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

(10) **DE 698 22 904 T2** 2005.04.21

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 855 649 B1**
(21) Deutsches Aktenzeichen: **698 22 904.5**
(96) Europäisches Aktenzeichen: **98 300 234.6**
(96) Europäischer Anmeldetag: **14.01.1998**
(97) Erstveröffentlichung durch das EPA: **29.07.1998**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **07.04.2004**
(47) Veröffentlichungstag im Patentblatt: **21.04.2005**

(51) Int Cl.[7]: **G06F 9/46**

(30) Unionspriorität:
9701566 25.01.1997 GB

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
International Business Machines Corp., Armonk, N.Y., US

(74) Vertreter:
Teufel, F., Dipl.-Phys., Pat.-Anw., 70569 Stuttgart

(72) Erfinder:
Foster, Richard Daniel, Martin, Hampshire SP6 3LF, GB; Hutchison, Gordon Douglas, Chandlers Ford, Hampshire SO53 1JZ, GB

(54) Bezeichnung: **Vorrichtung und Verfahren zur automatischen Transaktionsverriegelung in einem objektorientierten Rechnersystem**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

Bereich der Erfindung

[0001] Die Erfindung betrifft den Bereich der Rechnersysteme, beispielsweise Client/Server-Rechnersysteme (die auch als "verteilte" Rechnersysteme bekannt sind), bei denen eine Rechnereinheit ("der Client") eine andere Rechnereinheit ("den Server") ersucht, einen Teil der Arbeit des Client durchzuführen.

Der Erfindung zugrunde liegender allgemeiner Stand der Technik

[0002] Die Client/Server-Datenverarbeitung hat in den letzten Jahren in der Welt der Informationstechnologie zunehmend an Bedeutung gewonnen. Diese Art der verteilten Datenverarbeitung ermöglicht einer Maschine, einen Teil ihrer Arbeit an eine andere Maschine zu delegieren, die beispielsweise zur Durchführung dieser Arbeit besser geeignet sein könnte. Der Server könnte zum Beispiel ein hochleistungsfähiger Rechner sein, der ein Datenbankprogramm ausführt, das die Speicherung einer enormen Datenmenge verwaltet, während der Client einfach ein als Tischrechner ausgeführter Personal Computer (PC) ist, der Informationen von der Datenbank zur Verwendung in einem seiner lokalen Programme anfordert.

[0003] Die Vorteile der Client/Server-Datenverarbeitung konnten durch die Verwendung einer bekannten Rechner-Programmiertechnik mit der Bezeichnung objektorientierte Programmierung (OOP) sogar noch weiter ausgebaut werden, die es möglich macht, dass sich der Client und der Server auf verschiedenen (heterogenen) "Plattformen" befinden. Eine Plattform ist eine Kombination aus Hardware/Software/Betriebssystem/Kommunikationsprotokoll einer ganz bestimmten Art, die eine Maschine zur Verrichtung ihrer Arbeit verwendet. OOP ermöglicht dem Client-Anwendungsprogramm und dem Server-Anwendungsprogramm, auf ihren eigenen Plattformen zu arbeiten, ohne sich mit der Frage befassen zu müssen, wie die Arbeitsanforderungen der Client-Anwendung übertragen und von der Server-Anwendung entgegen genommen werden. Ebenso braucht sich die Server-Anwendung nicht mit der Frage zu befassen, wie das OOP-System die Verarbeitungsergebnisse der Server-Anwendung empfängt, umsetzt und zurück an die anfordernde Client-Anwendung sendet.

[0004] Einzelheiten darüber, wie OOP-Techniken in heterogene Client/Server-Systeme eingebunden wurden, sind in der US-Patentschrift Nr. 5 440 744 und der Europäischen Patentanmeldung 0 677 943 A2 erklärt. Ein Beispiel für die grundlegende Architektur wird jedoch nachstehend gegeben, um die Umgebung der Erfindung in ihrem Zusammenhang zu verstehen.

[0005] Wie in **Fig.** 1 gezeigt ist, hat der Client-Rechner **10** (der beispielsweise ein Personal Computer mit einem darauf installierten Betriebssystem IBM OS/2 sein könnte) ein auf seinem Betriebssystem laufendes Anwendungsprogramm **40** ("IBM" und "OS/2" sind Warenzeichen der International Business Machines Corporation). Das Anwendungsprogramm **40** muss in regelmäßigen Abständen Arbeiten auf dem Server-Rechner **20** verrichten und/oder Daten vom Server **20** zurückschicken lassen, damit sie anschließend vom Anwendungsprogramm **40** verwendet werden können. Der Server-Rechner **20** kann beispielsweise ein leistungsfähiger Großrechner sein, der mit dem Betriebssystem MVS von IBM ("MVS" ist ebenfalls ein Warenzeichen der IBM Corp.) läuft. Für die Zwecke der vorliegenden Erfindung es ist nicht von Bedeutung, ob die Anforderungen für vom Server zu erbringende Übertragungsdienste durch Interaktion des Benutzers mit dem ersten Anwendungsprogramm **40** ausgelöst werden oder ob das Anwendungsprogramm **40** unabhängig von einem Dialogverkehr mit dem Benutzer arbeitet und die Anforderungen automatisch während der Ausführung des Programms stellt.

[0006] Wenn der Client-Rechner **10** eine Anforderung für die Dienste des Server-Rechners **20** stellen möchte, informiert das erste Anwendungsprogramm **40** das erste Logikmittel **50** über den benötigten Dienst. Dies kann beispielsweise geschehen, indem das erste Anwendungsprogramm dem ersten Logikmittel den Namen einer fernen Prozedur zusammen mit einer Liste von Eingabe- und Ausgabeparametern schickt. Das erste Logikmittel **50** führt dann die Aufgabe des Aufbaus der notwendigen Verbindungen mit dem zweiten Rechner **20** durch, wobei es auf Angaben der verfügbaren Übertragungsdienste Bezug nimmt, die in der Speichereinheit **60** gespeichert sind. Alle möglichen Dienste sind als ein geschlossenes Rahmenwerk (framework) von Objektklassen **70** definiert, wobei diese Klassen von einer einzigen Objektklasse abgeleitet werden. Indem die Dienste auf diese Weise definiert werden, ergeben sich viele Vorteile in Bezug auf Leistungsfähigkeit und Wiederverwendbarkeit.

[0007] Um die notwendige Verbindung mit dem Server **20** herzustellen, stellt das erste Logikmittel **50** fest, welche Objektklasse in dem Rahmenwerk verwendet werden muss, und erzeugt dann eine Instanz dieses Objekts am Server, und es wird eine Nachricht an dieses Objekt gesendet, um das Objekt zu veranlassen, eine seiner Methoden aufzurufen. Dadurch wird die Verbindung mit dem Server-Rechner **20** über das Verbindungsmittel **80** aufgebaut und anschließend eine Anforderung an das zweite Logikmittel **90** gesendet.

[0008] Das zweite Logikmittel **90** leitet dann die Anforderung an das zweite Anwendungsprogramm **100**

(nachstehend entweder als die Dienst- oder Server-Anwendung bezeichnet) weiter, das auf dem Server-Rechner **20** ausgeführt wird, so dass die Dienst-Anwendung **100** die jeweilige von dieser Anforderung benötigte Aufgabe, wie zum Beispiel die Ausführung einer Datenabrufprozedur, durchführen kann. Sobald diese Aufgabe durchgeführt wurde, muss die Dienst-Anwendung gegebenenfalls Ergebnisse an den ersten Rechner **10** zurücksenden. Die Dienst-Anwendung **100** tritt während der Durchführung der angeforderten Aufgaben und wenn Ergebnisse an den ersten Rechner **10** zurückgesendet werden müssen mit dem zweiten Logikmittel **90** in Dialogverkehr. Das zweite Logikmittel **90** legt Instanzen von Objekten an und ruft nach Bedarf der Dienst-Anwendung **100** entsprechende Methoden dieser Objekte auf, wobei die Objekt-Instanzen von dem geschlossenen Rahmenwerk von Objektklassen erzeugt werden, das in der Speichereinheit **110** gespeichert ist.

[0009] Bei Verwendung des vorstehenden Verfahrens wird das Client-Anwendungsprogramm **40** gegenüber der Übertragungsarchitektur nicht offen gelegt. Außerdem wird die Dienst-Anwendung **100** über den Standardmechanismus für ihre Umgebung aufgerufen; sie weiß nicht, dass sie rechnerfern aufgerufen wird.

[0010] Die Object Management Group (OMG) ist ein internationales Konsortium von Organisationen, die an verschiedenen Aspekten der Client/Server-Datenverarbeitung auf heterogenen Plattformen mit verteilten Objekten beteiligt sind, wie in **Fig.** 1 gezeigt ist. Die OMG hat veröffentlichte Standards festgelegt, nach denen Client-Rechner (z. B. der Client-Rechner **10**) mit Server-Maschinen (z. B. der Server-Maschine **20**) (in OOP-Form) kommunizieren. Als Teil dieser Standards wurde ein Objektvermittlungsvermittler (Object Request Broker (ORB)) festgelegt, der die objektorientierte Brücke zwischen der Client-Maschine und der Server-Maschine bereitstellt. Der ORB entkoppelt die Client- und die Server-Anwendung von den Einzelheiten der objektorientierten Ausführung, wobei er mindestens einen Teil der Arbeit des ersten Logikmittels **50** und des zweiten Logikmittels **90** sowie des Verbindungsmittels **80** durchführt. Der ORB wickelt auch den gesamten Dialogverkehr zwischen verschiedenen Server-Objekten der Dienst-Anwendung **100** ab.

[0011] Der folgende Leitfaden kann zur Vertiefung der Kenntnisse in diesem Bereich herangezogen werden: "The Essential Distributed Objects Survival Guide" von R. Orfali, D. Harkey, J. Edwards, 1996, John Wiley & Sons, Inc., New York, XP002081892.

[0012] Mittels Rechner realisierte Transaktionsverarbeitungssysteme werden in vielen Branchen für kritische Geschäftsaufgaben eingesetzt. Eine Transaktion gibt eine einzelne Arbeitseinheit an, die entweder vollständig abgearbeitet oder vollständig abgebrochen werden muss, ohne dass ein Vorgang durchgeführt wird. Im Falle eines Bankautomaten zum Beispiel, von dem ein Kunde Geld abheben möchte, müssen die Vorgänge der Geldausgabe, der Verringerung des in dem Automaten verfügbaren Geldvolumens und der Verringerung des Kundensaldos entweder alle stattfinden, oder keiner von ihnen darf stattfinden. Wenn einer der untergeordneten Vorgänge fehlschlagen würde, würde dies zu Unstimmigkeiten zwischen den Datensätzen und den tatsächlichen Geschehnissen führen.

[0013] Die verteilte Transaktionsverarbeitung schließt eine Transaktion ein, die Ressourcen an mehr als einem physischen oder logischen Standort betrifft. In dem obigen Beispiel betrifft eine Transaktion Ressourcen, die am lokalen Bankautomaten verwaltet werden, sowie Banksalden, die vom Hauptrechner einer Bank verwaltet werden. Solche Transaktionen schließen einen bestimmten Client-Rechner (z. B. **10**) ein, der mit einem bestimmten Server-Rechner (z. B. **20**) über eine Reihe von Client-Anforderungen kommuniziert, die vom Server verarbeitet werden.

[0014] In typischen Client/Server-Systemen tragen jeweils das Client-System und das Server-System zu der Gesamtverarbeitung solcher Transaktionen bei. Außerdem können verschiedene Clients gleichzeitig versuchen, denselben Server zu benutzen, um sich mit getrennten Transaktionen zu beschäftigen. Zum Beispiel können viele verschiedene Bankautomaten (Client-Systeme) versuchen, gleichzeitig Transaktionen zu beginnen, um auf Daten von einem häufig genutzten Datenbankprogramm zuzugreifen, das auf dem großen Zentralserver der Bank ausgeführt wird. Überdies gibt es noch viele verschiedene serverinterne Anforderungen (z. B. Anforderungen, die von einem Server-Objekt an ein anderes Server-Objekt geleitet werden), die Teil von getrennten (gleichzeitigen) Transaktionen sein können. In diesen Situationen muss der Server diese gleichzeitigen Transaktionen isolieren können, so dass sie einander nicht stören. Das heißt, bis eine Transaktion beendet ist (entweder alle Teile werden festgeschrieben, oder alle Teile werden abgebrochen), müssen andere Transaktionen, die versuchen, auf dieselben Server-Objekte zuzugreifen, in den Wartezustand versetzt werden. Die Server-Objekte, die an einer Transaktion beteiligt sind, müssen gesperrt werden, während die Transaktion im Wartezustand ist. Durch die Sperre werden gleichzeitige Zugriffe auf die Server-Objekte außerhalb der Transaktion verhindert, die die aktuelle Transaktion beeinflussen würden.

[0015] Wenn beispielsweise ein Ehemann versucht, am Bankautomaten einer Bank am einen Ende der Stadt 2000 US-Dollar von einem Girokonto der Fami-

lie auf ein höherverzinsliches Sparkonto der Familie zu überweisen, und seine Frau versucht, dieselbe Funktion an einem anderen Bankautomaten (der einer anderen Bank gehört) am anderen Ende der Stadt auszuführen, muss der Server diese Situation wirksam handhaben können, so dass die beiden gleichzeitigen Transaktionen zu keinem Problem für die Bank führen, der der Datenbank-Server gehört.

[0016] Dieses Problem wird in dem objektorientierten Programmierkontext üblicherweise in der Art und Weise gelöst, dass das Server-Datenbankprogramm (Server- oder Dienst-Anwendungsprogramm **100**) so geschrieben wird, dass es nicht nur die wesentlichen Funktionen des Programms ausführt, sondern auch Transaktionen bei gleichzeitigen Zugriffen sperrt. Das heißt, die Server-Anwendung **100** würde so geschrieben werden, dass sie den Zugriff auf die Kontendaten der Familie sperrt, die in der Datenbank gespeichert sind, sobald ein erster Client (z. B. der Bankautomat des Ehemanns) den Zugriff anfordert. Dann würde die Transaktion des Ehemanns trotz der Tatsache, dass die Transaktion der Ehefrau gleichzeitig angefordert wurde, isoliert fortgesetzt werden. Der Client-Bankautomat der Ehefrau würde keinen Zugriff auf die Daten gewährt bekommen, weil im Client-Bankautomat des Ehemanns bereits eine Sperre auf das Objekt gesetzt worden wäre, das diese Daten kapselt.

[0017] Bindet man die Zuständigkeit für die Gleichzeitigkeitssteuerung in die Server-Anwendung (z. B. das Server-Anwendungsprogramm **100**) ein, setzt dies voraus, dass der Programmierer der Server-Anwendung die komplexen Sperrschemata in sein Programm aufnimmt. Außerdem muss der Programmierer eingehende Kenntnisse der Transaktionstheorie haben, damit er den Transaktionskontext in die Aspekte der Gleichzeitigkeitssteuerung des Programms integrieren kann. Viele Anwendungsprogrammierer haben keine Kenntnisse dieser Gleichzeitigkeitssteuerung und/oder Transaktionstheorie. Und selbst wenn sie über diese Kenntnisse verfügen, erschwert die Einbindung dieser Aspekte in die Server-Anwendung bei der Programmierung der wesentlichen Funktionen der Server-Anwendung ihre Programmieraufgabe zusätzlich.

Offenlegung der Erfindung

[0018] Gemäß einem Aspekt stellt die vorliegende Erfindung ein Rechnersystem nach Anspruch 1 bereit.

[0019] Da die Gleichzeitigkeitssteuerung in dem Mittel zum Empfang und zur Zuteilung von Anforderungen stattfindet, brauchen die eigentlichen Mittel zur Verarbeitung von Anforderungen (die Server-Objektinstanzen) keine Mechanismen zur Gleichzeitigkeitssteuerung zu enthalten. Dadurch wird die Entwicklung von Server-Anwendungen stark vereinfacht. Der Entwickler der Server-Anwendung braucht keine Einzelheiten der Gleichzeitigkeitssteuerung oder Transaktionstheorie zu kennen und kann sich stattdessen auf die Entwicklung der wesentlichen Punkte der Server-Anwendung konzentrieren.

[0020] In einer Ausführungsform ist das Mittel zum Empfang und zur Zuteilung ein Objektanforderungsvermittler (Object Request Broker). In einer anderen Ausführungsform ist das Mittel zum Empfang und zur Zuteilung ein Übersetzer einer objektorientierten Programmiersprache.

[0021] Vorzugsweise isoliert das Mittel zur Durchführung der Gleichzeitigkeitssteuerung die Auswirkungen einer Gruppe von zusammengehörenden Arbeitsanforderungen, die eine Transaktion bilden.

[0022] Gemäß einem weiteren Aspekt stellt die vorliegende Erfindung einen Prozess nach Anspruch 5 bereit.

[0023] Vorzugsweise beinhaltet der Auswerteschritt (b) die folgenden Teilschritte:
(b1) Prüfen, ob die erste Arbeitsanforderung den Zugriff auf fundamentale Zustandsdaten einer grundlegenden Art des Objekts anfordert, die direkt auf das Speichermedium des Rechnersystems abgebildet wird; und (b2) Prüfen, ob die erste Arbeitsanforderung Teil einer aktuellen Transaktion ist.

[0024] Vorzugsweise beinhaltet der Auswerteschritt (b) des Weiteren die folgenden Teilschritte:
(b3) Prüfen, ob das zu der Arbeitsanforderung gehörende Objekt bereits von einer vorhergehenden Anforderung gesperrt wurde, die zu derselben Transaktion wie die Arbeitsanforderung gehört; und (b4) Prüfen, ob das zu der Arbeitsanforderung gehörende Objekt bereits von einer einen Konflikt auslösenden Sperre gesperrt wurde, die zu einer anderen Transaktion als der Transaktion der Arbeitsanforderung gehört.

[0025] Vorzugsweise beinhaltet der Sperrschritt (c) die folgenden Teilschritte:
(c1) Erzeugen eines Stapelspeichers mit Objektinstanzen, während die Arbeitsanforderung von dem angeforderten Objekt an mindestens ein anderes Objekt übertragen wird, das das angeforderte Objekt aufruft; (c2) Prüfen des Stapelspeichers, um das Objekt in der obersten Behälterebene und das Objekt in der untersten Behälterebene festzustellen; und (c3) Setzen einer Absichtssperre auf das Objekt in der obersten Behälterebene. Der Sperrschritt beinhaltet vorzugsweise des Weiteren die Teilschritte (c4) des Setzens von Absichtssperren auf alle restlichen Objekte in dem Stapelspeicher von der obersten Behälterebene bis hinab zur untersten Behälterebene, diese jedoch nicht eingeschlossen; und (c5) des Set-

zens einer Lese- oder Schreibsperre auf das Objekt in der untersten Behälterebene.

[0026] In der ersten Ausführungsform wird in dem Teilschritt (c2) ein Attributkennzeichen einer Behälterobjekt-Klassendefinition geprüft, um die Behälterebene festzustellen.

[0027] Vorzugsweise ist das Rechnersystem ein Server, und die Arbeitsanforderungen werden von mindestens einem Client-Rechnersystem auf einer heterogenen Plattform an dem Server empfangen.

[0028] In der ersten Ausführungsform werden die Schritte (a), (b) und (c) von einem Objektanforderungsvermittler durchgeführt. In der zweiten Ausführungsform werden die Schritte (a), (b) und (c) von einem Übersetzer einer objektorientierten Programmiersprache durchgeführt.

[0029] Gemäß einem weiteren Aspekt stellt die Erfindung ein Rechnerprogrammprodukt bereit, das auf einem rechnerlesbaren Speichermedium gespeichert wird, um die Funktionen der vorstehend erörterten Prozesse auszuführen.

Kurze Beschreibung der Zeichnungen

[0030] **Fig.** 1 ist ein Blockschaltbild einer bekannten heterogenen Client/Server-Architektur, die die Objekttechnologie verwendet, in deren Kontext bevorzugte Ausführungsformen der vorliegenden Erfindung angewendet werden können;

[0031] **Fig.** 2 ist ein Ablaufdiagramm, das die von einem Objektanforderungsvermittler durchgeführten Schritte bei der Disposition von Client-Anforderungen und der Handhabung von Fragen bezüglich der Probleme der Gleichzeitigkeit von Transaktionen gemäß einer bevorzugten Ausführungsform der vorliegenden Erfindung zeigt;

[0032] **Fig.** 3 ist ein Beispiel eines Stapelspeichers für Paare von Objektinstanzen/Methoden, wobei ein solcher Stapelspeicher verwendet wird, um die automatische Sperrung gemäß bevorzugten Ausführungsformen der vorliegenden Erfindung durchzuführen; und

[0033] **Fig.** 4 ist ein Diagramm, das eine ungewöhnliche "Aufrufbeziehung" zwischen Objekten zeigt, die eine Änderung an dem grundlegenden System der vorliegenden Erfindung erforderlich macht.

Ausführliche Beschreibung der bevorzugten Ausführungsformen

[0034] Eine bevorzugte Ausführungsform der vorliegenden Erfindung enthält einen ORB, der Anforderungen vom Client an den Server (sowie serverinterne Anforderungen zwischen zwei Server-Objekten) vermittelt, wobei sich der Client und der Server auf getrennten Maschinen befinden. Jedoch können sich der Client und der Server auch auf derselben Maschine befinden, wobei ein Objekt auf der Maschine einen Dienst eines anderen Objekts auf derselben Maschine anfordert und diese Anforderung zusammen mit allen anderen Anforderungen zwischen Objekten den ORB der Maschine durchläuft. Der grundlegende Fluss der Gleichzeitigkeitssteuerung, die in dem ORB gemäß der bevorzugten Ausführungsform stattfindet, wird nun mit Bezug auf **Fig.** 2 beschrieben. Diese Art der Steuerung befasst sich mit dem Problem gleichzeitiger Anforderungen für den Zugriff auf dasselbe Server-Objekt. Wenn beispielsweise eine Anforderung, die zu einer Transaktion gehört, versucht, den Inhalt der gekapselten Daten eines Server-Objekts zu prüfen, sollte es einer späteren Anforderung, die zu einer anderen Transaktion gehört, nicht gestattet werden, diese Daten zu ändern, während die erste Anforderung die Daten noch prüft. Auch wenn eine aktuelle Anforderung, die zu einer Transaktion gehört, versucht, den Inhalt der gekapselten Daten eines Server-Objekts zu ändern, sollte es einer späteren Anforderung, die zu einer anderen Transaktion gehört, nicht gestattet werden, den Inhalt dieser Daten zu prüfen, bis die aktuelle Anforderung mit der Änderung der Daten fertig ist.

[0035] In **Fig.** 1 ist der ORB des Servers **20** in dem Verbindungsmittel **80** und dem zweiten Logikmittel **90** enthalten. Dieser Server-ORB vermittelt sämtliche Übertragungen zwischen Objekten, die mit dem Server **20** zu tun haben. Zu diesen Übertragungen zwischen Objekten gehören nicht nur Übertragungen zwischen einem Objekt des Servers **20** und einem Objekt des Client **10**, sondern auch Übertragungen zwischen zwei Objekten des Servers **20**. Immer, wenn der ORB des Servers eine solche Interobjektanforderung empfängt, führt der Server seine üblichen Funktionen aus, wie zum Beispiel die Auswertung der Anforderung, um festzustellen, auf welches Server-Objekt (Zielobjekt) diese bestimmte Anforderung den Zugriff anfordert (Schritt **201** von **Fig.** 2). Wenn zum Beispiel ein Bankkunde, der einen Client-Bankautomaten nutzt, die Anforderung stellt, Geld von seinem Girokonto abzuheben, wäre das Server-Objekt, auf das er den Zugriff anfordert, ein Server-Objekt, das Daten in Bezug auf sein bestimmtes Konto enthält (Kontonummer, Saldo, Konditionen usw.). Die angeforderte Abhebungsoperation ist als eine "Methode" des Girokontos des Server-Objekts bekannt. Das Server-Objekt antwortet, indem es eine Methode auf seinen gekapselten Daten in Bezug auf das Girokonto aufruft.

[0036] Gemäß einer bevorzugten Ausführungsform der vorliegenden Erfindung prüft (Schritt **202**) der ORB dann die Methode, die von diesem Zielobjekt angefordert wird, um festzustellen, ob die Anforde-

rung den Zugriff auf die Zustandsdaten des Zielobjekts anfordert, die direkt auf das Speichermedium des Server-Rechnersystems abgebildet werden. Das heißt, versucht die Anforderung, die gekapselten Daten eines Server-Objekts zu ändern (schreiben) oder zu prüfen (lesen), die in dem Speichermedium des Server-Rechnersystems gespeichert sind (d. h., die Zustandsdaten werden auf ihrer niedrigsten Abstraktionsebene ausgedrückt, auf der die Zustandselemente nicht von weiteren Anwendungsobjekten umgesetzt, sondern von grundlegenden Arten dargestellt werden). Wenn ja, schaltet der Ablauf zum Schritt **203**, der nachstehend beschrieben wird. Wenn nicht, wird die in der Anforderung enthaltene Methode zur Ausführung im Server durch das Ziel-Serverobjekt gesendet (oder "zugeteilt") (Schritt **211**). Das heißt, wenn die aktuelle Client-Anforderung nicht versucht, auf den Zustand des Ziel-Serverobjekts zuzugreifen (ihn zu prüfen oder zu ändern), gibt es kein Problem mit der Gleichzeitigkeit in Bezug auf andere Anforderungen, so dass die aktuelle Anforderung einfach direkt an das Ziel-Serverobjekt zur Ausführung gesendet werden kann.

[0037] Wenn eine Anforderung den Schritt **203** erreicht, geht es in der Anforderung tatsächlich darum, gekapselte Daten eines Server-Objekts zu schreiben oder die gekapselten Daten eines Server-Objekts, die in dem Speichermedium gespeichert sind, zu lesen. Im Schritt **203** prüft der ORB, ob die aktuelle Anforderung Teil einer Transaktion ist (indem er beispielsweise den Dienst-Kontext der Anforderung prüft). Das heißt, der ORB stellt fest, ob die Anforderung Teil einer Folge von zusammengehörenden Anforderungen ist, die alle entweder erfolgreich durchgeführt werden oder aber erfolglos verlaufen müssen, damit die ganze Transaktion erfolgreich ist (festgeschrieben wird) oder fehlschlägt (abgebrochen wird). Wenn nicht, ist es eine einmalige oder isolierte Anfrage, und alles, was getan werden muss, ist zu prüfen (Schritt **204**), ob der Zugriff auf das angeforderte Server-Objekt bereits mit einer einen Konflikt auslösenden Sperre gesperrt ist. Wenn ja, muss die einmalige Anforderung warten, bis das Objekt freigegeben wird (z. B. bis eine Transaktion mit zur Zeit noch ungewissem Ausgang entweder festgeschrieben oder abgebrochen wird), und dann kann diese Anforderung dem Ziel-Serverobjekt zur Ausführung zugeteilt werden (Schritt **211**).

[0038] Wenn die Anforderung im Schritt **203** vom ORB als Teil einer Transaktion festgestellt wird, stellt der ORB anhand einer Prüfung fest (Schritt **206**), ob der Zugriff auf das angeforderte Server-Objekt von einer vorhergehenden Anforderung, die zu derselben Transaktion gehört, in der erforderlichen Betriebsart bereits gesperrt worden ist. Das heißt, der ORB prüft, ob eine vorhergehende, zu einer Transaktion gehörende Anforderung dieses Objekt bereits vor gleichzeitigen, außerhalb der Transaktion gestellten Anforderungen ausreichend geschützt hat, und folglich ist es nicht notwendig, dass diese Anforderung weitere Sperren aktiviert. Wenn beispielsweise eine vorhergehende, zu einer Transaktion gehörende Anforderung bereits eine Schreibsperre auf das Objekt erhalten hat und die aktuelle Anforderung um das Lesen der Daten des Objekts nachsucht, ist das Objekt bereits ausreichend geschützt. Folglich wird die Anforderung einfach an das Ziel-Serverobjekt zur Ausführung gesendet (Schritt **211**). Das heißt, obgleich möglicherweise zwei Anforderungen versuchen, auf die gekapselten Daten desselben Objekts zuzugreifen, wird dies in diesen Situationen nicht als Problem betrachtet, da beide Anforderungen zu derselben Transaktion gehören und daher zusammenarbeiten dürfen (was auch von ihnen erwartet wird).

[0039] Wenn im Schritt **206** jedoch die NEIN-Verzweigung vorgenommen wird, führt der ORB eine Prüfung durch (Schritt **207**), um festzustellen, ob die Anforderung mit einer Sperre in Konflikt steht, welche einer vorhergehenden Anforderung, die zu einer anderen Transaktion gehört, bereits gewährt wurde. Beispielsweise hat die vorhergehende Anforderung eine Schreibsperre auf dieses Objekt erhalten, und die aktuelle Anforderung versucht, die gekapselten Daten desselben Objekts zu ändern. Dies ist als eine Situation mit einer einen Konflikt auslösenden Sperre bekannt. Dies ist beispielsweise die Situation, in der ein Ehemann bereits Zugriff auf das Girokonto der Familie erhalten hat und die Ehefrau anschließend versucht, Zugriff von einem anderen Bankautomaten am anderen Ende der Stadt zu erhalten, während die Transaktion des Ehemanns immer noch in Ausführung befindlich ist. Wenn es eine einen Konflikt auslösende Sperre gibt, wird die aktuelle Anforderung in den Wartezustand versetzt, bis die aktuelle Transaktion abgeschlossen ist. In dem Beispiel würde die Anforderung der Ehefrau so lange zurückgestellt werden, bis die Transaktion des Ehemanns abgeschlossen ist. Dann würde ihre Anforderung eine Sperre auf das Server-Objekterhalten (Schritt **208**) und anschließend an das Ziel-Serverobjekt zur Ausführung gesendet werden (Schritt **211**), und ihre Anforderung für die Abhebung eines Geldbetrags würde am Saldo eines Girokontos durchgeführt werden, der durch die abgeschlossene Transaktion des Ehemanns verringert worden ist.

[0040] Wenn es im Schritt **207** keine Sperren von einer anderen Transaktion gibt, die einen Konflikt auslösen, erhält der ORB eine Sperre auf das angeforderte Server-Objekt (im Schritt **208**). Die Einzelheiten darüber, wie der ORB diese Sperre erhält, werden nachstehend erklärt. Sobald der Zugriff auf das Objekt gesperrt ist, sendet der ORB die Anforderung an das Zielobjekt (Schritt **211**) zur Ausführung (er teilt sie ihm zu). Das Objekt ist nun geschützt, so dass die Transaktion vonstatten gehen und dabei unbesorgt sein kann, dass eine Anforderung von einer anderen

Transaktion gleichzeitig auf das Objekt zugreift. Nachdem die aktuelle Transaktion abgeschlossen ist, hebt der ORB die Sperre auf.

[0041] Bevor die Einzelheiten des Sperrschritts **208** beschrieben werden, wird kurz das bekannte Konzept der Objektverknüpfung erörtert, damit sich die nachfolgende Erläuterung des Sperrschemas im Zusammenhang verstehen lässt.

[0042] Jede Anforderung zwischen Objekten für den Zugriff (entweder des Prüf- oder des Änderungstyps) auf ein Server-Objekt versucht, Zugriff auf den Zustand des Objekts zu erhalten, der in dem Server-Objekt gekapselt ist. Wie in der objektorientierten Programmierung üblich, kann ein Objekt andere Objekte haben, die in ihm "enthalten" sind. Beispielsweise könnte ein Objekt, das eine Gehaltstabelle für eine bestimmte Abteilung darstellt, getrennte ("darin enthaltene") Objekte haben, die sich auf das Gehalt einer jeden Person beziehen und in diesem größeren ("Behälter"-)Objekt enthalten sind. Statt dass sich darin gekapselte Daten befinden, besteht der Zustand des Behälterobjekts aus Zeigern auf enthaltene Objekte. Wenn auf solch einen Zeiger zugegriffen wird, wird die Steuerung dem Objekt übertragen, auf das "gezeigt" wird. Sobald die Steuerung dieses Objekt, auf das "gezeigt" wird, erreicht, kann sein Zustand einen Zeiger auf noch ein weiteres Objekt, auf das "gezeigt" wird, haben. Auf diese Weise "springt" die Steuerung nacheinander zu einer Vielzahl von Objekten.

[0043] Zum selben Zeitpunkt, zu dem das Behälterobjekt erzeugt wird, wird auch das darin enthaltene Objekt erzeugt. Das heißt, wenn das System Platz für das Behälterobjekt schafft, muss es im Speicher auch Platz für das enthaltene Objekt schaffen. Wenn das Behälterobjekt zerstört wird, wird der ihm zugewiesene Platz zusammen mit dem Platz, der dem darin enthaltenen Objekt zugewiesen ist, entsprechend freigegeben. Die enthaltenen Objekte und die Behälterobjekte "leben und sterben" zusammen. Diese Art der Objektverknüpfung wird verwendet, wenn das Objekt, auf das "gezeigt" wird, nur so lange vorhanden sein soll, wie das "Zeiger"-Objekt vorhanden ist.

[0044] Im Gegensatz zu dieser Behälter/enthalten-Beziehung kann ein Objekt einfach auf ein anderes Objekt verweisen, da immer noch ein Zeiger auf dieses andere Objekt als Teil des Zustands des Objekts enthalten ist. Das Objekt, auf das "verwiesen" wird, ist unabhängig von dem ("Zeiger"-)Objekt, das den Verweis vornimmt, vorhanden und "lebt und stirbt" daher nicht mit dem den Verweis vornehmenden Objekt. Diese Art der Objektverknüpfung wird verwendet, wenn das Objekt, auf das "gezeigt" wird, ungeachtet dessen, ob das "Zeiger"-Objekt immer noch vorhanden ist oder nicht, vorhanden sein soll.

[0045] Als ein Beispiel für dieses "Springen" von einem Objekt zu einem anderen nehmen wir an, dass ein Client einen Server ersucht, Geld zu einem bestimmten Bankkonto hinzuzufügen. Sobald das Server-Objekt in Form des bestimmten Bankkontos diese Anforderung über den ORB des Servers empfängt, enthält das Objekt einen Zeiger auf ein anderes Server-Objekt mit der Bezeichnung "Kontensaldo", so dass die Verarbeitungssteuerung vom Behälterobjekt "Bankkonto" dem enthaltenen Objekt "Kontensaldo" übertragen wird. Auf diese Weise entsteht eine fortlaufende Abfolgekette von einem Objekt zu einem anderen Objekt. Jeder Sprung geht auch durch den ORB, da der ORB den gesamten Verkehr zwischen den Objekten vermittelt.

[0046] In diesen Situationen, in denen sich jedes nachfolgende Objekt in einer Behälter/enthalten-Beziehung befindet, muss auf jede Behälterebene eine Sperre gesetzt werden. Das heißt, der Zugriff auf jedes der Behälter/enthalten-Objekte muss gesperrt werden. Dadurch wird verhindert, dass eine einen Konflikt auslösende Anforderung eine komplette Löschung des Behälterobjekts durchführt (die auch das enthaltene Objekt zerstören würde, da sie zusammen "leben und sterben"). Ein Behälterobjekt zum Beispiel, das eine Gruppe von Girokonten einschließlich des Girokontos in Form des enthaltenen Objekts auflistet, das ausdrücklich von einem Client-Bankautomaten angefordert wird, darf nicht von einer gleichzeitigen, einen Konflikt auslösenden Client-Anforderung komplett gelöscht werden.

[0047] Folglich muss der ORB nicht nur das enthaltene Objekt sperren, sondern er muss auch jedes Behälterobjekt sperren, welches das angeforderte Objekt enthält. Der ORB erkennt die hierarchische (Behälter/enthalten-)Struktur unter den Objekten, indem er sich die Klassenstruktur anschaut, die in der Definition der Klasse des Behälterobjekts enthalten ist, und indem er ein Attributkennzeichen findet, das sich auf das enthaltene Objekt bezieht. Dieses Attributkennzeichen zeigt an, dass das Behälterobjekt das enthaltene Objekt enthält. Das Kennzeichen ist in der standardmäßigen CORBA IDL (Common Object Request Broker Architecture Interface Definition Language) enthalten, die von der OMG standardisiert wurde. Wenn eines dieser Attributkennzeichen vorhanden ist, weiß der ORB, dass es sich bei der Beziehung zwischen zwei Objekten um eine Beziehung "Behälter/enthalten" im Gegensatz zu der anderen Zeigerbeziehung "verweist auf" ("refers to") handelt.

[0048] Sobald der Sperrschritt **208** in **Fig.** 2 erreicht ist, um alle diese verknüpften Objekte zu erfassen, damit Sperren gesetzt werden können, macht der ORB von seinem fortlaufenden Stapelspeicher (im Speicher) aller Anforderungen Gebrauch, die er vermittelt hat, deren Ausführung aber noch nicht abgeschlossen ist. Das heißt, der Stapelspeicher führt alle

Objekte auf, zwischen denen die Steuerung sprang, während sie den Zeigern von Objekt zu Objekt folgte. Der Inhalt dieses Stapelspeichers ändert sich laufend. Ihm werden Einträge hinzugefügt, wenn der ORB eine Vermittlungsmethode empfängt, und Einträge werden aus ihm entfernt, sobald ein Objekt, das von einem anderen Objekt aufgerufen wurde, von diesem Aufruf zurückkehrt. Gleichzeitig enthält der Stapelspeicher eine Auflistung eines jeden Objekts, das auf ein anderes Objekt gezeigt hat, sowie des Objekts, auf das gezeigt wurde. Dieser Stapelspeicher ist bekannt und im ORB enthalten. Die vorliegende Erfindung nutzt den Stapelspeicher, um automatisch Sperren zu setzen, wie nachstehend beschrieben wird.

[0049] Sobald der ORB eine Anforderung empfängt, die die Zustandsdaten des Objekts prüft oder ändert, die in ihrer niedrigsten Abstraktionsebene ausgedrückt werden, d. h., die direkt auf das Speichermedium des Systems abgebildet werden, führt der Stapelspeicher zu diesem Zeitpunkt alle Objekte in der Behälterkette auf, von dem Objekt in der höchsten Behälterebene bis hinab zu dem Objekt in der niedrigsten Behälterebene. Wenn die Kette vollständig ist und unter der Annahme, dass der Schritt **208** über die Schritte **202**, **203**, **206** und **207** erreicht wird, werden nun die Objekte, die in dem Stapelspeicher aufgeführt sind, mit Sperren belegt, wie nachstehend beschrieben wird. Sobald diese Sperren gesetzt wurden, wird die Anforderung zur Ausführung zugeteilt (**211**), und das Server-Objekt führt die angeforderte Verarbeitung durch (die eine Lese- oder Schreiboperation ihrer gekapselten Zustandsdaten beinhaltet).

[0050] Der zweispaltige Stapelspeicher ist in **Fig.** 3 gezeigt. Eine Spalte **31** führt die Kette der Objektinstanzen auf, die beim Springen von einer Objektinstanz zur nächsten Objektinstanz beteiligt sind. Die andere Spalte **32** führt (zusammen mit jeder Objektinstanz) die Methode auf, auf die auch als die Methode gezeigt wird, die beim Aufrufen des Objekts, auf das gezeigt wird, beteiligt ist. Dies sind die Methoden, die der ORB vermittelt und noch nicht verlassen hat.

[0051] Als Beispiel nehmen wir an, dass ein Client die Anforderung stellt, dass von einem bestimmten Girokonto, das von der Bank (dem Bankserver) als Girokonto Nummer 45 bezeichnet wird, 200 US-Dollar abgehoben werden. Der ORB empfängt diese Client-Anforderung für den Zugriff auf ein erstes Server-Objekt "Konto Nummer 45", und "Hebe 200 US-Dollar ab" ist die Methode (und der zugehörige Parameter), die der Client vom Server auf dieser Objektinstanz aufgerufen haben möchte. In **Fig.** 3 wird dieses aus Objektinstanz/Methode bestehende Paar zum Stapelspeicher **300** als die Elemente **31a**/**32a** hinzugefügt.

[0052] Wenn die Server-Objektinstanz "Konto Nr. 45" die Anforderung empfängt, wird über ihre Methode "withdraw ___" ("Hebe ___ ab") auf ihren gekapselten Zustand zugegriffen, und nach diesem Zugriff wird festgestellt, dass es einen Zeiger auf ein weiteres Objekt "current balance" ("aktueller Saldo") und die Methode "get" ("abrufen") gibt. Das Objekt "current balance" ist ein Objekt einer grundlegenden Art, das direkt auf den Systemspeicher abgebildet wird. Seine Methoden sind generische Zugriffsmethoden (d. h. "get" oder "set"), die vom System bereitgestellt und nicht innerhalb des Anwendungsprogramms festgelegt werden. Damit das System Geld von einem Konto abheben kann, muss es zuerst den aktuellen Kontensaldo kennen. Deshalb erfolgte ein Sprung zu diesem Objekt. Folglich sendet das erste Server-Objekt "Konto Nummer 45" eine serverinterne Anforderung (über den ORB) an das zweite Server-Objekt "current balance". Das erste Server-Objekt enthält das zweite Server-Objekt. Aufgrund des Sprungs wird das neue Paar **31b**/**32b** zum Stapelspeicher **300** hinzugefügt.

[0053] In der gleichen Weise wird jedes Paar, zu dem "gesprungen" wurde, zu dem Stapelspeicher hinzugefügt, während der ORB Methoden vermittelt. Der ORB prüft den Stapelspeicher ständig, um festzustellen, ob das zuletzt hinzugefügte Paar eine Methode enthält, die Zustandsdaten ihres entsprechenden zu einem Paar zusammengefassten Objekts, das direkt auf das Speichermedium abgebildet wird (Schritt **202** von **Fig.** 2), prüft oder ändert. Sobald der ORB feststellt, dass ein neu hinzugefügtes Paar in dieser Phase diese Beziehung aufweist, führt er den Schritt **203** und die anderen nachfolgenden Schritte von **Fig.** 2 durch.

[0054] Im Schritt **208** stellt der ORB fest, welche Objekte im Stapelspeicher Behälterobjekte sind, und dann wird jedes der Behälterobjekte (dies sind die Objekte, die einen Attributanzeiger in ihren Klassendefinitionen haben, der das enthaltene Objekt kennzeichnet) mit einer Leseabsichtssperre oder einer Schreibabsichtssperre (Ir- oder Iw-Sperre) belegt. Ob die Absichtssperren Lese- oder Schreibabsichtssperren sind, hängt davon ab, ob die Methode in dem Paar, welches das Setzen der Sperre ausgelöst hat, versucht, den Zustand der entsprechenden Objektinstanz zu prüfen oder aber zu ändern. Die Objektinstanz des letzten Paares in dem Stapelspeicher, welches das Setzen der Sperre ausgelöst hat, wird dann mit einer Lesesperre (r) oder einer Schreibsperre (w) belegt. In dieser Phase ist der Sperrvorgang abgeschlossen, und der ORB teilt die Anforderung dem Server-Objekt zur Ausführung innerhalb der Transaktion zu (Schritt **211**). Wenn der ORB feststellt, dass die aktuelle Transaktion abgeschlossen ist (festgeschrieben oder abgebrochen wurde), werden alle Sperren für diese Transaktion aufgehoben.

[0055] Nehmen wir das in **Fig.** 3 gezeigte Beispiel.

Sobald das Paar **31b**, **32b** zum Stapelspeicher **300** hinzugefügt wird, stellt der ORB fest, dass dieses Paar eine Methode hat, die die Zustandsdaten ihrer zu einem Paar zusammengefassten Objektinstanz auf der niedrigsten Abstraktionsebene dieser Zustandsdaten (d. h., es sind Daten einer grundlegenden Art) prüft. Das heißt, die Methode ("get") in diesem Paar stellt die Anforderung, die fundamentalen Zustandsdaten (den tatsächlichen aktuellen Saldo des Kontos des Benutzers, wie er im Speichermedium des Rechnersystems gespeichert ist) ihres zu einem Paar zusammengefassten Objekts "current balance" **31b** zu lesen (aufgrund der Verwendung einer erzeugten Methode der unteren Ebene (low-level method) – "get" –, die die Daten einer grundlegenden Art zurückgibt). Das Objekt "current balance" ist als ein Objekt einer fundamentalen Art definiert. Letzteres Objekt ist in dem Objekt "account no. 45" **31a** enthalten, und das Objekt **31b** enthält keinen Zeiger auf ein weiteres Objekt. Außerdem kann ein Attribut für das Objekt **31b** in der Klassendefinition des Objekts **31a** gefunden werden. Daher setzt der ORB eine Leseabsichtssperre (ir-Sperre) auf das Objekt **31a** (es wird eine Lesesperre und keine Schreibsperre verwendet, da die Methode **32b** den Zustand des Objekts **31b** prüft und nicht ändert) und eine Lesesperre auf das Objekt **31b** (aus demselben Grund).

[0056] Der ORB weigert sich, später empfangene Anforderungen zu disponieren, die mit dem Zugriffsrecht, das er bereits einer Anforderung gewährt hat, die zu einer anderen Transaktion bei einer vorherigen Sperre gehört, in Konflikt geraten würden. Diese Sperren werden im ORB des Servers und nicht in der Dienst-Anwendung (**100**) realisiert. Somit braucht sich der Programmierer der Server-Anwendung während der Entwicklung einer Server-Anwendung weder mit Sperrschemata noch mit Transaktionstheorie zu befassen. Für diese Dinge ist stattdessen der ORB zuständig.

[0057] Indem Sperren in der Reihenfolge vom obersten Behälter bis hinab zum Stapelspeicher gesetzt werden, werden Verklemmungen vermieden. Wenn sich außerdem herausstellt, dass es notwendig ist, Lese- oder Schreibsperren auf eine große Anzahl von Methoden von enthaltenen Objekten einzeln anzufordern, kann das bekannte Konzept der "lock promotion" (Verstärken von bestehenden Sperren) zur Anwendung kommen, um eine Lese-(r-) oder Schreib-(w-)Sperre auf die Behälterebene (statt die enthaltene Ebene) zu erhalten.

[0058] Es gibt eine kleine Chance, dass ein aus Objekt/Methode bestehendes Paar im Stapelspeicher in bestimmten Fällen nicht in der Behälterfolge des anderen Paares ist (und folglich hat das Objekt, das auf dieses Objekt gezeigt hat, kein Attributkennzeichen, welches das Objekt des Paares als in ihm enthalten kennzeichnet). Dies würde beispielsweise vorkommen, wenn ein Behälterobjekt auf ein anderes Objekt "verweist", ohne dass es das Objekt "enthält".

[0059] Wie in **Fig.** 4 gezeigt ist, ruft eine Behälterobjektinstanz "a" (die vom ORB von einer Client-Objektinstanz empfangen und folglich zum Stapelspeicher hinzugefügt wird) zuerst eine andere Objektinstanz "b" auf, die in der Objektinstanz "a" enthalten ist (die in einen Kreis eingeschlossenen Ziffern veranschaulichen die Zeitfolge von Ereignissen, z. B. "erstens", "zweitens" usw.). Die Objektinstanz "b" wird folglich zum Stapelspeicher hinzugefügt, der bereits die Objektinstanz "a" enthält. Zweitens ruft die Objektinstanz "b" eine weitere Objektinstanz "c" auf, die sie nicht enthält. Die Objektinstanz "c" wird folglich zum Stapelspeicher hinzugefügt, der bereits die Objektinstanzen "a" und "b" aufführt. Das heißt, die Objektinstanz "b" verweist auf die Objektinstanz "c", "enthält" sie aber nicht. Drittens verweist die Objektinstanz "c" zurück auf die Objektinstanz "b". Folglich wird die Objektinstanz "b" wieder zum Stapelspeicher hinzugefügt, der bereits die Objektinstanzen "a", "b" und "c" aufführt. Viertens ruft die Objektinstanz "b" die Objektinstanz "d" auf, die in der Objektinstanz "b" enthalten ist. Folglich wird die Objektinstanz "d" zum Stapelspeicher hinzugefügt, der bereits die Objektinstanzen "a", "b", "c", "b" und "d" aufführt.

[0060] Wenn der ORB dann feststellt, dass sich die Objektinstanz "d" in der untersten Behälterebene befindet und folglich eine Sperre gesetzt werden sollte, würden gemäß der vorstehenden Beschreibung nur auf das im Stapelspeicher aufgeführte Objekt "b" Absichtssperren gesetzt werden (und das Objekt "d" würde mit einer Lese- oder Schreibsperre belegt werden), da das Objekt "b" das Objekt "d" enthält, und das Objekt "c" nicht das Objekt "b" enthält. Folglich scheint das Objekt "b" in der obersten Behälterebene zu sein. Das Problem ist jedoch, dass das Objekt "a", das mit einer Absichtssperre belegt werden sollte, keine solche Sperre erhielte.

[0061] Diese ungewöhnliche Situation wird so gehandhabt, dass jedes Objekt als ein Knoten in einem Graphen und jede "Behälter"/"enthalten"-Beziehung (die durch Prüfung des Zustands des Stapelspeichers zu dem Zeitpunkt, zu dem eine Sperre gesetzt wird, erkennbar wird) als eine gerichtete Kante in dem Graphen behandelt wird. Sobald das Objekt, dessen grundsätzlicher Zustand geändert wird (das Objekt "d" in dem Beispiel von **Fig.** 4), festgestellt ist, werden alle Objekte mit Sperren belegt, die einen Pfad von dem Knoten, der sie in dem Graphen darstellt, zu dem gesperrten Objekt (dem Objekt "d" in dem Beispiel von **Fig.** 4) haben. Dadurch werden Objekte (wie zum Beispiel das Objekt "a") erfolgreich gesperrt, die Teil der "Behälter"/"enthalten"-Folge sind, aber nicht durch die lineare "Behälter"/"enthalten"-Folge, die direkt durch den Stapelspeicher dargestellt wird, direkt verknüpft sind (sie sind aufgrund

des Vorhandenseins der Objektinstanz "c" in dem Stapelspeicher zwischen den Objektinstanzen "a" und "b" nicht direkt verknüpft). Dies führt nicht dazu, dass Objekte, wie zum Beispiel das Objekt "c", gesperrt werden, da das Objekt "c" keinen Pfad entlang gerichteter Kanten zum Objekt "d" in dem Graphen hat. Wir sperren dann das Objekt "d" mit einer Lese- oder Schreibsperre, sobald die Behälterobjekte mit Absichtssperren belegt wurden. Die Algorithmen für das Auffinden von Pfaden in einem gerichteten Graphen, der von verbundenen Paaren festgelegt wird, sind bekannt und werden hier nicht erörtert.

[0062] Um im Falle eines Systemausfalls (wenn das System beispielsweise während einer Transaktion abstürzt) die Trennung von Transaktionen aufrechtzuerhalten, während der Ausgang einer Transaktion noch ungewiss ist, stellt der ORB seine Sperren als Teil der Systemwiederherstellung wieder her, bevor die Anwendungsarbeit wieder aufgenommen wird. Dies geschieht vorzugsweise, indem eine wiederherstellbare Sperreinrichtung in den transaktionssensitiven Sperren-Werkzeugsatz aufgenommen wird.

[0063] Während die bevorzugte Ausführungsform als eine Ausführungsform beschrieben wurde, die von einem ORB durchgeführt wird, könnte ein Übersetzer einer objektorientierten Programmiersprache ebenfalls zur Ausführung der beschriebenen Funktionen verwendet werden. Das bekannteste Beispiel eines solchen Übersetzers ist die virtuelle Java-Maschine ("Java" ist ein Warenzeichen von Sun Microsystems), aber andere können ebenfalls verwendet werden. Ein Übersetzer kann an der Semantik der Anweisungen der Programmlogik, die die Klassendefinitionen bilden, "Behälter"/"enthalten"-Beziehungen zwischen Objekten feststellen.

[0064] Während die bevorzugte Ausführungsform als eine Ausführungsform beschrieben wurde, die die Sperrprozedur in Bezug auf Objekte beginnt, die direkt versuchen, gekapselte Daten zu lesen oder zu schreiben (Schritt **202**), könnte das Setzen von Sperren auch durch andere Methoden ausgelöst werden, indem eine Markierung (ein Kennzeichen) in der Klassenspezifikation (Interface Definition Language oder IDL) verwendet wird, um solche Methoden als Methoden zu kennzeichnen, die "Schreib-/Lese"-Funktionen in Bezug auf die betreffenden Objekte haben. Die Kennzeichen würden semantisch angeben, dass immer, wenn die gekennzeichnete Methode vom ORB vermittelt wird, eine Schreib-/Lesesperre gesetzt werden sollte.

[0065] Bei der vorstehenden Erörterung wird das Setzen von Sperren unter der Annahme beschrieben, dass eine Methode in einer Transaktion mit derselben Methode bei einer ersten Transaktion einen Konflikt hervorruft. Zum Beispiel verhindert eine bisherige Sperre auf einem Objekt, die von der Methode "withdraw()" bei einer ersten Transaktion ausgelöst wird, dass dieselbe Methode "withdraw()" bei einer anderen Transaktion für ein bestimmtes Objekt disponiert wird, bis die erste Transaktion entweder festgeschrieben oder abgebrochen wird. Jedoch kann der Konflikt auch als ein zwischen einer Methode und einer Untergruppe von Methoden in der Klasse vorhandener Konflikt definiert werden. Ein Klassenspezifikationskennzeichen könnte zur Kennzeichnung solcher Beziehungen verwendet werden.

[0066] Zwar wurde ein zweispaltiger Stapelspeicher beschrieben, doch könnte auch ein einspaltiger Stapelspeicher verwendet werden, der nur die Objektinstanzen aufführt. Die entsprechenden Methoden sind nur bei der Feststellung wichtig, ob auf fundamentale Zustandsdaten (Daten, auf die nicht über mehrere Objekte, sondern direkt von einem Objekt zugegriffen wird) des zuletzt hinzugefügten Objekts zugegriffen wird. Wenn die Objektinstanzen zum Stapelspeicher hinzugefügt werden, können die entsprechenden Methoden somit geprüft werden, und wenn sie dieses Kriterium nicht erfüllen, können sie verworfen werden.

[0067] Es gibt oft Situationen, in denen ein Objekt einen Zeiger auf eine Datenstruktur (wie zum Beispiel eine Liste oder einen Baum) enthält. Das Objekt "enthält" immer noch die Datenstruktur, aber jedes Element (Knoten) der Struktur "enthält" weder die Elemente unter ihm noch verweist es einfach auf eine unabhängige Struktur. Jedes Element ist mit den anderen nur durch deren Aufnahme in das strukturierte Format der Datenstruktur verbunden. Als weitere Verbesserung der vorliegenden Erfindung können solche Datenstrukturen berücksichtigt werden, indem Aktualisierungen der tatsächlichen Datenwerte in einem Element einer Datenstruktur nur in Bezug auf andere Aktualisierungen derselben Daten gesperrt werden. Aktualisierungen von Zeigern, die die Topologie der Datenstruktur beibehalten, müssen auf der Ebene der gesamten Datenstruktur gesperrt werden. Dies geschieht, indem Methoden erkannt werden, die die Topologie der Struktur (z. B. add_node, delete_node, move_node oder beliebige Schreib-/Leseverfahren mit ausgelagerten Zeigern) ändern und diese Methoden auf der Ebene der ganzen Gesamtdatenstruktur sperren. Methoden, die einfach nur den Wert der Daten an einem einzelnen Element (Knoten) lesen beziehungsweise an ein einzelnes Element (Knoten) schreiben, werden auf der Elementebene gesperrt (mit Absichtssperren auf der Gesamtstruktur, wie vorstehend für die "Behälter/enthalten"-Situation, die häufiger anzutreffen ist, beschrieben wurde).

[0068] Die Erfindung kann auch als ein Rechnerprogrammprodukt realisiert werden, das auf einem rechnerlesbaren Speichermedium wie zum Beispiel einem Magnetplattenlaufwerk, einem optischen Plat-

tenlaufwerk oder einem Halbleiterspeicher gespeichert wird.

Patentansprüche

1. Rechnersystem, das Folgendes umfasst:
ein Mittel, um Arbeitsanforderungen zu empfangen und sie zur Ausführung in dem Rechnersystem zuzuteilen;
ein Mittel, um die von dem Mittel zum Empfang und zur Zuteilung empfangenen Anforderungen zu verarbeiten, um die angeforderte Arbeit auszuführen; und
dadurch gekennzeichnet, dass das System des Weiteren Folgendes enthält:
ein Mittel, um die Gleichzeitigkeitssteuerung in Bezug auf gleichzeitige Anforderungen durchzuführen, wobei sich das Mittel zur Durchführung der Gleichzeitigkeitssteuerung in dem Mittel zum Empfang und zur Zuteilung befindet.

2. Rechnersystem nach Anspruch 1, wobei das Mittel zum Empfang und zur Zuteilung ein Objektanforderungsvermittler ist.

3. Rechnersystem nach Anspruch 1, wobei das Mittel zum Empfang und zur Zuteilung ein Übersetzer einer objektorientierten Programmiersprache ist.

4. Rechnersystem nach einem vorhergehenden Anspruch, wobei das Mittel zur Durchführung der Gleichzeitigkeitssteuerung die Auswirkungen einer Gruppe von zusammengehörenden Arbeitsanforderungen, die eine Transaktion bilden, isoliert.

5. Prozess zur Durchführung der Gleichzeitigkeitssteuerung an Arbeitsanforderungen in einem Rechnersystem, der die folgenden Schritte umfasst:
(a) Empfangen einer Arbeitsanforderung;
(b) Auswerten eines angeforderten Objekts und einer angeforderten Methode, die in der Arbeitsanforderung enthalten sind; und
(c) Sperren der Vermittlung der Methode.

6. Prozess nach Anspruch 5, wobei der Auswerteschritt (b) die folgenden Teilschritte beinhaltet:
(b1) Prüfen, ob die erste Arbeitsanforderung den Zugriff auf fundamentale Zustandsdaten einer grundlegenden Art des Objekts anfordert, die direkt auf das Speichermedium des Rechnersystems abgebildet wird; und
(b2) Prüfen, ob die erste Arbeitsanforderung Teil einer aktuellen Transaktion ist.

7. Prozess nach Anspruch 6, wobei der Auswerteschritt (b) des Weiteren die folgenden Teilschritte beinhaltet:
(b3) Prüfen, ob das zu der Arbeitsanforderung gehörende Objekt bereits von einer vorhergehenden Anforderung gesperrt wurde, die zu derselben Transaktion wie die Arbeitsanforderung gehört; und
(b4) Prüfen, ob das zu der Arbeitsanforderung gehörende Objekt bereits von einer einen Konflikt auslösenden Sperre gesperrt wurde, die zu einer anderen Transaktion als der Transaktion der Arbeitsanforderung gehört.

8. Prozess nach einem der Ansprüche 5 bis 7, wobei der Sperrschritt (c) die folgenden Teilschritte beinhaltet:
(c1) Erzeugen eines Stapelspeichers mit Objektinstanzen, während die Arbeitsanforderung von dem angeforderten Objekt an mindestens ein anderes Objekt übertragen wird, das das angeforderte Objekt aufruft;
(c2) Prüfen des Stapelspeichers, um das Objekt in der obersten Behälterebene und das Objekt in der untersten Behälterebene festzustellen; und
(c3) Setzen einer Absichtssperre auf das Objekt in der obersten Behälterebene.

9. Prozess nach Anspruch 8, wobei der Sperrschritt (c) des Weiteren die folgenden Teilschritte beinhaltet:
(c4) Setzen von Absichtssperren auf alle restlichen Objekte in dem Stapelspeicher von der obersten Behälterebene bis hinab zur untersten Behälterebene, diese jedoch nicht eingeschlossen; und
(c5) Setzen einer Lese- oder Schreibsperre auf das Objekt in der untersten Behälterebene.

10. Prozess nach Anspruch 8, wobei in dem Teilschritt (c2) ein Attributkennzeichen einer Behälterobjekt-Klassendefinition geprüft wird, um die Behälterebene festzustellen.

11. System nach einem der Ansprüche 1 bis 4, wobei das Rechnersystem ein Server ist und die Arbeitsanforderungen von mindestens einem Client-Rechnersystem auf einer heterogenen Plattform an dem Server empfangen werden.

12. Prozess nach einem der Ansprüche 5 bis 10, wobei die Schritte (a), (b) und (c) von einem Objektanforderungsvermittler durchgeführt werden.

13. Prozess nach einem der Ansprüche 5 bis 9, wobei die Schritte (a), (b) und (c) von einem Übersetzer einer objektorientierten Programmiersprache durchgeführt werden.

14. Auf einem rechnerlesbaren Speichermedium gespeichertes Rechnerprogrammprodukt, das über ein Code-Mittel verfügt, um die in den vorstehenden Prozess-Ansprüchen aufgeführten Funktionen auszuführen.

Es folgen 4 Blatt Zeichnungen

Anhängende Zeichnungen

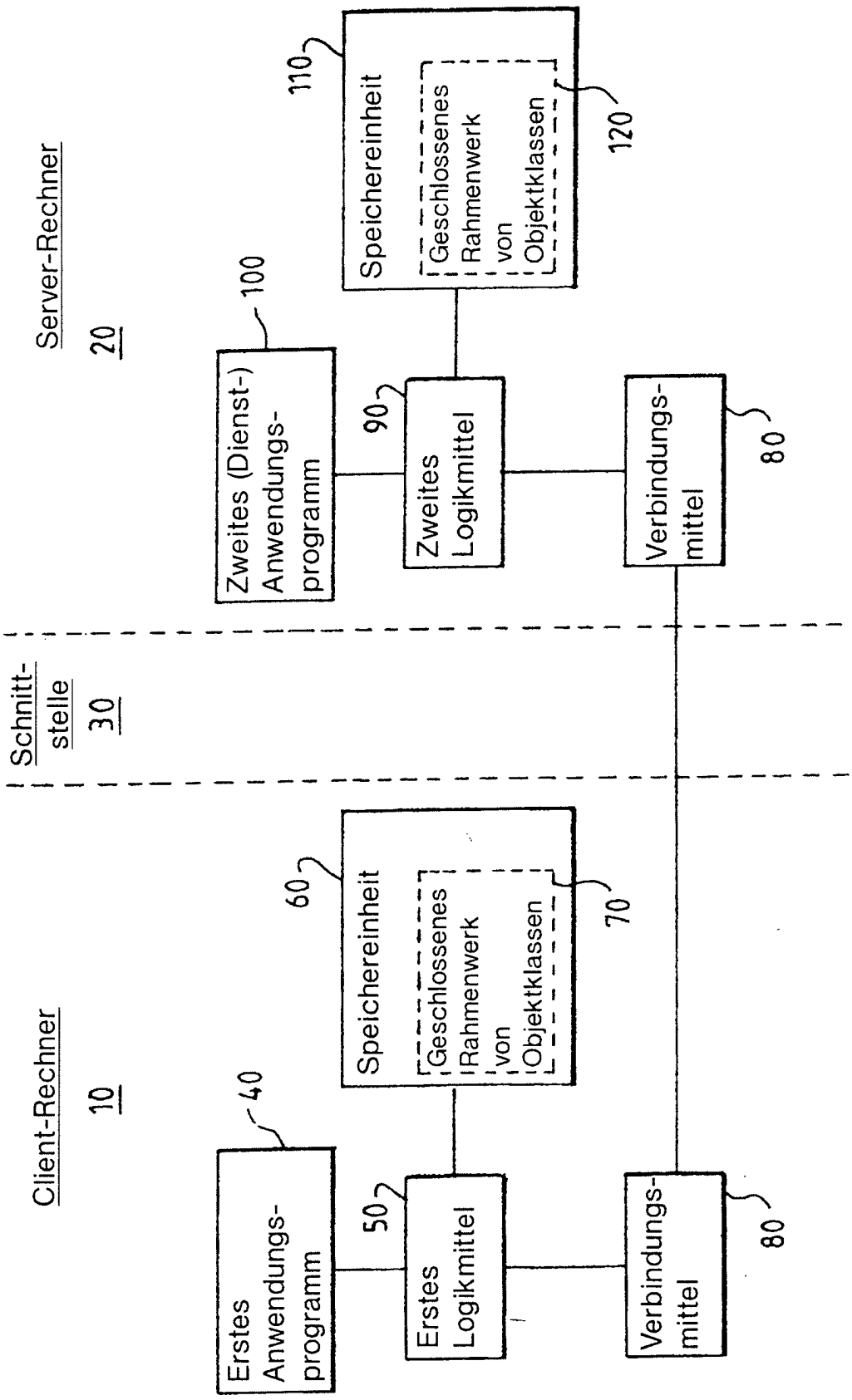


FIG. 1

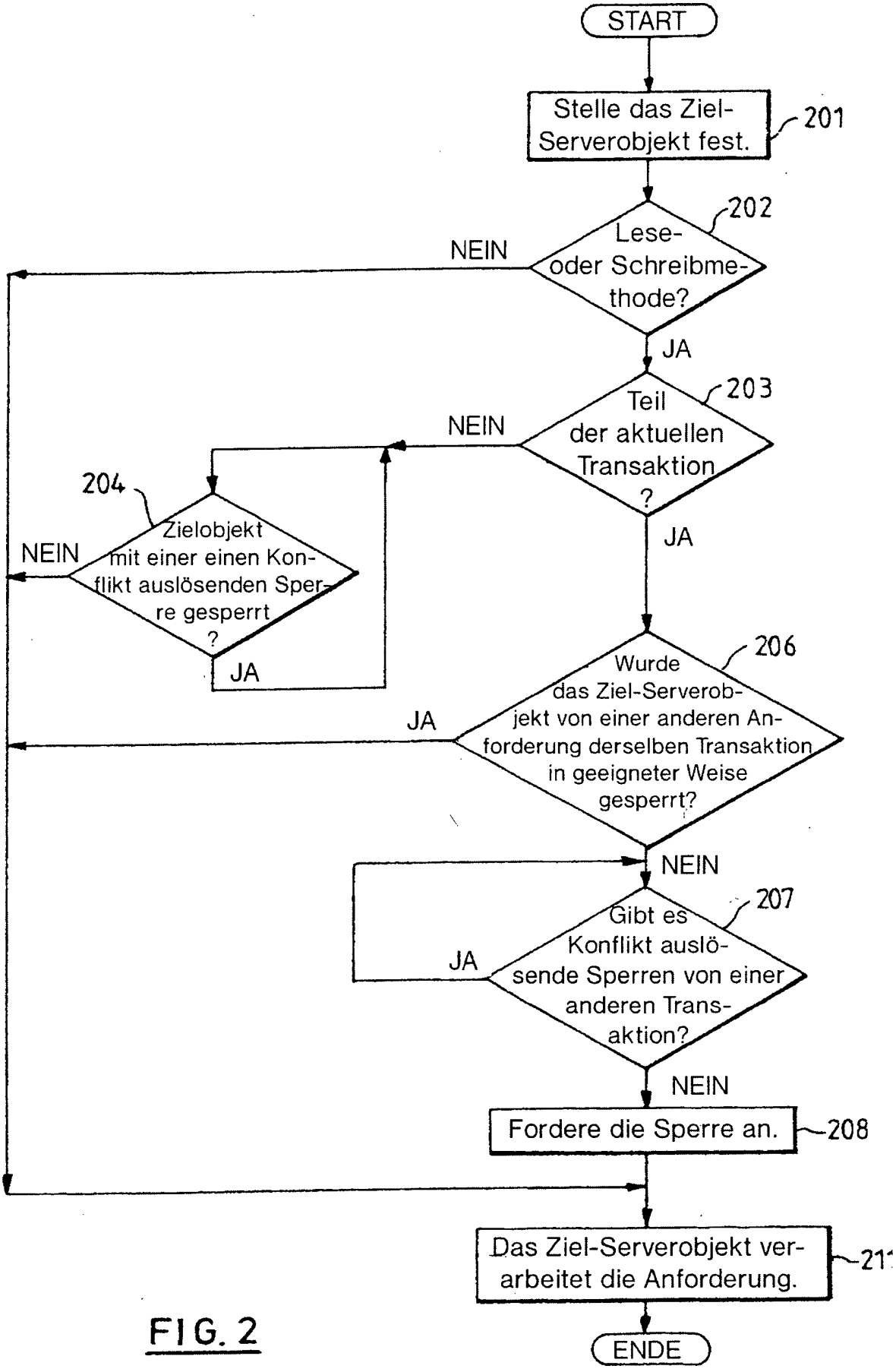

START
Stelle das Ziel-Serverobjekt fest.
201
202
Lese- oder Schreibmethode?
NEIN
JA
203
Teil der aktuellen Transaktion ?
NEIN
JA
204
Zielobjekt mit einer einen Konflikt auslösenden Sperre gesperrt ?
NEIN
JA
206
Wurde das Ziel-Serverobjekt von einer anderen Anforderung derselben Transaktion in geeigneter Weise gesperrt?
JA
NEIN
207
Gibt es Konflikt auslösende Sperren von einer anderen Transaktion?
JA
NEIN
Fordere die Sperre an.
208
Das Ziel-Serverobjekt verarbeitet die Anforderung.
21
ENDE

FIG. 2

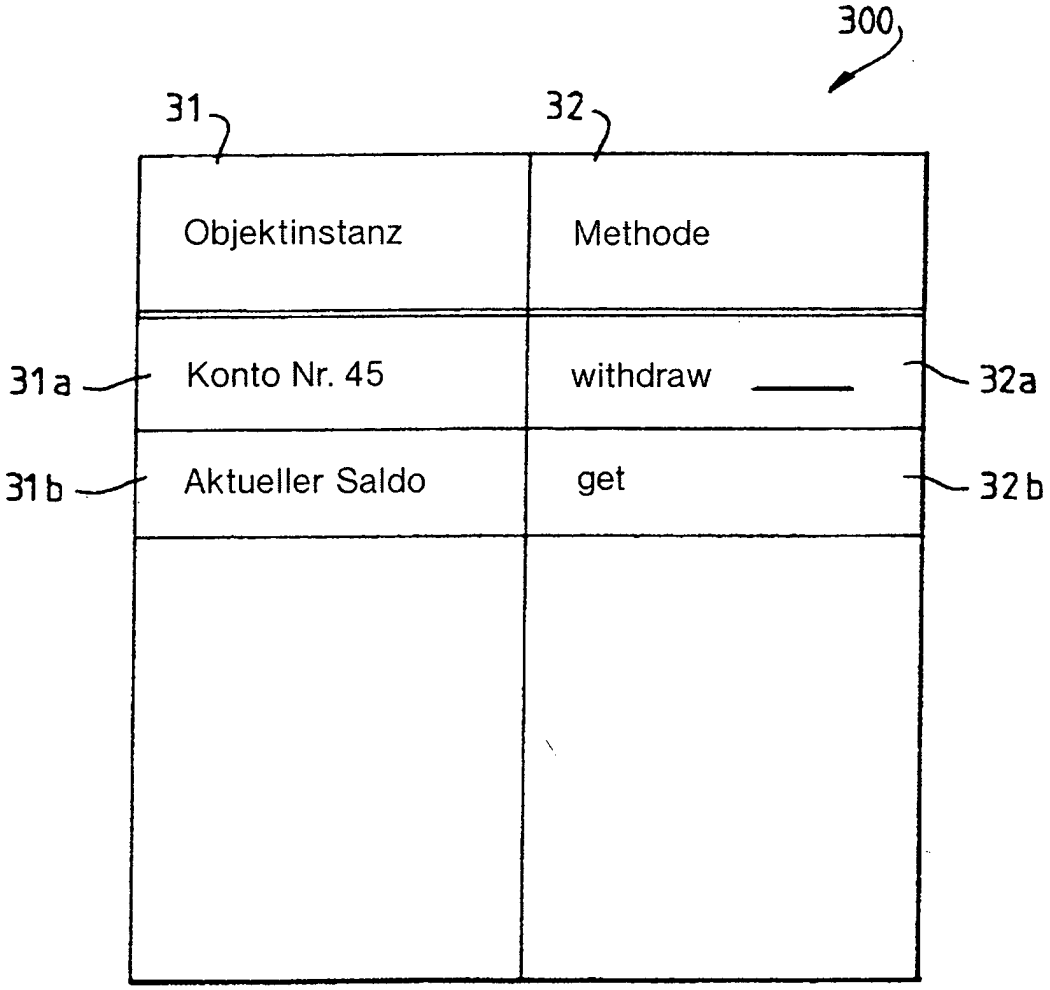

300
31
32
Objektinstanz
Methode
31a
Konto Nr. 45
withdraw
32a
31b
Aktueller Saldo
get
32b

FIG. 3

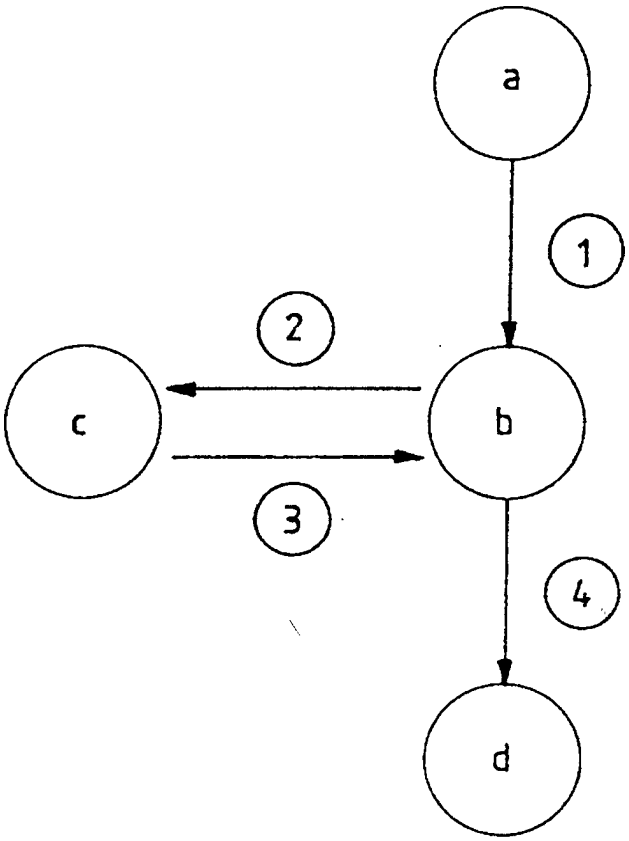


FIG. 4