung einer Instanz der Serverobjektklasse genutzt wird.

[0063] Sobald der Client des COM-Objekts **100** diesen ersten Schnittstellen-Zeiger des COM-Objekts erhalten hat, kann der Client Zeiger anderer gewünschter Schnittstellen der Komponente durch Nutzung des mit der gewünschten Schnittstelle verbundenen Schnittstellenidentifikators erhalten. COM+ definiert verschiedene, üblicherweise von COM-Objekten unterstützte Standardschnittstellen einschließlich der "IUnknown"-Schnittstelle. Diese Schnittstelle umfaßt eine Prozedur namens "QueryInterface()". Die "QueryInterface()"-Funktion kann mit einem Schnittstellenidentifikator als ein Argument aufgerufen werden und sendet einen Zeiger an die mit diesem Schnittstellenidentifikator verbundene Schnittstelle zurück. Die "IUnknown"-Schnittstelle jedes COM-Objekts umfaßt außerdem Prozeduren, "AddRef()" und "Release()", zum Verwalten einer Zahl von Clientprogrammen, die einen Verweis (z.B. einen Schnittstellenzeiger) auf das COM-Objekt halten. Üblicherweise sind die Prozeduren der "IUnknown"-Schnittstelle als Teil jeder Schnittstelle auf einem COM-Objekt enthalten. Dementsprechend kann jeder Schnittstellenzeiger, den der Client auf eine Schnittstelle des COM-Objekts **100** erhält, zum Aufruf der "QueryInterface"-Funktion genutzt werden.

Überblick über Transaktionsverarbeitung

[0064] Nochmals bezugnehmend auf Fig. 2, implementiert die COM+-Komponente außerdem eine automatische Transaktionsverarbeitung für die Komponentenanwendungsobjekte in der dargestellten Ausführungsumgebung **80**. Eine ausführlichere Offenbarung der automatischen Transaktionsverarbeitung von komponentenbasierten Serveranwendungen findet sich in dem US-Patent Nr. 5,890,161 von Helland et al. über "Automatic Transaction Processing Of Component-Based Server Applications", eingereicht am 28.10.1997 und veröffentlicht am 30.03.1999 (nachfolgend als "Patentanmeldung für Automatische Transaktionen" bezeichnet). Kurz beschrieben, koordiniert die automatische Transaktionsverarbeitung die Verarbeitungsaktivitäten von Komponentenanwendungs-Objekten in der Ausführungsumgebung **80**, die Teile einer Operation bilden, um als eine einzelne, unteilbare Arbeitseinheit, die im Allgemeinen als Transaktion bezeichnet wird, wirksam zu werden.

[0065] Transaktionen in der Ausführungsumgebung **80** werden über einen Transaktionsmanager **128** gesteuert. Der Transaktionsmanager **128** ist ein Systemdienst, der mehrere gesteuerte Transaktionsressourcen, wie beispielsweise Datenbanken, Dateisysteme usw., umfassende Transaktionen koordiniert. Der Transaktionsmanager **128** gewährleistet, daß die gesamte in eine Transaktion eingebundene Verarbeitungsarbeit (z.B. Updates von Datenbanken) in Übereinstimmung mit den ACID-Eigenschaften (Atomarität (Atomicity), Konsistenz (Consistency), Isolation (Isolation), Dauerhaftigkeit (Durability)) unter Nutzung des vorbekannten Two-Phase-Commit-Protokolls und ungeachtet von Fehlern (z.B. Computer-, Netzwerk-, Hardware- oder Softwarefehler oder Fehler durch Fehlverhalten von Ressourcenmanager oder -anwendung), Wettlaufsituationen (z.B. eine Transaktion beginnt mit der Übergabe, während ein Ressourcenmanager einen Abbruch initiiert) oder Verfügbarkeit (ein Ressourcenmanager bereitet eine Transaktion vor, kehrt aber nicht zurück) erfolgt. Der dargestellte Transaktionsmanager **148** ist der als Teil des Microsoft SQL Servers 6.5 veröffentlichte Microsoft-Koordinator von verteilten Transaktionen (Microsoft Distributed Transaction Coordinator) (MSDTC)). Für zusätzliche Hintergrundinformationen über Transaktionsverarbeitung siehe u.a. Jim Gray und Andreas Reuter: "Transaction Processing Concepts and Techniques", Morgan Kaufmann, 1993.

Laufzeitarchitektur von in einer Warteschlange angeordneten Komponenten

[0066] Weiterhin bezugnehmend auf Fig. 2 stellt eine Laufzeitarchitektur **130** von in einer Warteschlange angeordneten Komponenten (nachfolgend als "QC-Architektur" bezeichnet) Support für in einer Warteschlange angeordnete Aufrufe von Prozeduren unter Nutzung normaler COM-Aufrufsemantik bereit. Genauer gesagt, ruft ein Client **132** in einem Prozeß **134** auf dem Client-Computer **92** Prozeduren auf Schnittstellen **87** der War-

teschlangenkomponente **86** unter Nutzung der üblichen COM-Konventionen für synchrone Echtzeitinteraktion auf, einschließlich folgender Schritte: Erstellen des Objekts, beispielsweise über einen "CoGetObject()"-Aufruf oder einen "CoCreateInstance()"-Aufruf; Empfang eines Schnittstellenzeigers, beispielsweise durch Spezifizieren eines Schnittstellenidentifikators (IID) in einem "CoCreateInstance()"-API-Aufruf oder einem "QueryInterface()"-Aufruf; die Ausgabe von Aufrufen an das Objekt über dessen Tabelle der virtuellen Funktionen oder eine Dispatch-Schnittstelle (für dynamische Bindung) und schließlich die Freigabe des Objekts beispielsweise durch Aufrufen der "Release"-Prozedur des Objekts. (In einigen Implementierungen der Erfindung, wie beispielsweise in der hier dargestellten QC-Architektur **130**, kann der Client einen anderen Objekterstellungsmechanismus für in einer Warteschlange angeordnete Prozedur-Aufrufe (z. B. die oben beschriebene "CoGetObject()"-API und einen Warteschlangen-Moniker (unten beschrieben)) als für Aufrufe von Echtzeit-Prozeduren nutzen, wobei er allerdings Prozeduren des Objekts nach wie vor mittels normaler Aufrufsemantik aufruft.) Die Warteschlangenkomponente **86** andererseits ruft ihre Prozeduren über ihre Tabelle der virtuellen Funktionen oder eine Dispatch-Schnittstelle wie bei einem lokalen Prozedur-Aufruf auf. Anders formuliert, müssen der Client **132** und die Warteschlangenkomponente **86** nicht so geschrieben werden, daß sie jegliche Nachrichten einreihende API nutzen, um in einer Warteschlange angeordnete Aufrufe von Prozeduren der Warteschlangenkomponente auszugeben oder zu empfangen.

[0067] Die Aufrufe von Prozeduren des Clients auf die Warteschlangenkomponente **86** werden auf einer Client-Seite **140** der Client-zu-Objekt-Interaktion aufgezeichnet und zu einem späteren Zeitpunkt wieder abgespielt und an die Warteschlangenkomponente **86** auf einer Server-Seite **142** der Interaktion ausgegeben. Die dargestellte QC-Architektur **130** zeichnet alle Prozedur-Aufrufe des Clients auf der Warteschlangenkomponente **86** auf und spielt die Prozedur-Aufrufe erst dann wieder ab, wenn der Client **132** seine Nutzung der Warteschlangenkomponente **86** abgeschlossen hat (d. h. nach Freigabe der Warteschlangenkomponente durch den Client). Falls der Client **132** an einer Transaktion beteiligt ist, gilt des Weiteren, daß die Prozedur-Aufrufe erst nach regulärem Abschluß der Transaktion wieder abgespielt werden.

[0068] Die dargestellte QC-Architektur **130** wird in der COM+-Komponente des Microsoft Windows NT 5.0 Betriebssystems implementiert. Die COM+-Komponente stellt verschiedene Laufzeitobjekt-Dienste für auf dem Computersystem **20** laufende COM-Objekte bereit. Die Laufzeitdienste stellen eine Aufzeichnungseinrichtung **150**, eine Empfangseinrichtung **152** und eine Abspieleinrichtung **154** bereit, die in einer Warteschlange angeordnete Aufrufe von Prozeduren über normale Aufrufsemantik durch den Client **132** auf die Warteschlangenkomponente **86** ausführen. Die Aufzeichnungseinrichtung **150** fungiert als ein Proxy für die Warteschlangenkomponente, um das Arrangieren der Prozedur-Aufrufe des Clients mit deren Aufrufparametern und zugehörigen Daten in Nachrichten auszuführen, und nutzt außerdem eine Nachrichtenwarteschlangen bildende API (wie beispielsweise die "Microsoft Message Queue" oder "MSMQ"), um die Nachrichten in eine der Warteschlangenkomponente **86** zugehörige Nachrichtenwarteschlange für Prozedur-Aufrufe **158** einzureihen. (Für weitere Details zu MSMQ siehe Microsoft Developer Network (MSDN) Library Edition – Juli 1998, SDK Documentation, Platform SDK, Networking and Distributed Services, Microsoft Message Queue Server (MSMQ).) Die Empfangseinrichtung **152** wartet darauf, daß Nachrichten an der Warteschlange **158** eintreffen, und sendet die Nachrichten, sobald sie eingetroffen sind, an die Abspieleinrichtung **154**. Die Abspieleinrichtung **154** macht das Arrangieren der Prozedur-Aufrufe rückgängig und gibt die Prozedur-Aufrufe an die Warteschlangenkomponente **86** aus.

[0069] Diese verschiedenen, zur QC-Architektur **130** gehörigen Teile werden nachfolgend genauer beschrieben.

Warteschlangenkomponenten

[0070] Die Warteschlangenkomponente **86** in der dargestellten QC-Architektur **130** ist ein COM-Objekt, welches die oben beschriebe Struktur aufweist und in **Fig. 3** dargestellt ist. Die Warteschlangenkomponente **86** ist dadurch gekennzeichnet, daß sie in einer Warteschlange angeordnete Aufrufe von Prozeduren unterstützt, indem sie den Schnittstellen der Komponente ein Attribut (das "QUEUEABLE"-Attribut) zuordnet. Der Entwickler einer Komponente weist den Schnittstellen der Komponente das "QUEUEABLE"-Attribut durch Hinzufügen eines Schnittstellenattribut-Makros (d.h. des Wortes "QUEUEABLE") zur Schnittstellensektion der Schnittstellendefinitionssprachbeschreibung der Komponentenklasse zu. Alternativ kann der Entwickler das "QUEUEABLE"-Attribut mittels des oben beschriebenen "COM+-Explorer" setzen, welcher graphische Bedienelemente in Objekteigenschaftfenstern ("sheet dialogs") zum Setzen der den Objekten in der dargestellten Ausführungsumgebung **80** zugewiesenen Attribute bereitstellt. In einigen erfindungsgemäßen Ausführungsformen kann das "QUEUEABLE"-Attribut in einem Katalog **165**, einer Registrierdatenbank des Betriebssystems oder einer anderen, mit dem Betriebssystem oder der spezifischen Anwendungssoftware assoziierten Konfi-

gurationsdatenbank gespeichert sein.

[0071] Einige Attribute werden außerdem mit der COM+-Anwendung assoziiert, in welche die Warteschlangenkomponente **86** gepackt ist. Ein "Queued App"-Attribut zeigt an, ob die Objekte der COM+-Anwendung über in einer Warteschlange angeordnete Aufrufe von Prozeduren aufgerufen werden können. Ein "Queue Listener"-Attribut zeigt an, ob die COM+-Anwendung eine Warteschlangen-Empfangseinrichtung, wie beispielsweise die Empfangseinrichtung **152** (Fig. 2), starten sollte. Ebenso umfaßt ein "Queue BLOB"-Attribut MSMQ-Namen (als GUID-Formatnamen (GUID = global eindeutiger Identifikator) einer Menge von der COM+-Anwendung zugeordneten Warteschlangen. ("BLOB" ist ein Akronym für "Binary Large Object" (binäres großes Objekt)). In der dargestellten Architektur **130** sind die "Queued App"- und die "Queue Listener"-Attribute als Boolesche Marken gespeichert, die "on" oder "off" gesetzt werden können. Das "Queue BLOB"-Marken speichert MSMQ-Namen von fünf verschiedenen Warteschlangen. Das "Queued App"-Attribut und das "Queue Listener"-Attribut können von einem Anwendungsintegrator mittels eines COM+-Explorer-Werkzeugs, welches Kontrollkästchen-Elemente zum Setzen der "Queued App"- und "Queued Listener"-Attribute in einem COM+-Anwendungs-Eigenschaftsfenster bereitstellt, betrachtet und gekennzeichnet werden. Wenn die "Queued App"- und "Queue Listener"-Attribute auf "on" gesetzt wurden, generiert das Programm als Reaktion darauf das "Queue BLOB"-Attribut.

[0072] Wenn die COM+-Anwendung erstmalig erstellt wird, werden die "Queued App"- und "Queue Listener"-Attribute auf "off" gesetzt und das "Queue BLOB"-Attribut ist nicht vorhanden.

[0073] Wenn das "Queued App"-Attribut auf "on" gesetzt wird (z.B. durch Auswählen eines Kontrollkästchens im COM+-Explorer), initiiert der COM+-Explorer eine API-Funktion in den COM+-Laufzeitdiensten, die eine Menge von MSMQ-Warteschlangen für die COM+-Anwendung erstellt und deren MSMQ-Namen im "Queue BLOB"-Attribut speichert. Zusätzlich werden sowohl das "Queued App"- als auch das "Queue Listener"-Attribut auf "on" gesetzt. Der Anwendungsintegrator hat dann die Möglichkeit, die Kontrollkästchen eines dieser Attribute im COM+-Explorer zu deaktivieren, um die Attribute auf "off" zu setzen. Das bedeutet, daß die COM+-Anwendung nur Echtzeit-Prozedur-Aufrufe nutzen soll, obwohl sie ein Objekt enthält, das eine Warteschlangenkomponente sein kann.

[0074] In der dargestellten QC-Architektur **130** werden die Schnittstellen der Warteschlangenkomponente vom Entwickler direkt mit dem "QUEUEABLE"-Attribut gekennzeichnet. Ein COM-Objekt gilt als eine in einer Warteschlange anordbare Komponente, wenn es mindestens eine als "QUEUEABLE" gekennzeichnete Schnittstelle besitzt. Befindet sich das Objekt in einer als eine "Queued App" gekennzeichneten COM+ Anwendung, gilt es als Warteschlangenkomponente. Allerdings kann die Komponente in alternativen Ausführungsformen der Erfindung auch selbst mit dem "QUEUEABLE"-Attribut gekennzeichnet werden. Alle Schnittstellen der gekennzeichneten Komponente, deren Prozeduren nur [in]-Parameter besitzen, würden dann als "QUEUEABLE" gelten.

Aktivierung einer Warteschlangenkomponente

[0075] Wie oben beschrieben, erstellt der Client **132** die Warteschlangenkomponente **86** unter Nutzung normaler COM-Aufrufsemantik, die auch für Echtzeit-Prozedur-Aufrufe genutzt wird. Genauer gesagt, erstellt der Client in der dargestellten QC-Architektur **130** die Warteschlangenkomponente **86** in einem Aufruf an die "CoGetObject()"-API (oder an die äquivalente "GetObject-"-API entsprechend der Programmiersprachensemantik von Visual Basic oder Visual Java), die eine gewöhnliche Objektinstanziierungs-API von COM ist. Der Client legt fest, daß die zu erstellende Warteschlangenkomponente mit in einer Warteschlange angeordneten Prozedur-Aufrufen genutzt wird, indem er "queue:/new" und anschließend die Programm-ID oder den Zeichenketten-GUID (Global eindeutiger Identifikator) der Warteschlangenkomponente als "displayname"-Parameter des "CoGetObject()"-API-Aufrufs festlegt. Als Reaktion darauf parst die "CoGetObject()"-API diese Zeichenkette in einen neuen Moniker ("new"-Moniker) und einen Warteschlangen-Moniker ("queue"-Moniker), welche die "CoGetObject()"-API anschließend veranlaßt, sich an die Warteschlangenkomponente zu binden. Der neue Moniker führt, wie auch die "CoCreateInstance()"-API, die Verarbeitung zur Erstellung einer Instanz des Serverobjekts durch. Der Warteschlangen-Moniker führt die Verarbeitung zum Vorbereiten der in Fig. 4 dargestellten Aufzeichnungseinrichtung aus, damit der Client die in einer Warteschlange angeordneten Aufrufe von Prozeduren für die Warteschlangenkomponente nutzen kann.

[0076] In alternativen Ausführungsformen kann der Client die Warteschlangenkomponente **86** erstellen, ohne in einer Warteschlange angeordnete Aufrufe von Prozeduren in der Instanziierungsanforderung explizit festzulegen, wenn die Warteschlangenkomponente die geeigneten Attribute aufweist (d.h. das mit seinen Schnittstel-

len oder Klasse verbundene QUEUEABLE-Attribut). Der Client fordert einfach die Erstellung der Warteschlangenkomponenten-Instanz über die "CoGetObject()"- oder die "CoCreateInstance()"-API an, und diese erstellen anschließend das Objekt als eine auf diesem Attribut basierende Warteschlangenkomponente.

[0077] Beispielhafte Objekt-Instanziierungsaufrufe sind in einer Programmliste **159** in Fig. 7 dargestellt.

Aufzeichnungseinrichtung

[0078] Unter genauerer Bezugnahme auf Fig. 4 ist die Aufzeichnungseinrichtung **130** in der dargestellten QC-Architektur **150** ein in einer COM+-DLL bereitgestelltes COM-Objekt. Die DLL der Aufzeichnungseinrichtung wird mit Hilfe eines Setup-Programms in ein COM+-bezogenes Verzeichnis auf dem Computer **20**, wie beispielsweise "d:.\Program Files\Microsoft\COM+", installiert. Die Installation veranlaßt außerdem die DLL der Aufzeichnungseinrichtung, sich selbst zu registrieren, was das Speichern von die Aufzeichnungseinrichtung **150** identifizierenden Konfigurationsinformationen (wie beispielsweise des Klassenidentifikators, des DLL-Pfadnamens, von Attributen usw.) im Katalog oder einer anderen Konfigurationsdatenbank umfaßt.

[0079] In der dargestellten QC-Architektur umfaßt das Proxy-Objekt **160** eine als Proxy-Manager handelnde Aufzeichnungseinrichtung **150**. Die Aufzeichnungseinrichtung steuert einen oder mehrere Schnittstellenproxies **166** und **167**, die eine Implementierung der Schnittstellen **87** der Warteschlangenkomponente **86** bereitstellen, um stellvertretend für die Warteschlangenkomponente **86** im Client-Prozeß **134** zu handeln und Aufrufe von Prozeduren des Clients **132** auf die Warteschlangenkomponente als direkte Aufrufe an die Proxy-Schnittstellen zu empfangen. Die Generierung der Schnittstellenproxies **166**–**167** erfolgt gemäß der Standard-Marshaling-Architektur des Microsoft COM RPC (d.h. sie werden aus Microsoft Schnittstellendefinitionssprachbeschreibungen(Microsoft Interface Definition Language)(MIDL))-Beschreibungen) der Warteschlangenkomponente **86** generiert) oder gemäß dem Marshaler der Microsoft Automation Type Bibliothek. (Für eine umfangreichere und detailliertere Erörterung des Microsoft COM RPC siehe Brockschmidt: "Inside OLE", 2. Auflage, 277–338 (Microsoft Press 1995)).

[0080] Genauer gesagt implementiert die als Proxy-Manager fungierende Aufzeichnungseinrichtung **150** eine "IUnknown"-Schnittstelle **162**. Wie im vorstehenden Abschnitt über Komponentenanwendungsobjekte beschrieben, fordert der Client **132** einen Schnittstellenzeiger an, indem er in einem Aufruf an die "QueryInterface()"-Prozedur der "IUnknown"-Schnittstelle **162** einen Schnittstellenidentifikator (IID) der Schnittstelle festlegt. Unter der Voraussetzung, daß die festgelegte IID Teil einer in einem Katalog **165** (Fig. 2) aufgeführten unveränderlichen IID-Gruppe ist, lädt und aggregiert die Aufzeichnungseinrichtung **150** einen Schnittstellen-Proxy **166**–**167** (mitunter auch als „Facelet" bezeichnet), der von einem MIDL-Übersetzer der Standard-Marshaling-Architektur generiert ist, um eine Proxy-Schnittstelle **168**–**169** für die entsprechende Schnittstelle der Warteschlangenkomponente, deren IID im Aufruf des Clients festgelegt ist, zu implementieren. Die Proxy-Schnittstellen **168**–**169** stimmen mit den Schnittstellen der Warteschlangenkomponente überein, welche abschließend instanziiert wird, um die in einer Warteschlange angeordneten Aufrufe von Prozeduren zu empfangen.

[0081] Die Aufzeichnungseinrichtung **150** implementiert außerdem eine "IRpcChannelBuffer"-Schnittstelle **170** und eine "IObjectControl"-Schnittstelle **172**. Die "IObjectControl"-Schnittstelle **172** ist eine Schnittstelle, die über den Microsoft Transaktionsserver (MTS) definiert und von der Aufzeichnungseinrichtung **150** genutzt wird, um Benachrichtigungen über die Deaktivierung eines Objekts gemäß der "Just-In-Time"-Aktivierungsfunktion von MTS (die in COM+ integriert ist) zu empfangen. Die "IRpcChannelBuffer"-Schnittstelle ist eine in der COM RPC Standard Marshaling-Architektur definierte Schnittstelle.

[0082] Die Schnittstellenproxies **166**–**167** werden vom MIDL-Übersetzer generiert, um die Prozeduraufrufe des Clients mit geeigneten Aufruf-Parametern und entsprechenden Daten aus dem Speicher des Client-Prozesses **134** in einen Puffer zu arrangieren. Entsprechend der Standard Marshaling-Architektur des Microsoft COM RPC nutzen die Schnittstellenproxies **166**–**167** die "IRpcChannelBuffer"-Schnittstelle **170** (die eine in der Standard-Marshaling Architektur definierte Standard-COM-Schnittstelle ist), um den Puffer an den ASP **90** der Warteschlangenkomponente zu übertragen. Statt allerdings den Prozedur-Aufruf über einen Echtzeit-RPC zu übertragen, zeichnet die Implementierung der "IRpcChannelBuffer"-Schnittstelle **170** in der Aufzeichnungseinrichtung **150** alle Prozeduraufrufe des Clients auf die Warteschlangenkomponente **86** (im Gegensatz zu den Aufrufen an die Prozeduren der "IUnknown"-Schnittstelle) in einem fortlaufenden Puffer auf. Die Aufzeichnungseinrichtung implementiert diese "IUnknown"-Prozeduren lokal und zeichnet somit diese Prozedur-Aufrufe im Puffer auf.

[0083] Nachdem der Client die Nutzung der Warteschlangenkomponente abgeschlossen hat (d.h., der Client gibt seinen Verweis an die Warteschlangenkomponente frei), leitet der MSQM-Ressourcen-Dispenser **176** den Puffer der Prozedur-Aufrufe an MSMQ weiter. Nach dem erfolgreichen Abschluß der Transaktion des Clients sendet MSMQ den die aufgezeichneten Prozedur-Aufrufe enthaltenden fortlaufenden Puffer als eine Nachricht an die Nachrichtenwarteschlange **158** der COM+-Anwendung, welche die Warteschlangenkomponente **86** enthält. Wird die Transaktion des Clients hingegen abgebrochen, verwirft der MSMQ den Puffer, sendet die Nachricht nicht, und die aufgezeichneten Prozedur-Aufrufe werden gelöscht. (In einigen Fällen, in denen ein Abbruch der Transaktion droht, leitet der MSMQ-Ressourcen-Dispenser den Puffer ganz einfach nicht an MSMQ weiter, da der Puffer bei einem Prozeß-Abbruch ohnehin verworfen würde.) Der MSMQ-Ressourcen-Dispenser **176** weist eine Schnittstelle **178** auf, um die Anfrage der Aufzeichnungseinrichtung zum Senden des gepufferten Prozedur-Aufrufs an die Nachrichtenwarteschlange zu empfangen. Der MSMQ-Ressourcen-Dispenser **176** stellt einen Cache von offenen MSMQ-Warteschlangen bereit und nutzt MSMQ-APIs, um den Puffer der Prozedur-Aufrufe über MSMQ an die Nachrichtenwarteschlange **158** zu senden.

Empfangseinrichtung

[0084] Unter erneuter Bezugnahme auf Fig. 2 ist die Empfangseinrichtung **152** ein von COM+ bereitgestelltes Objekt, das die die Warteschlangenkomponente **86** enthaltende Nachrichtenwarteschlange **158** der COM+-Anwendung überwacht. Die Empfangseinrichtung **152** wird beim Starten der COM+-Anwendung erstellt, wenn die COM+-Anwendung die in ihrem "Queue BLOB"-Attribut festgelegte Nachrichtenwarteschlange aufweist und die Nachrichtenwarteschlangen-Empfangseinrichtung auf "on" gesetzt ist. Die Empfangseinrichtung **152** öffnet die Nachrichtenwarteschlange **158** der COM+-Anwendung. und wartet auf den Eingang von Nachrichten. Sobald Nachrichten eingehen, fertigt die Empfangseinrichtung **152** einen Aktivitätsträger (Thread) ab, um eine Instanz der Abspieleinrichtung **154**, welche die Nachrichten aufnimmt und verarbeitet, auszuführen. In der dargestellten QC-Architektur **130** ist eine einzelne Empfangseinrichtung **152** pro ASP **90** gezeigt.

[0085] Beim Starten der Empfangseinrichtung **152** kann die Nachrichtenwarteschlange **158** möglicherweise eine große Menge an Nachrichten von in einer Warteschlange angeordneten Aufrufen von Prozeduren mehrerer Clients **132** enthalten. Darüber hinaus kann die Empfangseinrichtung **152** verschiedene Nachrichtenwarteschlangen für COM+-Anwendungen im ASP **90** überwachen. Die Empfangseinrichtung **152** nutzt dagegen vorzugsweise einen Thread-Pool von geringerer Größe, von welchem die Threads der Abspieleinrichtung so verteilt werden, daß der Nachrichtenverarbeitungsdurchsatz maximiert wird. Die geringe Menge der Threads verhindert eine Überlastung des Prozessors des Server-Computers. Da ferner die in einer Warteschlange angeordneten Aufrufe von Prozeduren keine Echtzeit-Antwort erfordern, besteht nicht die Notwendigkeit, neue Threads für die in einer Warteschlange angeordneten Nachrichten sofort zu planen. Alternativ kann das Sammeln von Objektinstanzen der Warteschlangenkomponente mit einer nach oben und unten beschränkten Anzahl von gesammelten Objekten genutzt werden, um eine Überlastung zu verhindern. Bei mehreren Nachrichtenwarteschlangen verteilt die Empfangseinrichtung **152** die Abfertigung der Threads der Abspieleinrichtung vorzugsweise angemessen auf die Warteschlangen, so daß die Nachrichten einer Warteschlange noch nicht vollständig abgefertigt sind, bevor die Abfertigung an einer anderen Schlange beginnt.

Abspieleinrichtung

[0086] Wie in Fig. 5 dargestellt, ist die Abspieleinrichtung **154** in ein in einer COM+-Utility-Bibliothek (ebenfalls eine DLL-Datei im COM+-Verzeichnis) bereitgestelltes COM-Objekt implementiert. Wie oben beschrieben, wird die Abspieleinrichtung **154** im ASP **90** der Empfangseinrichtung durch die Empfangseinrichtung **152** erstellt und aufgerufen, wenn eine Nachricht mit Prozeduraufrufen für die Warteschlangenkomponente **86** eingeht. Die Abspieleinrichtung **154** hat ein auf "Transaktion angefordert" gesetztes Transaktionsattribut-Set. Dieses veranlaßt die COM+-Ausführungsumgebung **80**, automatisch eine Transaktion zu starten, im Zuge welcher die Abspieleinrichtung gemäß der in obengenannter Patentanmeldung für Automatische Transaktionen beschriebenen Automatischen Transaktionsarchitektur erstellt wird. Die von der Abspieleinrichtung **154** an die Warteschlangenkomponente **86** wieder abgespielten Prozedur-Aufrufe werden außerdem automatisch mit dieser Transaktion in Beziehung gesetzt.

[0087] Nach der Erstellung ruft die Abspieleinrichtung **154** in der Empfangseinrichtung **152** Routinen auf, um die in einer Warteschlange angeordnete Nachricht, welche Prozedur-Aufrufe auf die Warteschlangenkomponente **86** enthält, abzurufen. Die Abspieleinrichtung **154** holt zunächst einen Puffer und nutzt anschließend die Routinen der Empfangseinrichtung, um die Nachrichten über eine "MQReceiveMessage"-API-Operation (ein Standard-MSMQ-API-Verfahren) in den Puffer zu laden. Die "MQReceiveMessage"-API-Operation wurde zum

Teil der Transaktion der Abspieleinrichtung gemacht.

[0088] Danach instanziiert die Abspieleinrichtung **154** die Warteschlangenkomponente **86** im ASP **90** und fungiert als ein Stub-Manager in einem Stub-Objekt **180**, der Schnittstellen-Stubs **182–183** steuert, die entsprechend der Standard Marshaling Architektur des Microsoft COM RPC (d. h. aus den Microsoft Schnittstellendefinitionssprachbeschreibungen (MIDL-Beschreibungen) der Warteschlangenkomponente **86**) oder entsprechend dem Marshaler der Microsoft Automation Type Library generiert sind. Als Stub-Manager lädt die Abspieleinrichtung **154** die (mitunter auch als "Stublets" bezeichneten) Schnittstellen-Stubs **182–183** für die Schnittstelle **87** der Warteschlangenkomponente, sobald deren entsprechende Schnittstellenidentifikatoren (IIDs) in der Nachricht auftreten. Die Abspieleinrichtung **154** nutzt die Stublets **182–183**, um das Arrangieren der Prozedur-Aufruf-Daten aus der Nachricht rückgängig zu machen und die rückumgewandelten Prozedur-Aufrufe an die Warteschlangenkomponente **86** auszugeben. Die Abspieleinrichtung **154** interpretiert außerdem von der Aufzeichnungseinrichtung **150** eingegebene Sicherheitsköpfe durch Aufrufen der geeigneten Sicherheitsdienste.

Ständige Server-seitige Abbrüche

[0089] Die Empfangseinrichtung **152** und die Abspieleinrichtung **154** kooperieren, um "vergiftete Nachrichten" oder "ständige Server-seitige Abbrüche" zu behandeln. (Eine "vergiftete Nachricht" ist im gegebenen Kontext eine Nachricht, die Prozedur-Aufrufe umfaßt, die ständig zu Transaktionsabbrüchen führen, wenn sie an die Warteschlangenkomponente wieder abgespielt werden). Genauer gesagt wird, wenn es beim Wiederabspielen einer Gruppe von Prozedur-Aufrufen an die Warteschlangenkomponente **86** zu einem Abbruch der Transaktion der Abspieleinrichtung kommt, die Arbeit der Warteschlangenkomponente wiederholt. Der Transaktionsabbruch bewirkt zudem, daß MSMQ als Teil seiner Verarbeitung des Transaktionsabbruchs die Prozedur-Aufruf-Nachricht zurück in die Nachrichtenwarteschlange **158** der COM+-Anwendung für einen Wiederholungslauf verschiebt. In der dargestellten QC-Architektur **130** bricht die Abspieleinrichtung **154** die Transaktion bei Empfang eines einen Fehler oder einen Ausfall anzeigenden Rückgabewerts (z. B. eines HRESULT-Fehlerwerts) von der Warteschlangenkomponente **86** ab. Falls die Verarbeitung eines Prozedur-Aufrufs in der Warteschlangenkomponente einen Abbruch gewährleistet, kann die Warteschlangenkomponente **86** außerdem die "SetAbort()"-Prozedur (wie in der obengenannten Patentanmeldung für Automatische Transaktionen beschrieben) aufrufen, um die Transaktion abzubrechen.

[0090] In der dargestellten QC-Architektur **130** sind fünf der COM+-Anwendung zugewiesene Nachrichtenwarteschlangen gezeigt, und zwar eine normale Eingabewarteschlange, eine erste Wiederholungswarteschlange, eine zweite Wiederholungswarteschlange, eine dritte Wiederholungswarteschlange und eine Endablage-Warteschlange. Die QC-Architektur nutzt diese Warteschlangen zur Behandlung von ständigen Server-seitigen Abbrüchen. Insbesondere durchläuft die QC-Architektur **130**, falls das Wiederabspielen der Prozedur-Aufrufe an die Warteschlangenkomponente **86** mehrmals zu einem Transaktionsabbruch führt, folgende Schritte: (1) die Abspieleinrichtung **154** empfängt die Prozedur-Aufruf-Nachricht von der Warteschlange der COM+-Anwendung; (2) die Abspieleinrichtung **154** instanziiert die Warteschlangenkomponente **86** und spielt die Prozedur-Aufrufe an die Warteschlangenkomponente ab; (3) die Transaktion wird abgebrochen und wiederholt und (4) MSMQ sendet die Nachricht an den Anfang der Warteschlange der COM+-Anwendung, von welcher sie abgerufen wurde.

[0091] Bei nachfolgenden Abbrüchen durchlauft die Abspieleinrichtung **154** mit der „vergifteten Nachricht" die Folge der Wiederholungswarteschlangen der COM+-Anwendung. Alternative Ausführungsformen der QC-Architektur können eine andere Anzahl an Wiederholungen und Intervallen als die nachfolgend für die dargestellte Architektur beschriebenen nutzen. Insbesondere sendet die Abspieleinrichtung **154** eine nach einem Abbruch an die normale Eingabewarteschlange zurückgesandte Nachricht an die erste Wiederholungswarteschlange. Die Empfangseinrichtung **152** bedient die erste Wiederholungswarteschlange ein Mal pro Minute, bis diese abgearbeitet ist, wobei diese die Nachricht über die Abspieleinrichtung **154** in einem Intervall von etwa einer Minute an die Warteschlangenkomponente abspielt.

[0092] Wurde die Nachricht zum dritten Mal an die erste Wiederholungswarteschlange zurückgesandt, sendet die Abspieleinrichtung **154** die Nachricht an die zweite Wiederholungswarteschlange. Die Empfangseinrichtung **152** bedient die zweite Widerholungswarteschlange im Drei-Minuten-Takt, bis diese abgearbeitet ist. Diese wiederum spielt die Nachricht über die Abspieleinrichtung **154** in denselben Zeitabständen an die Warteschlangenkomponente **86** ab.

[0093] Wurde die Nachricht zum dritten Mal an die zweite Wiederholungswarteschlange zurückgesandt, sen-

det die Abspieleinrichtung **154** die Nachricht an die dritte Wiederholungswarteschlange. Die Empfangseinrichtung **152** bedient die dritte Wiederholungswarteschlange im Fünf-Minuten-Takt, bis diese abgearbeitet ist. Diese wiederum spielt die Nachricht über die Abspieleinrichtung **154** in denselben Zeitabständen an die Warteschlangenkomponente **86** ab.

[0094] Wurde die Nachricht zum fünften Mal an die dritte Wiederholungswarteschlange zurückgesandt, sendet die Abspieleinrichtung **154** die Nachricht an die Endablage-Warteschlange der COM+-Anwendung. Die Endablage-Warteschlange wird allerdings nicht von der Empfangseinrichtung **152** bedient. Die Nachricht verbleibt in der Endablage-Warteschlange, bis sie (mit Hilfe des unten beschriebenen Programms zum Verschieben von Warteschlangenkomponenten-Nachrichten) verschoben oder mittels des MSMQ-Explorers (einem Hilfsprogramm zur Steuerung von MSMQ) bereinigt wird.

[0095] Bei jedem Abbruch erstellt die Abspieleinrichtung **154** eine Ereignislog-Nachricht. Die Abspieleinrichtung **154** erstellt eine zusätzliche Ereignislog-Nachricht, wenn eine Nachricht aus einer Warteschlange in eine andere Warteschlange verschoben wird. Sie erstellt außerdem eine weitere Nachricht, wenn die Nachricht in die Endablage-Warteschlange verschoben wird. Die Abspieleinrichtung **154** kann die Nachricht vor dem Verschieben in eine andere Warteschlange zusätzlich so verändern, daß die Nachricht mehr Diagnoseinformation enthält.

[0096] Die Warteschlangenkomponente **86** kann optional einen zugeordneten Ausnahmezustandsbehandler **188** umfassen, der durch einen für die Warteschlangenkomponente **86** im Katalog **165** eingetragenen „Exception_CLSID"-Eintrag festgelegt ist. Der Ausnahmezustandsbehandler ist ein COM-Objekt, das eine „IPlaybackControl"-Schnittstelle **190** unterstützt. Wenn der Ausnahmezustandsbehandler **188** für die Warteschlangenkomponente **86** registriert wird, ruft die Abspieleinrichtung **154** zunächst die "IPlaybackControl"-Schnittstelle **190** des Ausnahmezustandsbehandlers auf, bevor sie jegliche Prozedur-Aufrufe von der Nachricht wieder abspielt. Die Warteschlangenkomponente **86** kann die "IPlaybackControl"-Schnittstelle **190** selbst bereitstellen und sich selbst als Ausnahmezustandsbehandler im "Exception_CLSID"-Eintrag festlegen. Wird kein "Exception_CLSID"-Eintrag festgelegt, sucht die Abspieleinrichtung **154** außerdem nach diesem und ruft die "IPlaybackControl"-Schnittstelle **190** an der Warteschlangenkomponente **86** zur Ausnahmebehandlung auf.

[0097] Die "IPlaybackControl"-Schnittstelle **190** dient dazu, den Ausnahmezustandsbehandler **188** zu informieren, daß eine Nachricht kurz davor steht, in die Endablage-Warteschlange verschoben zu werden, um so eine alternative Behandlung des ständigen Abbruchs zu ermöglichen. Der Ausnahmezustandsbehandler kann die Sammlung von Problemdiagnose-Informationen oder die Erzeugung eines Objekts oder einer Nachricht, welches bzw. welche den Client über ein konsistentes und ernstes Problem als Reaktion auf die Aufrufe der Abspieleinrichtung an diese Schnittstelle informiert, implementieren. Besitzt die Warteschlangenkomponente **86** keinen zugehörigen Ausnahmezustandsbehandler, wird die "vergiftete Nachricht" einfach in die Endablage-Wartschlange verschoben, wenn die Wiederholungen wie oben beschrieben abgearbeitet wurden. Existiert dagegen ein Ausnahmezustandsbehandler, ruft die Abspieleinrichtung **154** die Prozeduren der "IPlaybackControl"-Schnittstelle ein letztes Mal auf, bevor die Nachricht in die Endlage-Warteschlange verschoben werden würde. Führt das Wiederabspielen des Prozeduraufrufs an die Warteschlangenkomponente **86** bei diesem letzten Mal immer noch zu einem Abbruch, wird die „vergiftete Nachricht" in die Endablage-Warteschlange verschoben.

[0098] Auf der Client-Seite kann es andererseits vorkommen, daß MSMQ die Nachricht nicht an die Eingabewarteschlange der COM+-Anwendung weiterleiten kann. Dies kann beispielsweise der Fall sein, wenn die Warteschlangen-Zugangskontrolle verhindert, daß die Nachricht vom Client an den Server gesandt wird. In einem solchen Fall sendet MSMQ die Nachricht an eine Client-seitige "Xact Dead Letter"-Warteschlange des Warteschlangenmanagers. Die QC-Architektur **130** stellt eine Vorrichtung zur Überwachung von Warteschlangen für unzustellbare Nachrichten **194** bereit. Die Überwachungsvorrichtung **194** instanziiert den durch den "Exception_CLSID"-Katalog-Eintrag für die Warteschlangenkomponente **86** auf dem Client-Computer **92** festgelegten Ausnahmezustandsbehandler **188** und gibt einen "QueryInterface()"-Aufruf mit dem IID der "IPlaybackControl"-Schnittstelle **190** aus. Verläuft dies erfolgreich, ruft die Überwachungsvorrichtung die "IPlaybackControl::FinalClientRetry()"-Prozedur auf und spielt die Prozedur-Aufrufe von der Nachricht an die Client-seitige Implementierung der Warteschlangenkomponente **86** wieder ab. Diese Client-seitige Warteschlangenkomponente kann optional Diagnoseinformation bewahren oder Maßnahmen ergreifen, um die Wirkung einer vorangegangenen Transaktion umzukehren (auszugleichen). Wird das Wiederabspielen ausgeführt, dann wird die Nachricht aus der Warteschlange für unzustellbare Nachrichten entfernt. Wird das Wiederabspielen abgebrochen oder sind der Ausnahmezustandsbehandler **188** und die "IplaybackControl"-Schnittstelle **190** nicht

verfügbar, verbleibt die Nachricht in der Warteschlange für unzustellbare Nachrichten zur manuellen Behandlung.

Programm zum Verschieben von Nachrichten von Warteschlangenkomponenten

[0099] Die QC-Architektur **130** stellt ein als „Programm zum Verschieben von Warteschlangenkomponenten" bezeichnetes Verwaltungswerkzeug bereit, das es dem Systemverwalter oder einer anderen Person ermöglicht, Nachrichten manuell aus einer Warteschlange der COM+-Anwendung in eine andere zu verschieben. Wie oben beschrieben, behandelt die QC-Architektur **130** ständige Server-seitige Abbrüche, indem sie die Nachricht in eine Endablage-Warteschlange verschiebt und so verhindert, daß die Empfangseinrichtung **152** und die Abspieleinrichtung **154** die Nachricht in einer „Endlosschleife" ununterbrochen wiederholen. In einigen Fällen läßt sich die Ursache der ständigen Abbrüche auf dem Server-Computer **84** beheben. So können Abbrüche beispielsweise dadurch hervorgerufen werden, daß eine Ressource (z. B. eine Datenbank) nicht verfügbar ist. Nach einer manuellen Berichtigung dieser Situation kann die zuvor vergiftete Nachricht mit Hilfe des Programms zum Verschieben von Nachrichten von Warteschlangenkomponenten manuell zurück in die Eingabewarteschlange der COM+-Anwendung verschoben werden, um so weitere Wiederholungen zu ermöglichen. Das Programm zum Verschieben Nachrichten von Warteschlangenkomponenten verschiebt die Nachrichten vorzugsweise als eine Transaktion, so daß keine Nachrichten verloren gehen oder dupliziert werden, falls es während des Verschiebens zu einem Fehler kommt. Das Programm zum Verschieben von Nachrichten von Warteschlangenkomponenten kann programmäßig über OLE Automation, beispielsweise unter Nutzung einer Visual Basic Script, betrieben werden.

Schnittstellen

[0100] Fig. 8 zeigt eine Programmliste **200**, welche die "IPlaybackControl"-Schnittstelle **190** definiert. Wie oben beschrieben, wird diese Schnittstelle durch den Ausnahmezustandsbehandler **188** implementiert, der im Katalog **165** für die Warteschlangenkomponente **86** registriert ist, um sich an der Nicht-Standard-Behandlung von ständigen Server-seitigen Abbrüchen und Client-seitigen MSMQ-Übermittlungsfehlern zu beteiligen. Die Schnittstelle **190** umfaßt eine "FinalClientRetry()"-Prozedur und eine "Final ServerRetry()"-Prozedur.

[0101] Die "FinalClientRetry()"-Prozedur informiert den Ausnahmezustandsbehandler **188** auf dem Client-Computer **92** (falls dieser definiert ist), daß alle Versuche, die Nachricht über MSMQ an den Server-Computer **84** zu übermitteln, abgelehnt wurden und die Nachricht in der "Xact Dead Letter"-Warteschlange (Xact-Warteschlange für unzustellbare Nachrichten) gelandet ist. So kann es möglicherweise sein, daß die Berechtigungen der Warteschlange eine Übertragung der Nachricht an die Warteschlange nicht zulassen. Sobald Nachrichten in der "Xact Dead Letter"-Warteschlange eingehen, ruft die QC-Architektur **130** die "FinalClientRetry()"-Prozedur auf, um so den Ausnahmezustandsbehandler zu informieren. Der Ausnahmezustandsbehandler kann dann eine auf die Warteschlangenkomponentenklasse bezogene Ausnahmemaßnahme, wie beispielsweise das Aufzeichnen des Fehlers in anwendungsspezifischer Sprache oder das Senden einer Mail-Nachricht an den Endbenutzer oder Verwalter, oder sogar eine Client-seitige Kompensationsmaßnahme, wie beispielsweise das Umkehren der Wirkung einer vorangegangenen Transaktion, vornehmen. Falls der für die Warteschlangenkomponente identifizierte Ausnahmezustandsbehandler diese Schnittstelle nicht implementiert oder deren "FinalClientRetry()"-Prozedur-Aufruf ein Fehlerergebnis zurücksendet, verbleibt die Nachricht in der Xact-Warteschlange für unzustellbare Nachrichten.

[0102] Die "FinalServerRetry()"-Prozedur informiert den Ausnahmezustandsbehandler der Warteschlangenkomponente auf dem Server-Computer **84**, daß alle Versuche, die zurückgestellte Aktivierung an die Warteschlangenkomponente wieder abzuspielen, gescheitert sind (z. B. durch einen HRESULT-Fehler oder einen Transaktionsabbruch) und daß die Nachricht dabei ist, in die Endablage-Warteschlange der COM+-Anwendung verschoben zu werden. Der Server-seitige Ausnahmezustandsbehandler kann eine auf die Warteschlangenkomponentenklasse bezogene Ausnahmemaßnahme, wie beispielsweise das Aufzeichnen des Fehlers in anwendungsspezifischer Sprache oder das Senden einer Mail-Nachricht an den Endbenutzer oder Verwalter, oder sogar eine Client-seitige Maßnahme, wie beispielsweise das Umkehren der Wirkung einer vorangegangenen Transaktion, vornehmen. Vorzugsweise sollte die Warteschlangenkomponente jede Anstrengung unternehmen, um diese Transaktion erfolgreich abzuschließen, andernfalls ist für eine Wiederverarbeitung der Nachricht ein manueller Eingriff erforderlich. Falls der für die Warteschlangenkomponenten-Klasse registrierte Ausnahmezustandsbehandler die "PlaybackControl"-Schnittstelle nicht implementiert oder die Implementierung des "FinalServerRetry()"-Aufrufs ein Fehlerergebnis zurücksendet oder die Transaktion abgebrochen wird, erfolgt die Verschiebung der Nachricht in die Endablage-Warteschlange.

[0103] Wie aus Fig. 9 und Fig. 10 ersichtlich, weist eine die Prozedur-Aufrufe des Clients umfassende Nachricht **220**, die von der Aufzeichnungseinrichtung **130** in der QC-Architektur **130** aufgezeichnet wird, ein Format auf, das in Fig. 10 dargestellt ist und Datenstrukturen nutzt, die in der in Fig. 9 dargestellten Programmliste **220** definiert sind. Gemäß diesem Nachrichtenformat umfaßt die Nachricht **220** einen Nachrichtenkopf **224** und einen Nachrichtenkörper **226**. Der Nachrichtenkörper **226** umfaßt einen Behälter(container)-Abschnitt **228** und einen oder mehrere Prozedur-, Sicherheits- oder Diagnoseabschnitte **230**. Die Behälter-, Prozedur-, Sicherheits- und Diagnoseabschnitte **228** und **230** beginnen mit einem geeigneten Abschnittstyp-Kopf **234**, **236**. Die Abschnittsköpfe **234**, **236** umfassen weiterhin alle einen gemeinsamen Kopf (die "_CommonHeader"-Struktur in Fig. 9). Der Behälterabschnitt **228** besitzt einen Behälterabschnittskopf **234**, wohingegen die Prozedurabschnitte jeweils einen langen oder kurzen Prozedurkopf **236** und Parameterdaten **238** umfassen.

[0104] Der Nachrichtenkopf **224** enthält eine Menge an entweder von der Aufzeichnungseinrichtung oder von MSMQ gesetzten Nachrichteneigenschaften, die mit dem Nachrichtenkörper übermittelt werden. Diese Nachrichteneigenschaften umfassen eine Priorität, eine Nachrichten-ID, eine Korrelations-ID, einen MSMQ-Pfadnamen und einen Antwortwarteschlangennamen. Die Priorität ist ein Wert, der gesetzt werden kann, um die Reihenfolge zu bestimmen, in welcher die Nachrichten von der Eingabewarteschlange der COM+-Anwendung wieder abgespielt werden. Ein höherer Prioritätswert kann von der Client-Anwendung zugewiesen werden, damit bestimmte Operationen bevorzugt verarbeitet werden, wie beispielsweise bei einer Bankanwendung die Bevorzugung der Autorisierung von Kreditkarten vor der Verarbeitung von Schecks. Die Nachrichten-ID identifizert die einzelne Nachricht. Die Korrelations-ID wird in der QC-Architektur **130** genutzt, um eine Menge von in einer Warteschlange angeordnete Prozedur-Aufrufe enthaltenden Nachrichten in einem Arbeitsfluß zu gruppieren. Die ursprüngliche Nachricht, die den Arbeitsfluß beginnt, erstellt eine Nachricht mit einer nicht vorhandenen Korrelations-ID. Die Nachrichten-ID der ursprünglichen Nachricht wird zur Korrelations-ID aller innerhalb des Arbeitsflusses nachfolgenden Nachrichten. Der MSMQ-Pfadname besteht aus dem Namen eines Zielcomputers, einem linksseitigen Schrägstrich (d.h. "\") und einem Namen einer Warteschlange auf dem Bestimmungscomputer, z.B. "MachineName\payroll". Ein Punkt als Maschinenname kennzeichnet "diesen Computer", d.h. den Client-Computer. Der Name der Antwortwarteschlange kennzeichnet eine Warteschlange für Antwortnachrichten vom Server-Computer.

[0105] Der (in der Programmliste **206** von Fig. 9 durch die Strukturdeklaration "_CommonHeader" definierte) gemeinsame Kopf **232**, **233** in jedem der Abschnitte **228**, **230** umfaßt zwei Werte, und zwar einen Typ und eine Länge. Der Typwert identifiziert, wie in der nachstehenden Tabelle gezeigt, den Typ des Abschnitts. Der Längenwert bezieht sich auf die Länge des Abschnittkopfs und dessen Inhalte. Folglich wird durch das Hinzufügen der Länge an einen Zeiger auf den aktuellen "_CommonHeader" zum nächsten "_CommonHeader" übergegangen.

Tabelle 1. Abschnittstypen-Feld	
Abschnittstyp	Feldwert
Behälter	"CHDR"
Prozedur	"METH"
Kurzprozedur	"SMTH"
Sicherheit	"SECD"
Sicherheitsverweis	"SECR"
Diagnose	"DIAG"

[0106] Der Behälter-Abschnitt **228** umfaßt einen Behälter-Kopf **234**. Wie durch die "_ContainerHeader"-Strukturdeklaration in Fig. 9 definiert, umfaßt der Behälter-Kopf **234** eine GUID-Signatur, Versionsnummern und einen Moniker zur Erstellung des Server-Objekts. Die GUID-Signatur ("guidSignature") identifiziert die Nachricht als eine Nachricht von einer Aufzeichnungseinrichtung an eine Abspieleinrichtung in der QC-Architektur **130**. Unterschiedliche GUID-Signatur-Werte können genutzt werden, um verschiedene Abspieleinrichtungs-Klassen zu identifizieren. Der Moniker ist eine Streamform eines COM-Monikers zur Erstellung der Warteschlangenkomponente **86**. In alternativen Architekturen kann der Behälter-Kopf zusätzliche Werte umfassen, wie beispielsweise eine Nachrichtenprotokollversionsnummer und Client-seitige Version-Identifikatoren.

[0107] Der (in den Strukturdeklarationen "_ShortMethodheader" und "_MethodHeader" in Fig. 9 definierte) Prozedur-Kopf **236** stellt einen Prozedur-Aufruf auf die Warteschlangenkomponente **86** dar. Der Proze-

dur-Kopf kann ein langer oder ein kurzer Kopf sein, wobei der Unterschied darin besteht, daß der lange Prozedur-Kopf einen Schnittstellenidentifikator (IID) festlegt. Der IID für den kurzen Prozedur-Kopf wird vom neuesten langen Prozedur-Kopf, der in einem Links-Nach-Rechts-Durchlauf der gesamten Nachricht aufgetreten ist, impliziert. Der Prozedur-Kopf **236** umfaßt einen Prozedur-Identifikator, eine Datenrepräsentation, ein Markierungsfeld, eine Länge, eine Unterbrechungsmarkierung und einen IID (für einen langen Prozedur-Kopf). Der Prozedur-Identifikator ("dwMethodId") ist ein Index der Tabelle **104** der virtuellen Funktionen (Fig. 3) der aufzurufenden Prozedur. Die Datenrepräsentation umfaßt den "RPCOLEDATREP"-Wert von einer "RPCOLEMESSAGE" gemäß der Vereinbarung der Standard Marshaling Architektur von COM RPC, der einer der durch den Schnittstellenproxy **166**–**167** an die "IRPCChannelBuffer"-Schnittstelle **170** der Aufzeichnungseinrichtung **150** während des Arrangierens des Prozedur-Aufrufs (Fig. 4) gemäß der Vereinbarung der Standard Marshaling-Architektur von COM RPC dargestellten Parameter ist. Das Markierungsfeld umfaßt das "RPCOLEMESSAGE rpcFlags"-Feld gemäß der Vereinbarung der Standard Marshaling-Architektur von COM RPC, das auch von den Schnittstellenproxies an die Aufzeichnungseinrichtung **150** während des Arrangierens des Prozedur-Aufrufs dargestellt wird. Das Längenfeld umfaßt das "RPCOLEMESSAGE cbBuffer"-Feld gemäß Vereinbarung der Standard Marshaling-Architektur von COM RPC, das auch durch die Schnittstellenproxies an die Aufzeichnungseinrichtung **150** während des Arrangierens des Prozedur-Aufrufs dargestellt wird. Die Unterbrechungsmarkierung umfaßt entweder 0 oder 1, um die verwendete Arrangier-Prozedur anzuzeigen. Eine 0 bedeutet, daß ein MDL-generierter Schnittstellenproxy genutzt wurde. Der IID in einem langen Prozedur-Kopf zeigt die Schnittstelle derjenigen Warteschlangenkomponente **86** an, deren Prozedur aufgerufen wurde.

[0108] Die Parameterdaten **238** im Prozedur-Abschnitt sind die vom Schnittstellenproxy **166**–**167** des Prozedur-Aufrufs erzeugten, arrangierten Parameterdaten.

[0109] Der Nachrichtenkörper **226** kann außerdem Sicherheitskopf-Abschnitte, Sicherheitsverweis-Abschnitte und Diagnoseabschnitte umfassen, wie sie im Abschnittstyp des gemeinsamen Kopfs des Abschnitts identifiziert sind (die Werte sind in obenstehender Tabelle 1 abgebildet). Ein Sicherheitskopf-Abschnitt umfaßt einen Sicherheitskopf entsprechend der "_SecurityHeader"-Datenstrukturdeklaration von Fig. 9. Ein Sicherheitskopf-Abschnitt umfaßt an der Aufzeichnungseinrichtung extrahierte Sicherheitsinformation, die zur Überprüfung des Zugangsprivilegs beim Wiederabspielen von Prozeduraufrufen genutzt wird. Diese Sicherheitsinformationen können sich von Prozedur zu Prozedur unterscheiden. Der Sicherheitsverweis-Abschnitt umfaßt einen Sicherheitsverweiskopf, der auf einen vorherigen Sicherheitskopf-Abschnitt in der Nachricht verweist, um eine Wiederholung der Sicherheitsinformationen zu vermeiden, wenn dieselben Sicherheitsinformationen einen nachfolgenden Prozedur-Aufruf in der Nachricht betreffen. Der Diagnose-Abschnitt besteht aus Daten, die an den Nachrichtenkörper angefügt werden, wenn die Nachricht nach einem Server-seitigen Abbruch zwischen Warteschlangen der COM+-Anwendung verschoben wird. Die Diagnosedaten können beispielsweise die Zeit und die Ursache für den Fehler umfassen. Die Diagnose-Abschnitte werden von der Abspieleinrichtung **154** während des Wiederabspielens der Nachricht ignoriert, können allerdings zur Vereinfachung eines manuellen Eingriffs genutzt werden (z.B. nachdem die Nachricht in der Endablage-Warteschlange der COM+-Anwendung eingeht).

[0110] Nachdem Prinzipien unserer Erfindung unter Bezugnahme auf eine dargestellte Ausführungsform beschrieben und dargestellt wurden, ist festzuhalten, daß die dargestellte Ausführungsform hinsichtlich ihrer Gestaltung und Details geändert werden kann, ohne daß sie von diesen Prinzipien abweicht. Es versteht sich, daß, sofern nichts Anderweitiges angegeben ist, die vorstehend beschrieben Programme, Prozesse oder Verfahren sich weder auf einen bestimmten Typ einer Computervorrichtung beziehen noch auf einen solchen beschränkt sind. Verschiedene Computervorrichtungstypen für allgemeine oder spezielle Anwendungen können in Verbindung mit der vorstehend beschriebenen Lehre genutzt werden oder dieser entsprechende Operationen ausführen. Bestandteile der dargestellten, in Software abgebildeten Ausführungsform können in Hardware implementiert werden und umgekehrt.

[0111] Angesichts der zahlreichen möglichen Ausführungsformen, auf welche sich die Prinzipien unserer Erfindung anwenden lassen, ist festzuhalten, daß die dargelegten Ausführungsformen lediglich illustrativen Charakter besitzen und nicht als den Umfang unserer Erfindung einschränkend zu betrachten sind. Vielmehr beanspruchen wir als unsere Erfindung jedwede Ausführungsformen, die in den Bereich der nachfolgenden Ansprüche fallen, sowie diesbezügliche Äquivalente.

Patentansprüche

1. Objektlaufzeitdienste-System (**130**) in einem verteilten Computernetzwerk zum Laufenlassen von Objekten (**86**) auf einem Computer, wobei die Objekte auf objektseitigen Computer Schnittstellen (**87**) mit Proze-

duren zum Aufrufen durch Clients (**132**) auf Client-seitigen Computer in dem verteilten Computernetzwerk aufweisen, wobei ein mit den Objektschnittstellen (**87**) verbundenes Attribut das Objekt als in einer Wartschlange angeordnete Aufrufe von Prozeduren unterstützend identifiziert, wobei das System umfaßt:
eine Referenz zur Verwendung durch einen Client (**132**) auf einem Client-seitigen Computer zum asynchronen Aufrufen von Prozeduren eines Objekts;
eine Aufzeichnungseinrichtung (**150**) auf einem Client-seitigen Computer, die betreibbar ist, um als ein Proxy zu fungieren und stellvertretend für das Objekt (**86**) zu handeln und Aufrufe von Prozeduren des Clients für das Objekt als direkte Aufrufe zum stellvertretenden Handeln für Schnittstellen zu empfangen, wodurch eine Unterbrechung einer Vielzahl von Aufrufen von Prozeduren bewirkt wird, die in einer ersten Transaktion durch einen Client (**132**) abgegeben wurden, um Prozeduren eines Objekts (**86**) auf einem Objekt-seitigen Computer aufzurufen, und die Aufrufe von Prozeduren als eine Nachricht aufzuzeichnen;
eine Nachrichtenwarteschlange (**158**) zum Einreihen der Nachricht zu einem späteren Zeitpunkt, wobei die Nachricht bei erfolgreichem Abschluß der ersten Transaktion an die Nachrichtenwarteschlange übergeben wird; und
eine Abspieleinrichtung (**154**) auf dem Objekt-seitigen Computer, die betreibbar ist, um die Aufrufe von Prozeduren aus der Nachricht zu extrahieren und die Aufrufe von Prozeduren an das Objekt zu einem späteren Zeitpunkt in einer zweiten Transaktion auszugeben.

2. System nach einem vorangehenden Anspruch, dadurch gekennzeichnet, daß die Aufzeichnungseinrichtung (**150**) bewirkt, daß eine Datenstromdarstellung (**230**) der Vielzahl von Aufrufen von Prozeduren in einer Prozeduraufrufnachricht zur Eingabe in die Nachrichtenwarteschlange (**158**) arrangiert wird.

3. System nach einem vorangehenden Anspruch, dadurch gekennzeichnet, daß die Abspieleinrichtung (**154**) als Antwort auf Empfangen einer Datenstromdarstellung der Vielzahl von Aufrufen von Prozeduren in einer Nachricht das Arrangieren der Datenstromdarstellung rückgängig macht und die Aufrufe von Prozeduren an die Objektklasseninstanz (**86**) ausgibt.

4. System nach einem vorangehenden Anspruch, ferner umfassend:
einen Objektkonfigurationsspeicher (**165**), der Informationen über Objekteigenschaften enthält, die Eigenschaften von in dem System ausführbaren Objektklassen repräsentieren, wobei die Informationen über Objekteigenschaften angeben, welche Objektklassen in einer Warteschlange angeordnete Aufrufe von Prozeduren unterstützen.

5. System nach Anspruch 4, ferner umfassend:
eine Einrichtung (**150**, **160**) zur Aufzeichnung von Aufrufen von Prozeduren, die betreibbar ist, um besagte Aufzeichnungseinrichtungen (**150**) für Objektinstanzen von Objektklassen bereitzustellen, die zum Unterstützen von in einer Warteschlange angeordneten Aufrufen von Prozeduren vorgesehen sind.

6. System nach einem vorangehenden Anspruch, ferner umfassend: eine Einrichtung (**154**, **180**) zur Widergabe von Aufrufen von Prozeduren, die betreibbar ist, um genannte Abspieleinrichtungen (**154**) für Objektinstanzen von Objektklassen bereitzustellen, die zum Unterstützen von in einer Warteschlange angeordneten Aufrufen von Prozeduren vorgesehen sind.

7. System nach einem vorangehenden Anspruch, dadurch gekennzeichnet, daß die Aufzeichnungseinrichtung bei Abschluß der Transaktion die Nachricht an die Programmierschnittstelle für Nachrichtenwarteschlangenbildungsanwendungen übergibt.

8. System nach Anspruch 7, dadurch gekennzeichnet, daß die Abspieleinrichtung die Aufrufe von Prozeduren an das Objekt in einer separaten Transaktion ausgibt.

9. System nach einem vorangehenden Anspruch, dadurch gekennzeichnet, daß die Aufzeichnungseinrichtung die Nachricht an die Programmierschnittstelle für Nachrichtenwarteschlangenbildungsanwendungen übergibt, nachdem der Client eine Referenz zu dem Objekt freigibt.

10. Verfahren zum asynchronen Remoting von Aufrufen von Prozeduren eines Client-Programms (**132**) auf einem Client-seitigen Computer an ein Objekt (**86**) auf einem Objekt-seitigen Computer über eine Nachrichtenwarteschlange (**158**), in Computer in einem verteilten Computernetzwerk (**86**, **92**), wobei das Verfahren umfaßt:
Bereitstellen einer Referenz zur Verwendung durch einen Client (**132**) auf einem Client-seitigen Computer zum asynchronen Aufrufen von Prozeduren eines Objekts (**86**);

als Antwort auf die Ausgabe eines Satzes von Aufrufen von Prozeduren für das Objekt in einer ersten Transaktion eines Clients: Empfangen von Aufrufen von Prozeduren des Clients an dem Objekt als direkte Aufrufe an Proxy-Schnittstellen, wodurch der Satz von Aufrufen von Prozeduren unterbrochen wird, und Aufzeichnen der Aufrufe von Prozeduren des Satzes in einer Nachricht;
Eingabe der Nachricht in eine mit dem Objekt verbundene Nachrichtenwarteschlange;
Einreihen der Nachricht zu einem späteren Zeitpunkt, wobei die Nachricht bei erfolgreichem Abschluß der ersten Transaktion in die Nachrichtenwarteschlange eingereiht wird; und
Extrahieren der Aufrufe von Prozeduren aus der Nachricht; und
Ausgeben der Aufrufe von Prozeduren an das Objekt auf dem Objekt-seitigen Computer zum späteren Zeitpunkt in einer zweiten Transaktion.

11. Computerprogramm mit computerlesbaren Codes, die geeignet sind, um alle Schritte von Anspruch 10 durchzuführen, wenn das Programm auf einem Computer läuft.

12. Computerprogramm nach Anspruch 11, ferner umfassend: eine Aufzeichnungskonstruktionseinrichtung, die eine Aufzeichnungseinrichtung als Antwort auf eine Anforderung einer Referenz zu einem Objekt (**86**) eines Clients (**132**) erzeugt.

13. Computerprogramm nach Anspruch 11 oder 12, dadurch gekennzeichnet, daß die Aufzeichnungseinrichtung Daten der Aufrufe von Prozeduren in einem Datenstrom einer Nachricht arrangiert und die Nachricht in eine mit dem Objekt verbundene Warteschlange eingibt.

14. Computerprogramm nach einem der Ansprüche 11 bis 13, dadurch gekennzeichnet, daß die Abspieleinrichtungsextraktion Rückgängigmachen des Arrangierens des Datenstroms in der Nachricht umfaßt.

15. Computerprogramm nach einem der Ansprüche 11 bis 14, verkörpert in einem computerlesbaren Medium.

Es folgen 7 Blatt Zeichnungen

Anhängende Zeichnungen

FIG. 1

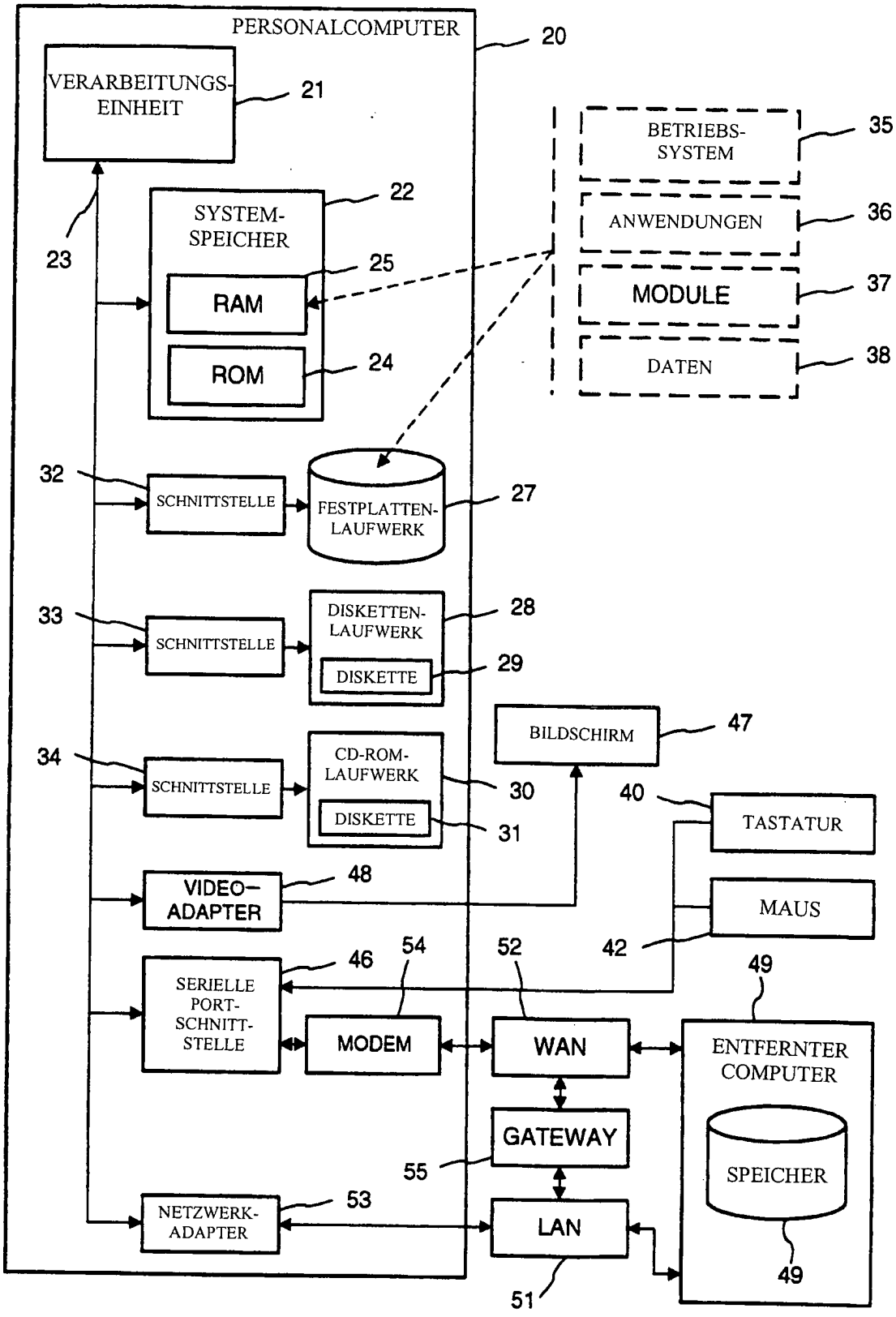


FIG. 2

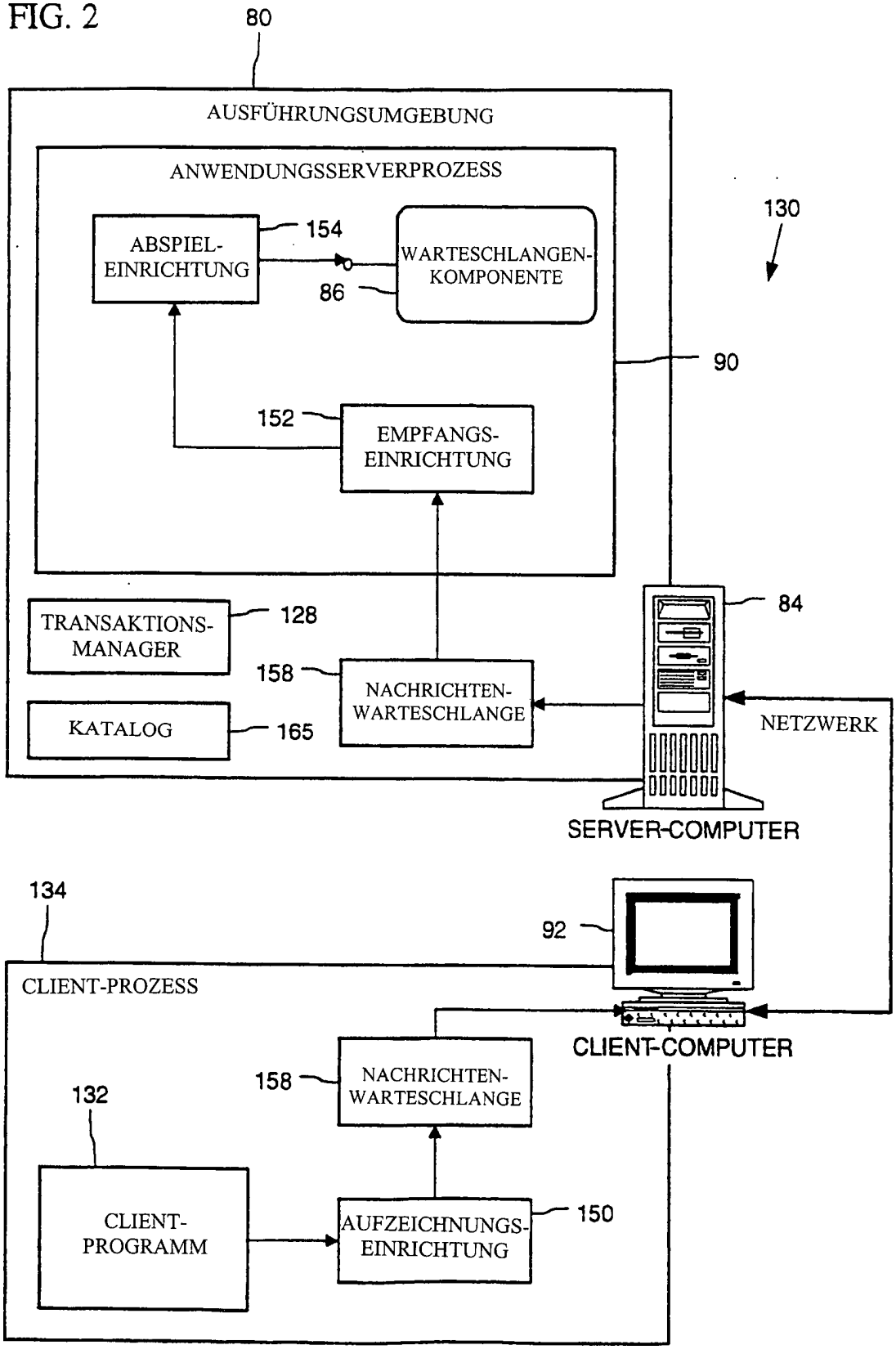

80
AUSFÜHRUNGSUMGEBUNG
ANWENDUNGSSERVERPROZESS
ABSPIEL-EINRICHTUNG
154
WARTESCHLANGEN-KOMPONENTE
86
130
90
152
EMPFANGS-EINRICHTUNG
TRANSAKTIONS-MANAGER
128
158
NACHRICHTEN-WARTESCHLANGE
KATALOG
165
84
NETZWERK
SERVER-COMPUTER
134
92
CLIENT-PROZESS
CLIENT-COMPUTER
158
NACHRICHTEN-WARTESCHLANGE
132
CLIENT-PROGRAMM
AUFZEICHNUNGS-EINRICHTUNG
150

FIG. 3

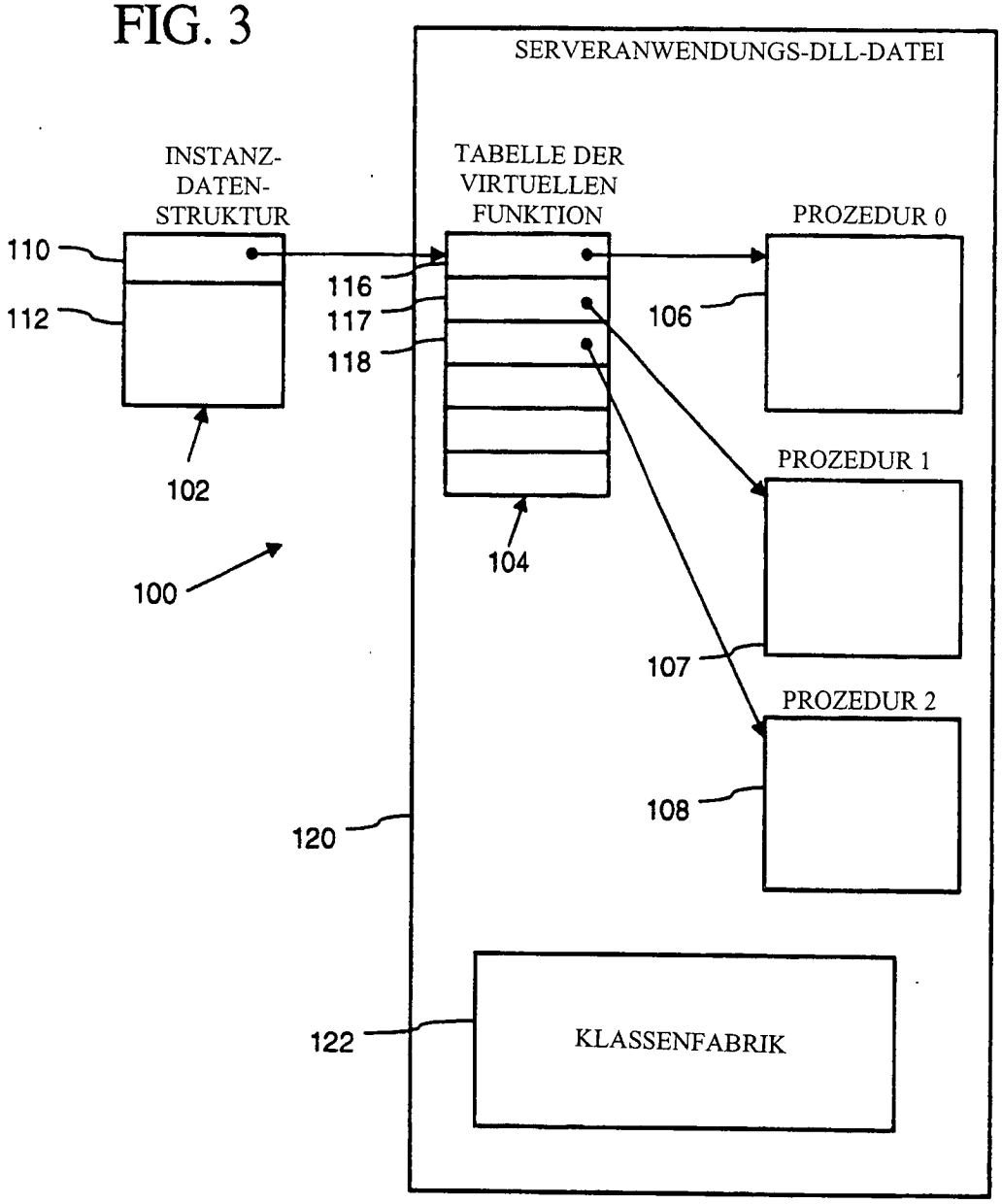

SERVERANWENDUNGS-DLL-DATEI
INSTANZ-DATEN-STRUKTUR
TABELLE DER VIRTUELLEN FUNKTION
PROZEDUR 0
PROZEDUR 1
PROZEDUR 2
KLASSENFABRIK
110
112
102
100
116
117
118
104
106
107
108
120
122

FIG. 4

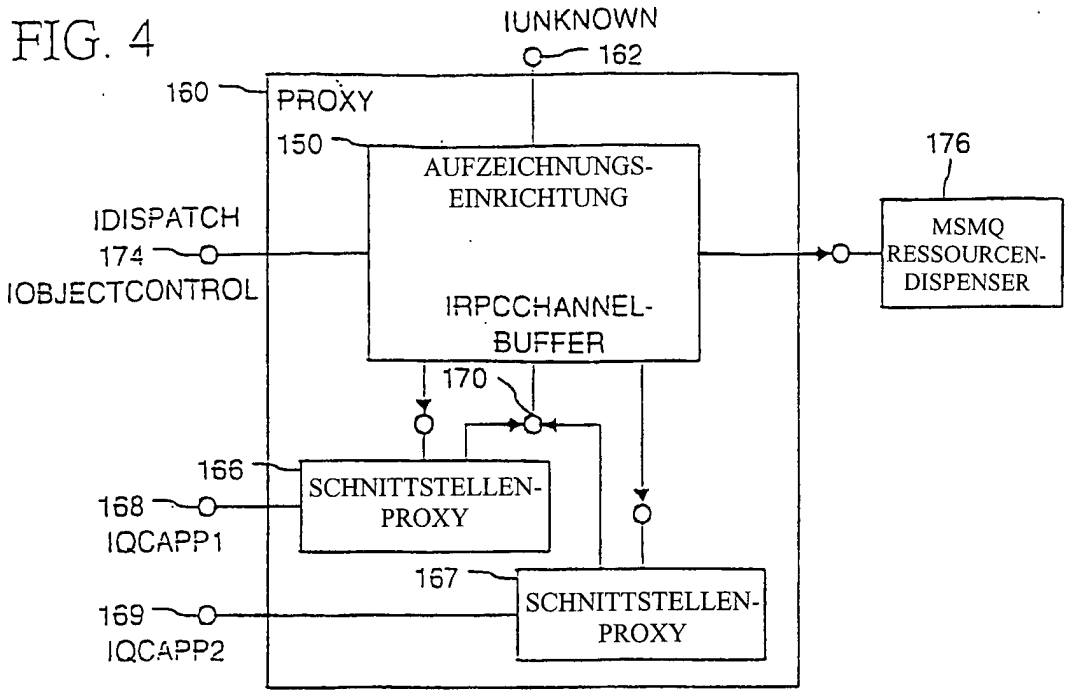

IUNKNOWN
162
160
PROXY
150
AUFZEICHNUNGS-
EINRICHTUNG
176
MSMQ
RESSOURCEN-
DISPENSER
IDISPATCH
174
IOBJECTCONTROL
IRPCCHANNEL-
BUFFER
170
166
SCHNITTSTELLEN-
PROXY
168
IQCAPP1
167
SCHNITTSTELLEN-
PROXY
169
IQCAPP2

FIG. 5

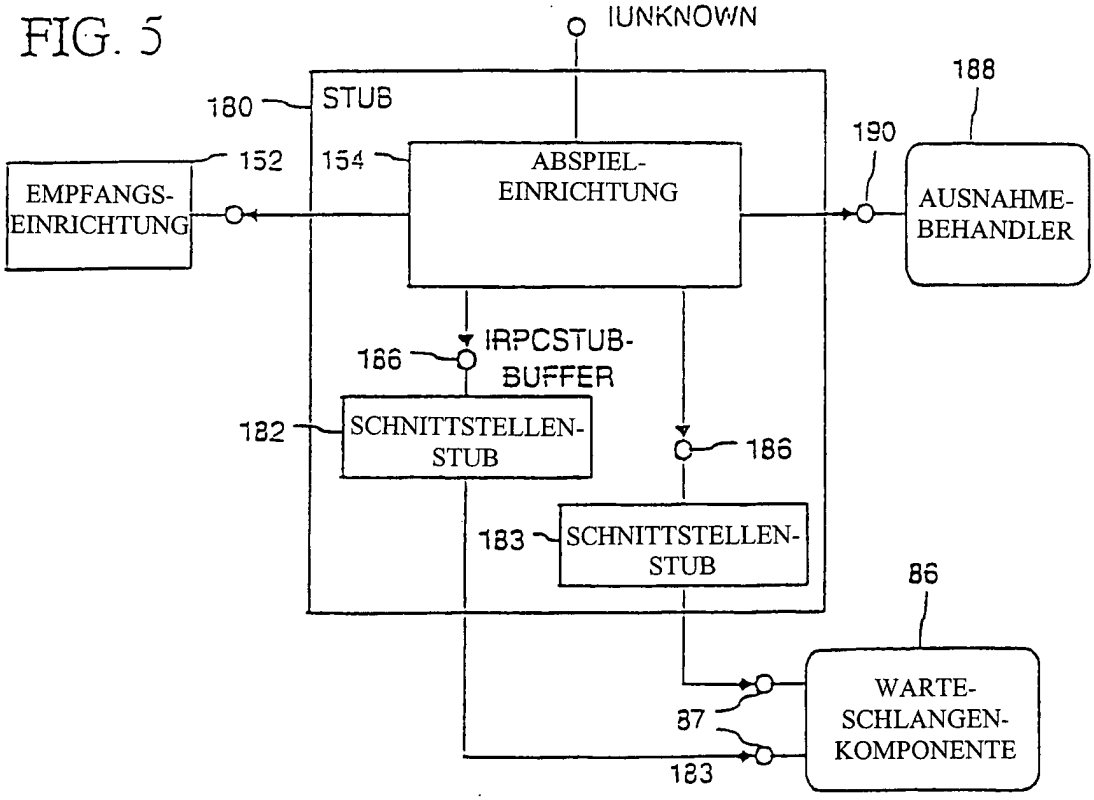

IUNKNOWN
188
180
STUB
190
152
154
ABSPIEL-
EINRICHTUNG
EMPFANGS-
EINRICHTUNG
AUSNAHME-
BEHANDLER
IRPCSTUB-
BUFFER
186
182
SCHNITTSTELLEN-
STUB
186
183
SCHNITTSTELLEN-
STUB
86
WARTE-
SCHLANGEN-
KOMPONENTE
87
183

FIG. 6

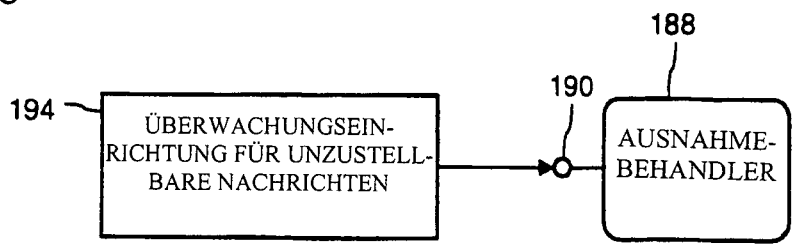


FIG. 7

```
CoGetObject(L"queue:/new:Ship", 0, IID_IShip, (void **)&pShip);

CoGetObject(L"queue:/new:812DF40E-BD88-11D0-8A6D-
00C04FC340EE", 0, IID_IShip, (void**)&pShip);

CoGetObject(L"queue:/new:{812DF40E-BD88-11D0-8A6D-
00C04FC340EE}", 0, IID_IShip, (void**)&pShip);

CoGetObject(L"new:Ship", 0, IID_IShip, (void **)&pShip);

CoCreateInstance( CLSID_Ship, NULL, CLSCTX_ALL,
IID_IShip, (void **)&pShip);
```

159

FIG. 8

```
interface IPlaybackControl : IUnknown
{
        HRESULT FinalClientRetry();
        HRESULT FinalServerRetry();
}
```

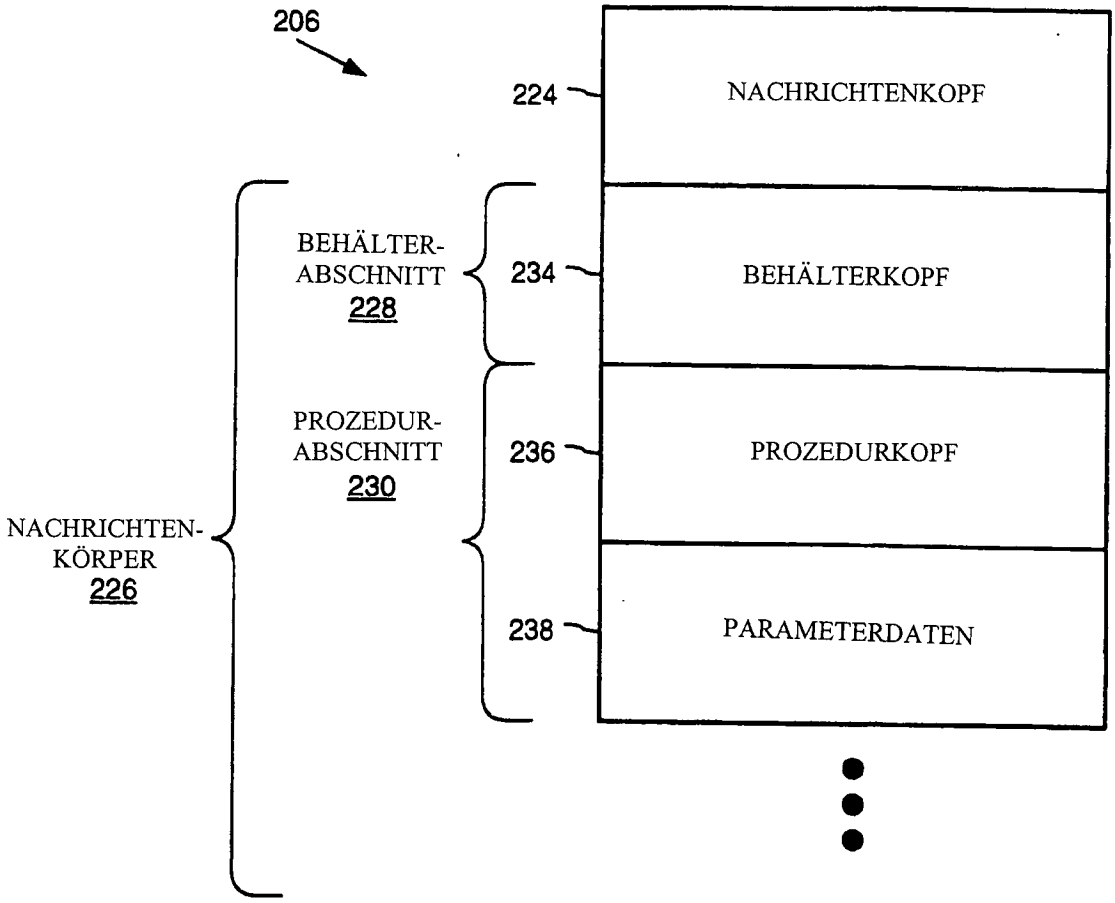
200

FIG. 9

206

```
typedef struct _CommonHeader
{       INT     TypeOfHeader;       // Type of header
        INT     SizeOfBlock; // size of the header +
                                                // associated data
} COMMON_HEADER;

typedef struct _ContainerHeader : COMMON_HEADER
{       //the fixed part of the container header
        GUID  guidSignature;
An IStream form of a moniker that can activate the server object.
} CONTAINER_HEADER;

typedef struct _ShortMethodHeader : COMMON_HEADER
{       INT     MethodId;
        RPCOLEDATAREP dataRepresentation;
        INT     RpcFlags;
        INT     cbVariableData;
        INT     UsingInterceptor;
        INT     Padding;
} SHORTMETHOD_HEADER;

typedef struct _MethodHeader : SHORTMETHOD_HEADER
{       IID iid; // interface to be invoked
} METHOD_HEADER;

typedef struct _SecurityHeader : COMMON_HEADER
{       INT cbVariableData;         // data length without padding
        INT Padding;          // pad to 8-byte boundary
} SECURITY_HEADER;

typedef struct _SecurityRefHeader : COMMON_HEADER
{       INT SecurityHeaderOffset;
        INT Padding; // pad to 8-byte boundary
} SECURITYREF_HEADER;
```

FIG. 10


206
224
NACHRICHTENKOPF
BEHÄLTER-
ABSCHNITT
228
234
BEHÄLTERKOPF
PROZEDUR-
ABSCHNITT
230
236
PROZEDURKOPF
NACHRICHTEN-
KÖRPER
226
238
PARAMETERDATEN