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

(10) **DE 696 35 099 T2** 2006.06.29

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 750 412 B1**
(21) Deutsches Aktenzeichen: **696 35 099.8**
(96) Europäisches Aktenzeichen: **96 108 876.2**
(96) Europäischer Anmeldetag: **03.06.1996**
(97) Erstveröffentlichung durch das EPA: **27.12.1996**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **24.08.2005**
(47) Veröffentlichungstag im Patentblatt: **29.06.2006**

(51) Int Cl.[8]: ***H04L 29/06*** (2006.01)

(30) Unionspriorität:
472227 **07.06.1995** **US**

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

(73) Patentinhaber:
Hewlett-Packard Development Co., L.P., Houston, Tex., US

(74) Vertreter:
BOEHMERT & BOEHMERT, 80336 München

(72) Erfinder:
Ratner, Mitchell, Los Gatos, US; Blevins, Michael R., Felton, US; Schorow, David J., San Jose, US; Limprecht, Rodney T., Scotts Valley, US

(54) Bezeichnung: **Verfahren und Vorrichtung für kontextempfindliches Pfadsende**

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

HINTERGRUND DER ERFINDUNG

[0001] Bei typischen Online-Transaktionen zwischen einer Vielzahl von Clients und einem oder mehreren Servern eines Serverpools ist es möglich, daß die Client-Anwendungen eine große Anzahl sehr kurzer Nachrichten oder Anfragen (oft zehntausende Datenpakete, die jeweils weniger als 32 k groß sind) senden, die von einem bestimmten Server in dem Serverpool verarbeitet werden müssen. Die Clients und Server arbeiten unabhängig (asynchron) voneinander, und die Übertragung von Daten durch das Netz, welches die Clients und die Server verbindet, ist ebenfalls asynchron, da die Datenpakete mit einem Nachrichten-Weiterleitungsverfahren gesendet werden. Es wäre in einer solchen asynchronen Ein-/Ausgabeumgebung vorteilhaft, wenn einer Gruppe von Nachrichten ein „Kontext" gegeben werden könnte, so daß ein bestimmter Server die Gruppe von Nachrichten so schnell wie möglich bedienen könnte, ohne jedesmal, wenn eine Nachricht empfangen wurde, teure und zeitaufwendige Such- und Warteschlangeprozesse heranzuziehen, wie es im Stand der Technik der Fall ist. Ein Ansatz, dieses Problem zu lösen, ist das Versorgen der Nachrichten, der Server- und der Client-Anwendungen mit Informationen, um eine Art zeitlich zugewiesener Verbindung zwischen dem Server und dem Client aufzubauen. Die vorliegende Erfindung ist auf einen solchen „kontextsensitiven bzw. kontextabhängigen Dialog" gerichtet.

[0002] Andere Programme haben den Ansatz verfolgt, kontextsensitive Dialoge („pathsends") vorzusehen, beispielsweise Tuxedo, Encino und CPIC (das Protokoll für einige diese Programme ist in Veröffentlichungen des Komitee für den XOpen-Standard zu finden), jedoch sieht die vorliegende Erfindung einen neuen und verbesserten Dialog vor, der zahlreiche Vorteile gegenüber den vorherigen Ansätzen aufweist.

[0003] Die Druckschrift EP 0 600 285 A1 offenbart eine Objektschnittstelle, die drei Modi der Interobjekt-Kommunikation unterstützt, d.h. Nachrichtenverarbeitung, Dialog-Kommunikation und Fern-Prozeduraufruf. Von einer Vielzahl von Clients stammende Anfragen bzw. von Servern vorgesehene, darauf reagierende Dienste werden von einem Dienstvermittler (Servicebroker) vermittelt. Bei einer Dialog-Sitzung hält der Server den Daten- und Ausführungsstatuskontext bei, um Sequenzen von voneinander abhängende Client-Anfragen auszuführen. Dialog-Server-Sitzungen müssen explizit von dem Client-Programm beendet werden. Mit anderen Worten offenbart EP 0 600 285 Dialog-Server, die Spezial-Server sind, welche Kontextinformationen hinsichtlich eines Dialogs speichern können. Jedoch können solche Dialog-Server weder einen Dialog aufbauen, noch einen Dialog aufrechterhalten oder abschließen.

ABRISS DER ERFINDUNG

[0004] Ein Ziel der vorliegenden Erfindung ist es, die Leistungsfähigkeit für einen Empfangsprozeß zu optimieren, welche sich auf die Verarbeitung einzelner und dennoch logisch miteinander verknüpfter Anfragen bezieht, indem die Notwendigkeit für den Empfangsprozeß vermieden wird, den vorherigen Kontext wiederherzustellen, um es dem Empfangsprozeß zu ermöglichen, die aktuelle Anfrage zu verarbeiten.

[0005] Ein weiteres Ziel der vorliegenden Erfindung ist es, eine größere Funktionalität und eine vereinfachte Handhabung von Anwendungen vorzusehen, die mehrere, logisch miteinander in Beziehung stehende Eingangs-/Ausgangsprozesse ausführen müssen.

[0006] Ein weiteres Ziel der Erfindung liegt darin, mit Problemen umgehen zu können, deren Lösung das Senden logischer Nachrichten mit einer Größe von mehr als 32 k Bytes erfordert, ohne mit jeder gesendeten Nachricht den Kontext wiederherstellen zu müssen.

[0007] Ein weiteres Ziel ist es, bei jedem logisch in Beziehung stehenden Eingangs-/Ausgangsprozeß einen auf eine Transaktion bezogenen Kontext zu übertragen.

[0008] Ein weiteres Ziel ist es, einen Mechanismus vorzusehen, der das Verringern der Verarbeitungszeit für individuelle Anfragen innerhalb eines Dialogs (Sitzung) ermöglicht, indem die Möglichkeit vorgesehen wird, die Erzeugungsprozesse mit den Empfangsprozessen für die Dauer des Dialogs während der Laufzeit zu verbinden. Dadurch wird gewährleistet, daß alle Nachrichten innerhalb des Dialogs zwischen den gleichen Erzeugungs- und Empfangsprozessen ablaufen, und es wird ermöglicht, daß der Empfangsprozeß den sich auf den Dialog beziehenden Kontext beibehält.

[0009] Die vorliegende Erfindung sieht ein Verfahren zum Senden von Information in einem Server-Client-Netz gemäß Anspruch 1 sowie ein entsprechendes System gemäß Anspruch 19 vor.

KURZBESCHREIBUNG DER ZEICHNUNGEN

[0010] Fig. 1 offenbart ein typisches Netz mit mehreren Ausführungslinien (multi-threaded), das die vorliegende Erfindung realisiert.

[0011] Fig. 2 offenbart eine typische Online-Überweisungsaktion, welche die vorliegende Erfindung anwendet.

[0012] Fig. 3 offenbart ein typisches Netz mit mehreren Ausführungslinien (multi-threaded), das die vorliegende Erfindung realisiert.

[0013] Fig. 4 offenbart ein Schaltbild eines Primärspeichers, der zum Zwecke der vorliegenden Erfindung in Blöcke segmentiert ist.

[0014] Fig. 5 offenbart ein Flußdiagramm, das die Verwendung eines Aspekts einer Ausführung der vorliegenden Erfindung durch eine Client-Anwendung darstellt.

[0015] Fig. 6 offenbart ein Blockdiagramm zur Darstellung der Art und Weise, wie Objekte, welche einen Aspekt der vorliegenden Erfindung vorsehen, miteinander in Beziehung stehen.

DETAILLIERTE BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGEN

[0016] In einer typischen Online-Anwendung der vorliegenden Erfindung kann ein Bankautomat mit einer „Client"-Anwendung (Softwareprogramm) und/oder mit einem Knoten verbunden sein. Ein Knoten bzw. eine Workstation ist eine softwarebetriebene Hardware, die einen Prozessor, primären und sekundären Speicher und Eingabe-/Ausgabeeinrichtungen umfaßt, und die typischerweise mehrere Anwendungen aufweist. In dem allgemeinsten Fall kann ein einzelner Knoten sowohl Client- als auch Server-Anwendungen umfassen und ablaufen lassen.

[0017] Eine typische Online-Anwendung kann erfordern, daß eine Client-Anwendung und/oder ein Netz mit der Hostserver-Anwendung und/oder mit einem Knoten für eine kurze Zeitdauer (nur wenige Sekunden) kommuniziert. Für jeden Serverknoten können Tausende solcher Client-Knoten vorgesehen sein, sowie eine Vielzahl miteinander verbundener Server-Knoten. Typischerweise werden zu jedem Zeitpunkt Zehntausende von Nachrichten zwischen Client und Server übertragen.

[0018] Eine typische Transaktion zwischen einem Client und einem Server ist eine Belastungs/Guthaben-Transaktion, und eine solche Transaktion weist eigene spezifische Probleme auf. Am Client-Knoten kann ein Bankkunde sein Girokonto durch eine Barauszahlung belasten. Die Transaktion zwischen dem Client und dem Server muß überwacht werden, so daß im Falle des Eintretens eines unerwarteten Fehlers während der Kommunikation zwischen dem Client und dem Server die Transaktion zu dem anfänglichen Zustand vor dem Fehler „rückgängig gemacht" werden kann; ansonsten würde der Kunde oder die Bank dafür, daß eine Transaktion nicht abgeschlossen wurde, unberechtigterweise belastet werden bzw. eine Gutschrift erhalten. Ferner können Client-Anwendungen asynchron zu Server-Anwendungen ablaufen, und während einer typischen Sitzung kann die Client-Knotenanwendung Aufgaben ausführen, beispielsweise einen Bildschirmausgabe vorsehen, eine Quittung drucken, oder ähnliches, während eine Server-Knotenapplikation eine Datenbankoperation bezüglich der Datenbanken der Bank ausführen kann. Jedoch können Umstände eintreten, in denen ein bestimmter Client auf einen bestimmten Server zugreifen und wiederholt zugreifen muß. Ferner können Umstände eintreten, in denen von einem Girokonto eine Anzahl ausstehender Zahlungen erlangt werden müssen. Das Resultat kann eine Größe aufweisen, für die der Nachrichtenspeicher nicht ausreichend ist. Durch Senden einer zweiten und weiterer folgender Nachricht an den gleichen Server wird erfindungsgemäß der Overhead vermieden, der sich durch erneutes Stellen der Anfrage ergibt. Die vorliegende Erfindung spricht diese Belange ebenfalls an, um ein Minimum an Systemressourcen in einer möglichst schnellen Weise zu verwenden.

[0019] Ein weiterer Aspekt der vorliegenden Erfindung ist es, so wenig Overhead wie möglich beim Aufbauen oder Abbrechen einer zwischen dem Client und dem Server vorliegenden Verbindung (Sitzung) vorzusehen. Das Verfahren zum Ausführen der vorliegenden Erfindung verwendet diese Eigenschaft in weitestem Umfang. Daher ist der Aufwand zum Herstellen oder Abbrechen der Verbindung (Betriebssystemressourcen, einschließlich Speicher; Nutzungszeit des Betriebssystems) gering.

[0020] Bezugnehmend auf die Fig. 1 wird mindestens ein Client-Knoten **110** dargestellt (der mindestens einen Client oder ein Client-Programm oder eine Anwendung **115'** aufweist), der mit zumindest einem Server-Knoten **120** (der zumindest ein Server-Programm oder eine Anwendung **125** aufweist) verbunden werden muß, welcher einen gewissen Dienst anbietet, beispielsweise Datenbankoperationen. Die Kommunikation zwischen den Client- und Serveranwendungen wird mittels einer Betriebssystem-Prozeßapplikation **130** ausgeführt, die in der vorliegenden Erfindung „Linkmon" (LM) genannt wird, welche in einem von den Client- und Server-Knoten **110**, **120** getrennten Knoten **132** vorliegen kann, und typischerweise dort vorgesehen ist. Wie üblich, kommuniziert eine Vielzahl von Client-Knoten mit jedem einzelnen Server, beispielsweise mit den Client-Knoten **110**–**115**, und in der allgemeinen Umgebung liegen eine Vielzahl von Server-Knoten **120**–**122** vor, die alle miteinander kommunizieren, wie durch die Kommunikationsleitungen **126** angegeben ist. Die Kommunikationsleitungen **137** geben an, daß Nachrichten zwischen den Knoten ausgetauscht werden.

[0021] In einer bevorzugten Ausführung der Erfindung kommuniziert die Linkmon-Anwendung **130** mit der Betriebssystem-Anwendung **140** und ruft diese auf, wobei diese ebenfalls dem Betriebssystem ent-

sprechen muß, das die Server-Anwendung und die Client-Anwendung gemeinsam haben, und das in Fig. 1 in Knoten **132** vorliegt. In dieser bevorzugten Ausführung liegt der Knoten **132** in einen Tandem-Computer vor, und das Betriebssystem **140** läuft innerhalb des Guardian Tandem-Betriebssystems (Non-Stop Kernel OS, Version B30.02) ab.

[0022] Die Betriebssystem-Anwendung **140**, welche in der Hardware **132** vorliegt, wird auch „Prozeßmonitor" genannt, der als Knoten **145** dargestellt ist, und optimiert das Dienstverhältnis zwischen Client und Server, um den Server auf die beste Art und Weise mit den Clients zu verbinden. In der Tandem-Umgebung wird dieser Prozeßmonitor „Pathway" oder TS/MP („transaction services/massively parallel") genannt, da es dieser parallelen Servern ermöglicht, ein Problem als Tandem abzubearbeiten. Typischerweise eignen sich TS/MP-kompatible Server gut für Online-Transaktionsverarbeitung (OLTP), und werden hierfür eingesetzt. In solchen TS/MP-Umgebungen ist das Erweitern der Computerressourcen um ein Faktor N eine von N Computern vorgesehene additive Funktion, so daß das Hinzufügen von N weiteren Computerknoten zu dem TS/MP-Netz die Computerressourcen um N verstärkt, da die Computer größtenteils unabhängig voneinander und parallel arbeiten.

[0023] Wie in der Fig. 1 dargestellt ist, läuft die vorliegende Erfindung typischerweise auf „kaskadierten Servern" **120**–**122** ab, wobei ein Server die Serverfunktionen an einen anderen Server an einem anderen Ort delegieren kann. Ferner ist dargestellt, daß typischerweise ein einzelner Server-Knoten Dienste für eine Vielzahl von Client-Knoten vorsieht. Zudem kann ein bestimmter Server-Knoten einer bestimmten Gruppe von Client-Knoten zugeordnet sein, so daß nicht jeder Server jeden beliebigen Client-Knoten bedienen kann. In der Vergangenheit hätten Server-Anwendungen bei der Verwendung von nicht-kontextabhängigen Pathsends in einer Nachricht zwischen den Client- und Server-Anwendungen alle mit dem Client zusammenhängenden Zustände erneut erstellen müssen. Dieses Problem wird durch das kontextabhängige bzw. kontextsensitive Pathsend der vorliegenden Erfindung beseitigt.

[0024] Es ergeben sich Schwierigkeiten in dem Client-Server-Verhältnis innerhalb der Tandem-Umgebung dadurch, daß eine Vielzahl von Servern **120**–**122** bzw. Server-„Pool" **190** im allgemeinen eine noch größere Anzahl von Client-Knoten (im allgemeinen Clients **105**) und deren zugeordneten Anwendungen bedienen müssen (wobei mehrere Anwendungen pro Knoten möglich sind). Die Idee, die dem TS/MP-Knoten **145** zugrunde liegt, ist es, die Vielzahl von Servern **190** hinsichtlich der Vielzahl von Clients **105** zu multiplexen, so daß die Transaktion-Zugriffszeit zum Aufbau einer Verbindung angemessen kurz ist. Bei einigen Transaktionen ist es möglich, daß der Client für wenige Sekunden auf den einen Server zugreifen muß. Typischerweise wird eine Warteschlange mit Clients ausgebildet, die auf den Server zugreifen möchten.

[0025] Bei Tandem-Umgebungen nach dem Stand der Technik gab es nur einen allgemeinen Dienstsitzungs(Dialog)-Typ, der von dem Client angefordert werden konnte: eine kontextfreie Anfragesitzung (request session). Bei einer kontextfreien Anfrage wird eine Dienstanfrage bzw. eine Anfrage hinsichtlich eines Dienstes innerhalb des Servers ausgeführt, jedoch bleibt der Server hinsichtlich der Anfrage kontextfrei. Daher würde der gleiche Client, der einen bestimmten Server mit einer gleichen darauffolgenden Anfrage aufrufen würde, wahrscheinlich einen gänzlich anderen Server mit einem Kontext erhalten, der nur von der Client-Anwendung gespeichert wird. Der Server müßte dann alle Zustände erneut herstellen, die mit dem Clients verknüpft sind, um die korrekten Verhältnisse einzurichten, wobei dies auf eine zeitaufwendige Weise geschieht. Die vorliegende Erfindung vermeidet diese Notwendigkeit. Die vorliegende Erfindung bietet ferner eine Schichtaufteilung verschiedener Anwendungen über den Tandem-Pathway oder über das TS/MP-Subsystem.

[0026] In einer kontextabhängigen Sitzung würde bei einer Vielzahl von Sitzungen der Client nach einer ersten Nachricht in einem Dialog für alle darauffolgenden Nachrichten in dem Dialog den gleichen Server wie die erste Nachricht erhalten.

[0027] In der bevorzugten Ausführung der Implementierung des kontextabhängigen Client-/Server-Pathsends besteht ein Aufruf eines bestimmten API(Application Programming Interface)-Typs (im Anhang finden sich Spezifikationen der APIs, die hier beigefügt sind, wie im weiteren beschrieben ist), der „BEGIN DIALOG" genannt wird, welcher eine Verbindung zwischen Client und Server aufbaut und gleichzeitig die erste Anfrage (Send) stellt. Ein Dialog (Sitzung) kann entweder von dem Server oder von dem Client unterbrochen werden. Hinsichtlich der Unterhaltung bzw. Beibehaltung einer Sitzung durch den Server bestätigt ein Server die von dem Client kommende Nachricht nach der Übertragung der Nachricht von dem Client-Knoten an den Server, welcher eine Verbindung aufbaut, mit einer Bestätigungsnachricht (acknowledgment message), die von dem Server zurück an den Client geleitet wird. Solange der Server mit einem speziellen Fehlercode, FeContinue (FEHLERCODE 70), antwortet, bleibt der Client mit dem Server verbunden und die Anfrage sowie die Verbindung wird aufrechterhalten. Zu diesem Zeitpunkt sind der Server und der Client eineindeutig miteinander verbunden und es gibt keine Notwendigkeit, jede von dem Client stammende nachfolgende Nachricht dahingehend zu überprüfen, ob diese von dem richtigen Client kommt.

[0028] Wenn zu einem beliebigen Zeitpunkt der Server mit einer Fehlernachricht antwortet, die sich von dem speziellen Fehlercode (FEHLER 70 oder FeContinue) unterscheidet, wobei dies der Code für „Setze diese Sitzung fort" (continue) ist, wird von dem Server eine Antwort an den Client gegeben, jedoch wird die Verbindung zwischen dem Server und dem Client direkt danach unterbrochen. Danach, wenn die Verbindung unterbrochen ist, ist es gut möglich, daß der Client, wenn er das nächste Mal eine Verbindung aufbaut, diese mit einem anderen Server aufbaut. Fehlernachrichten, die sich von dem speziellen Fehlercode FEHLER 70 (FeContinue oder „Continue") unterscheiden, umfassen Fehlernachrichten, die eine normale Beendigungsanfrage anfordern, Rückgabecodes, wie FeOK (wodurch eine Sitzung beendet wird), sowie Fehlercodes, die mit anormalen Ereignisse verknüpft sind.

[0029] Hinsichtlich der Aufrechterhaltung einer Sitzung durch den Client-Knoten kann der Client ebenfalls, als Reaktion auf einer Anfrage von dem Server, die Sitzung beenden, indem er eine „DIALOG ABORT"-Nachricht zurücksendet. Dadurch wird die Sitzung sofort beendet, wodurch ein unterbrochener Dialog gewährleistet wird. Die Gewährleistung unterbrochener Dialoge ist, wie oben bemerkt, bei bestimmten Bankautomaten-Transaktionen wichtig, um Transaktionen rückgängig zu machen bzw. diese rückwärts abzuwickeln, um die Parteien im gleichen Zustand zurückzulassen, in dem sie vor dem Beginn der Transaktionen waren (wird aufgrunddessen, daß die Transaktion vollständig ist, oder nicht, als „Transaktionsprimitiv" bezeichnet). Abgebrochene bzw. unterbrochene (aborted) Transaktionen können von zurückgenommenen (canceled) Transaktionen dadurch unterschieden werden, daß unterbrochene Transaktionen durch ungeplante Ereignisse auftreten können, jedoch ist diese Unterscheidung hinsichtlich der Erfindung nicht wesentlich.

[0030] Zusätzlich zum Beenden der Sitzung durch Unterbrechen (abort) kann ein Client-Knoten eine Anfrage zurücknehmen (cancel), wodurch der Client dahingegend benachrichtigt wird, daß die Sitzung beendet ist. Wenn eine verbleibende Anfrage vorliegt (in der Warteschlange eingetragen ist), kann die Anfrage zurückgenommen werden, und eine zu diesem Zweck vorgesehene Nachricht kann zurück an den Server gesendet werden. Dies wird durch die DIALOG ABORT-Nachricht ausgeführt.

[0031] Das obengenannte System ist mit dem Tandem-TM/MP-Subsystem integriert (hinsichtlich einer Transaktionsüberwachungs-Einrichtung, einem proprietären Transaktions-Logmanager von Tandem), so daß jegliche anormale Beendigung des Dialogs die Transaktion beendet. Anormale Beendigungen können frei definiert werden und an jeder bestimmten Transaktionsbereichsabgrenzung auftreten. Eine Transaktionsbereichsabgrenzung (transaction demarcation boundary) kann als „Point of no return" für eine Transaktion betrachtet werden, bevor dem eine Transaktion rückgängig gemacht werden kann, jedoch nicht danach.

[0032] Im folgenden wird eine typische Belastungs-/Guthaben-Transaktion betrachtet, die in Fig. 2 dargestellt ist. In diesem Fall überweist ein Kunde Geld von einem Konto zu einem anderen, so daß ein Konto belastet wird (beispielsweise ein Sparkonto), während ein anderes Konto (beispielsweise ein Girokonto) gleichzeitig in der gleichen Transaktion eine Gutschrift erhält. Typischerweise muß ein bestimmter Client-Knoten auf einen bestimmten Server-Knoten zugreifen (der an einem bestimmten Ort, beispielsweise die regionale Zentrale einer Bank, vorgesehen sein kann), um eine solche Belastungs-/Guthaben-Transaktion ausführen. Selbstverständlich würde die TS/MP-Anwendung (nicht dargestellt) den Prozeß zwischen dem Client- und dem Server-Knoten **210** koordinieren. Wie oben bemerkt, wird durch die erste Nachricht **201** zwischen dem Client- und dem Server-Knoten **200** nicht nur eine Verbindung aufgebaut, indem ein bestimmter Client-Knoten identifiziert wird, der in die Transaktion eingebunden ist, sondern eine Nachricht selbst kann ferner Information aufweisen, die mit der Transaktion verbunden ist, beispielsweise welches Konto zu belasten ist, zum Beispiel das Sparkonto. Der Prozeß des Kombinierens von auf das Aufbauen einer Verbindung bezogener Information mit Information, welche die Transaktion und somit den Zweck der Verbindung betrifft, ist als Huckepack-Hinzufügen (piggy-backing) des Setups zu der ersten Sende-Anfrage bekannt und verringert die Setupzeit, insbesondere wenn zahlreiche solcher Transaktionsnachrichten vorliegen.

[0033] Eine Bestätigungsnachricht (Bezugszeichen **203**) wird von dem Server an den Client zurückgegeben (und umfaßt Fehlercode, hier FeContinue, um den Dialog fortzusetzen). Danach wird eine weitere Nachricht (**205**) von dem Client gesendet und von dem Server empfangen, welche die Identifikation und das Vorsehen einer Gutschrift auf einem zweiten Konto, beispielsweise dem Girokonto, betrifft. Nachdem diese Nachricht empfangen wurde, bestätigt eine Bestätigungsnachricht **207**, daß die Nachricht **205** empfangen wurde und weist gleichzeitig in der gleichen Nachricht den Client **200** an, die Sitzung zu beenden („tear down") (d.h. Abbrechen oder Zurücksetzen bzw. abort oder cancel) (mittels des damit einhergehenden Fehlercodes FeOK).

[0034] Durch Huckepack-Hinzufügen („piggy-backing") der Nachrichten wird eine effiziente Wirkungsweise hinsichtlich der Anzahl der gesendeten Nachrichten erzielt, wobei dies ein erheblicher Vorteil darstellt, wenn viele Tausende solcher Dialoge bearbeitet werden müssen. Oft kann das Huckepack-Hin-

zufügen von Nachrichten die Gesamtbetriebskosten des Systems zur Übertragung nahezu halbieren, da bei einer typischen Belastungs-/Guthaben-Transaktion nur wenige Nachrichten (in diesem Beispiel vier) gesendet werden.

[0035] Ein weiterer Aspekt der vorliegenden Erfindung ist, daß im Falle eines Fehlers beide Anwendungen am Client- und am Server-Knoten wissen, an welcher Stelle in der Sitzung der Fehler aufgetreten ist und die Sitzung rückgängig machen können (wenn der Fehler aufgetreten ist, bevor eine vorbestimmte Transaktionsbereichsgrenze übertreten wurde). Ferner überwachen eine Reihe von Segmenten des primären Speichers, die Dialogsteuerblöcke genannt werden, ebenfalls die mit den einzelnen Clients verknüpften Nachrichten.

[0036] Im weiteren ist die bevorzugte Harware- und Softwareimplementierung der vorliegenden Erfindung in der Fig. 3 dargestellt, und zeigt einen Client-Knoten **305** (workstation) mit Client-Anwendungen (**307**, **309**, **311**), eine Betriebssystem-Anwendung „Linkmon" **315**, die einen proprietären Tandem-Prozeß darstellt, der Teil des „Pathway"-TS/MP-Umgebung ist. Linkmon ist für alle ausgehenden (outbound) Ressourcen des Client-Knoten **305** verantwortlich, einschließlich der CPU-Ressourcen des Client-Knotens. Der Client-Knoten **305** und die Client-Anwendungen **307**, **309**, **311** sind über das IPC(Inter Processor Communication)-Protokoll mit dem Linkmon-Prozeß verbunden. Der Linkmon-Prozeß selbst ist mit der Vielzahl miteinander verbundener Server-Knoten **320**–**323** verbunden, sowie über das IPC-Protokoll auch alle darin vorliegenden Anwendungen (und deren Laufzeitbibliotheken bzw. run-time libraries). Der Linkmon-Prozeß ist diejenige Instanz, welche die „Bindings" (Verknüpfungen) zwischen den Client-Anwendungen und dem Server enthält. Der Linkmon kommuniziert ferner mit Serveranwendungen, welche die Bestätigungsnachrichten und andere hier beschriebene Nachrichten verarbeiten. Der Linkmon-Prozeß berücksichtigt alle derartigen Nachrichten, die von Client-Anwendungen und/oder Client-CPUs stammen, und sieht diese kaskadiert für die eine oder mehreren Server-Anwendungen und/oder Server-Knoten oder CPUs vor.

[0037] Im kontextfreien Fall würde der Linkmon-Prozeß eine einzelne Anfrage von einem Client an einen Server weiterleiten, die Antwort abwarten, die Antwort zurückleiten und jedes Mal nach dem Ende jeder Nachricht den Kontext unterbrechen (die Sitzung abbrechen). Im kontextabhängigen Fall der vorliegenden Erfindung muß der Linkmon-Prozeß ferner verantwortlich für das Beibehalten der temporären Verbindung zwischen dem Client und dem Server sein. Ferner verwendet das Verfahren der vorliegenden Erfindung den gleichen Routingmechanismus bzw. Weiterleitungsmechanismus (Pfade bzw. paths) für die kontextfreien Anfragen wie für die kontextabhängigen Anfragen, so daß das Verfahren der vorliegenden Erfindung rückwärtskompatibel zu älteren Tandem-Anwendungen ist, die nicht kontextabhängig vorgesehen sind. Daher verwendet die bevorzugte Ausführung der vorliegenden Erfindung die gleichen APIs (Anwendungsprogrammierschnittstellen) für sowohl kontextfreie als auch kontextabhängige Dialoge, wobei nur bestimmte Flags vorgesehen sind, um jegliche Unterschiede zwischen den Dialogen anzugeben.

[0038] Ein weiteres Beispiel (welches nicht dargestellt ist), in dem ein kontextabhängiger Pathsend der vorliegenden Erfindung vorteilhaft wäre, ist das Ergebnis einer SQL-Datenbankabfrage, bei der nicht klar ist, ob die Antwort auf eine Anfrage eine Zeile oder tausend Zeilen ausmachen würde. Wenn die Antwort tausend Zeilen betragen würde, könnten typischerweise wegen der Speicherbeschränkungen nicht alle tausend Zeilen auf einmal gesendet werden, und in einer kontextfreien Anfrage müßte die Verbindung zu der Untersprachen-Anwendung bzw. Sublanguage-Applikation, welche die SQL verarbeitet, wiederholt aufgebaut werden. Durch Aufbauen eines kontextabhängigen Pathsends wird jedoch der Aufwand zum Übertragen der Antworten (Zeilen) auf eine solche Anfrage, davon unabhängig, ob sie eine Zeile oder tausend Zeilen betragen würde, minimiert sein.

Detaillierte Implementierung einer bevorzugten Ausführung

[0039] In einer bevorzugten Ausführung wurden die folgenden Anwendungsprogrammierschnittstellen (Application Programmatic Interfaces, APIs) erfunden, um es dem Erzeugungsprozeß zu ermöglichen, einen Dialog zum Ausführen der Eingabe/Ausgabe bezüglich des Dialogs zu erzeugen, um auf die Transaktion bezogenen Kontext auf die Eingabe/Ausgabe innerhalb des Dialogs zu übertragen und den Dialog abzuschließen.

- SERVERCLASS_DIALOG_BEGIN_
- SERVERCLASS_DIALOG_SEND_
- SERVERCLASS_DIALOG_ABORT_
- SERVERCLASS_DIALOG_END_

[0040] Die folgenden APIs wurden erweitert, um es zu ermöglichen, daß die Dialoginformation von dem Empfangsprozeß abgerufen werden kann:

- FILE_GETRECEIVEINFO_

Verwendung:

- Der Erzeugungsprozeß stellt den SERVERCLASS_DIALOG_BEGIN_-Prozeduraufruf aus, wodurch ein spezieller Typ eines Empfangsprozesses angesprochen wird.

– Die Laufzeitumgebung startet einen Empfangsprozeß und erzeugt eine logische Verbindung zwischen den zwei kooperierenden Prozessen (dem Erzeugungsprozeß und dem Empfangsprozeß).
– Der Empfangsprozeß verwendet die bestehenden Dateimechanismen, die als „READUPDATE" bekannt sind, um die hereinkommende Nachricht zu empfangen, und ruft dann FILE_GETRECEIVEINFO auf, um zusätzliche Information über die gerade gelesenen Nachricht zu erlangen. Es wurde ein neues Feld erfunden, um es dem Empfangsprozeß zu ermöglichen, die Kontexteigenschaft der hereinkommenden Anfrage zu detektieren. Nach open receipt (offenem Empfang) und Detektion einer SERVERCLASS_DIALOG_BEGIN_-Nachricht sollte der Empfangsprozeß entweder mit einem FeOK- oder einem FeContinue-Rückgabecode antworten.
– Ein Rückgabecode FeOK veranlaßt die Laufzeitumgebung, die Verbindung zwischen dem Erzeugungsprozeß und dem Empfangsprozeß zu unterbrechen und der Erzeugungsprozeß ruft bei Detektion des FeOK-Rückgabecodes SERVERCLASS_DIALOG_END_ auf, um die Ressourcen in dem Adreßraum des Erzeugungsprozesses zu säubern (clean up).
– Ein Rückgabecode FeContinue veranlaßt die Laufzeitumgebung, die Verbindung zwischen dem Erzeugungsprozeß und dem Emfangsprozeß aufrechtzuerhalten, so daß darauffolgende SERVERCLASS_DIALOG_SEND_-Anfragen zu der gleichen Empfangsprozeßinstanz geleitet werden, die den Kontext beibehält, welcher den Dialog betrifft.
– Der Client kann jederzeit während des Dialogs eine SERVERCLASS_DIALOG_ABORT-Anfrage ausgeben, die eine neue Nachricht des Betriebssystems erzeugt, die als „DIALOG ABORT" bekannt ist. Die Nachricht wird an den Empfangsprozeß gesendet, so daß der Empfangsprozeß Dialoge säubern kann, die Ressourcen betreffen. Die Nachricht des Betriebssystems wird asynchron gesendet, wodurch der Erzeugungsprozeß die „DIALOG ABORT"-Nachricht parallel zu der Verarbeitung durch den Empfangsprozeß weiterverarbeiten kann.

[0041] Ferner wurde in der bevorzugten Ausführung das Konzept einer Dialog-Identifikation, Dialog-ID, erfunden, um die eindeutige Identifikation des Dialogs durch den erzeugenden Prozeß, der Laufzeitumgebung und des Empfangsprozesses zu ermöglichen (beispielsweise der Zustand des Client-Server-Verhältnisses). Da die Dialog-Identifikation zusammen mit allen Nachrichten bezüglich des Dialogs gesendet wird, verwendet dies der Empfangsprozeß als Schlüssel, wodurch die Notwendigkeit der Authentifizierung des Erzeugungsprozesses bezüglich aller Dialoge, bis auf die erste Eingabe/Ausgabe innerhalb des Dialogs, vermieden wird. Dialog-Identifikationen müssen nicht nur hinsichtlich des Authentifizierungsdienstes vorgesehen werden, sondern werden ferner von der Laufzeitumgebung für Hochleistungs-Zugriffe auf bestehende Datenstrukturen verwendet.

[0042] Im weiteren ist eine detaillierte Beschreibung der vorliegenden Erfindung vorgesehen, die ebenfalls in dem Anhang offenbart ist, wobei auf die Anwendungs-Programmierschnittstellen „DIALOG BEGIN", „DIALOG SEND", „DIALOG ABORT", Zurücksetzen (cancel) und Abbrechen (abort) der Übertragung bezuggenommen wird. Die ersten drei Anwendungs-Programmierschnittstellen (APIs) und das „Abbrechen der Übertragung" sind für die TS/MP(Pathway)-Komponente der vorliegenden Erfindung vorgesehen, während „Zurücksetzen" für das Dateisystem vorgesehen ist, welches von dem Guardian-Betriebssystem betrieben wird. Der Code für diese APIs würde in der Tandem-Systembibliothek vorgesehen werden. Wenn die Linkmon-Anwendung der vorliegenden Erfindung von dem Tandem-Pathway-Untersystem betrieben wird, wird ein zugehöriges Segment in dem primären Speicher (RAM) des Knotens, auf dem das Subsystem abläuft, derart eingerichtet, daß die Systembibliothek auf dieses zugreift. Die Fig. 4 zeigt ein solches Speichersegment, das Dialogsegment genannt wird, und das das Bezugszeichen **45** trägt, als Teil des RAMs **40**, und ist in vier Stücke unterteilt, die als Dialogkontrollblöcke (Dialog Control Blocks, DCB) bezeichnet werden. Jedesmal, wenn die Client-Anwendung eine der APIs des TS/MP (Pathway) aufruft, versetzt es dadurch die Dialogsteuerblöcke in jeden Dialogsegment in einen anderen Zustand und ruft somit eine Änderung hervor, welche diesen Zustand wiedergibt. Pro Sitzung gibt es ein Dialogsegment. Wenn daher beispielsweise ein Aufruf hinsichtlich DIALOG SEND ausgeführt wird, führt dies zur Einleitung einer Nachricht entweder durch die Server-Anwendung oder durch die Client-Anwendung, wobei dies durch ein Primitiv des Tandem-Betriebssystems ausgeführt wird, welches ein Puffer formatiert, beispielsweise das WriteRead-Tandem-Betriebssystem-Primitiv. Ein DIALOG BEGIN bedingt ein DIALOG SEND sowie die Bestätigung einer Nachricht, daß eine Sitzung gestartet wurde, wodurch vermieden wird, daß zwei in Konflikt stehende Sitzungen für eine einzelne Client-Anwendung aufgebaut werden. Ein Algorithmus verfolgt die Änderungen in dem Dialogsegment (Zustandsinformation für jede Sitzung) für jeden von der Client-Anwendung stammenden und an die APIs gerichteten Aufruf. Die Server-Anwendung verwendet Dateisystem-APIs, beispielsweise READUPDATE, FILE_GETREIVEINFO und REPLY. Somit wird der Zustand einer Client-Anwendung bewahrt, so daß bei einem unerwarteten Systemfehler die Ressourcen gesäubert (clean up) und wiederhergestellt werden können, und/oder die Übertragungen rückgängig ge-

macht werden können.

[0043] Ferner wurden Mechanismen erfunden, um noch ausstehende Eingaben/Ausgaben bezüglich des Dialogs falls notwendig zurücksetzen zu können.

[0044] Hinsichtlich der Vervollständigung der Nachrichten können andere Module überprüft werden. Dialogsteuerblöcke zum Überprüfen von Zuständen, die in Verbindung, und die mit den APIs in Netzen verwendet werden, sind per se bekannt und bilden nur eine mögliche detaillierte bzw. low-level Implementierung der vorliegenden Erfindung.

[0045] Daher ruft beispielhaft mit Hinblick auf Fig. 5 eine Client-Anwendung über Linkmon die Anwendungs-Programmierschnittstelle „DIALOG BEGIN" zuerst auf, welche eine „Beginne Übertragung" (Dialog begin)-Routine aufruft (Schritt **502**), einen Dialogsteuerblock (DCB) erzeugt (Schritt **504**), eine Dialog-Identifikation erzeugt (Schritt **506**), eine für den Linkmon-Prozeß vorgesehene Sende-Anfrage veranlaßt, indem der WriteRead-Befehl von Tandem verwendet wird (Schritt **508**), und den Zustand sowie die Zustandstabellen des Dialogsteuerblocks derart setzt, daß diese das Einleiten eines DIALOG BEGIN widerspiegeln. Daraufhin wird eine Datenstruktur der unteren Ebene (low-level) („T-Datei") in dem Prozeßdateisegment inkrementiert, welche Information enthält, die sich auf die geöffnete Übertragung bezieht (Schritt **510**). Sobald die Sitzung vorbei ist, kehrt die Datenstruktur der unteren Ebene in Schritt **510** zu ihrem ursprünglichen Zustand zurück und Speicherressourcen werden freigegeben und wieder hergestellt.

[0046] Bezugnehmend auf Fig. 6 ist die objektorientierte Darstellung des Dialogs zwischen Client und Server für eine bevorzugte Ausführung des kontextabhängigen Falls dargestellt. Objekte, die sich auf die in Fig. 5 dargestellte Schritte beziehen, werden von dem „Linkmon"-Abschnitt der Client-Anwendung erzeugt und gemanagt bzw. gehandhabt. Diese umfassen: ein „Sende offenen Steuerblock"-Objekt **602** (Send Open Control Block, SOCB); und ein „Anfrage"-Datenobjektsteuerblock bzw. REQCB(request control block)-Objekt **604**, die beide erzeugt werden, wenn ein Client zum ersten Mal versucht, über den Linkmon mit einem Server zu kommunizieren. Hinsichtlich der Server-Anwendung wird ein Objekt **606** erzeugt, das SCCB (Serverklasse-Steuerblock, Server Class Control Block) genannt wird, das sich auf die Serverklasse (SC) bezieht (ein solches Objekt pro Serverklasse). Ein Verknüpfungszugriff-Steuerblock (Link Access Control Block, LACB) **608** (Darstellung des Öffnens eines Server-Prozesses) kennzeichnet den Zugriffspfad auf einen Server-Prozeß eindeutig. Schließlich überwacht ein Servermonitor-Steuerblock (server monitor control block, SMCB) **610** die anderen Serverobjekte.

[0047] Die Steuerblöcke **606**, **608** und **610** erzeugen Verknüpfungen (Bindungen) untereinander (symbolisch als Verknüpfungen **601** dargestellt), wenn die Server-Anwendung gestartet wird. Wenn Client-Applikationen Kontakt zu einem Server (über Linkmon) herstellen und DIALOG BEGIN aufgerufen wird (wie oben bemerkt wird dem DIALOG BEGIN ferner ein DIALOG SEND per Huckepack hinzugefügt), werden die Steuerblöcke **602**, **604** erzeugt und die Steuerblöcke **602**, **604** werden miteinander verknüpft (wie symbolisch durch Verknüpfung **603** dargestellt ist). Wenn Clients Anfragen (über Linkmon) ausstellen und DIALOG SEND aufgerufen wird, wird der REQCB-Block **604** mit den Serverklassen-Steuerblöcken **606**, **608** und **610** verknüpft, wie symbolisch durch die Verknüpfungen **605** dargestellt ist.

[0048] Nachdem die Verknüpfungen **601**, **603** und **605** hergestellt wurden, wird die Anfrage selbst vom Client durch den Linkmon an den Server-Prozeß weitergeleitet. Die Wichtigkeit des Herstellens von Verknüpfungen liegt darin, daß dadurch ein Kommunikationspfad zu einer Server-Prozeßinstanz erstellt wird, so daß der Pfad zur Verwendung leicht identifizierbar ist. Das Vorliegen von Verknüpfungen zwischen den Objekten bedeutet, daß die Objekte alle leicht zu identifizieren sind, indem die Verknüpfungen durchlaufen werden.

[0049] Im kontextfreien Fall würden die Verknüpfungen **603**, **605** zu dem REQCB-Anfragesteuerblock jedesmal unterbrochen werden, wenn eine Antwort vom Server zurück an den Client gesendet wird, und daher müssen diese Verknüpfungen jedesmal neu erstellt werden, wenn eine Nachricht gesendet wurde, wodurch Zeit und Ressourcen verschwendet werden. Jedoch findet dies bei kontextabhängigen Nachrichten wegen des Vorliegens eines weiteren Objekts, des Dialogkontrollblocks (DCB) **650**, nicht statt. Der DCB **650** ist ferner dafür verantwortlich, eindeutige Dateinummern und/oder Dialog-Identifikationen zu speichern, die eine Sitzung gegenüber dem Server repräsentieren.

[0050] Wenn in der vorliegenden Erfindung eine Nachricht empfangen wird, fragt der Linkmon einen Flag in FILE_GETRECEIVEINFO ab, der von dem Schreib-/Lese-Betriebssystem bzw. Write Read OS gehandhabt wird, um zu ermitteln, ob die Nachricht kontextabhängig ist. Wenn die Nachricht nicht kontextabhängig ist, kann die vorliegende Erfindung die Nachricht in der üblichen kontextfreien Art und Weise bearbeiten (d.h. die vorliegende Erfindung ermöglicht die Rückwärtskompatibilität mit bestehender Software). Das Flag in FILE_GETRECEIVEINFO enthält ferner Information darüber, ob die Nachricht vom Typ DIALOG BEGIN, DIALOG SEND, oder DIALOG ABORT ist.

[0051] Wenn jedoch der Flag in

FILE_GETRECEIVEINFO angibt, daß die Nachricht kontextabhängig ist, erzeugt der Linkmon ferner das Dialogsteuerungsblock-Objekt (Dialog Control Block, DCB) **650**, der Information über das genaue Wesen der Dialogbeziehung enthält, die zwischen dem Client und dem Server besteht, und verknüpft den DCB **650** mit dem SOCB-Block **602** und dem LACB-Block **608**, wie durch die symbolischen Verknüpfungen **607** in Fig. 6 dargestellt ist. Wenn diese zwei Strukturen, SOCB **602** und LACB **608**, durch die Verknüpfungen **607** und den Dialogsteuerblock **650** verknüpft sind, bedeutet dies für die Server-Anwendung, daß eine kontextabhänigge Pathsend-Sitzung vorliegt. Unter der Annahme, daß kein ABORT oder CANCEL besteht, liegt in einem kontextfreien Fall kein Multiplexen von Dialogen vor, und es wird eine Art zeitlich zugeordnete Kommunikation zwischen dem Client und dem Server aufgebaut. Zudem weiß der Server, daß nicht mehr als eine Sitzung zu öffnen ist, indem das Vorliegen einer Verknüpfung zwischen dem SOCB (Sende offenen Steuerblock, Send Open Control Block) und dem LACB (Verknüpfungszugriff-Steuerblock, Link Access Control Block) überprüft wird. Das gesamte Obenstehende ist die Reaktion auf ein Dialog Begin- oder Dialog Send-Signal und die Reaktion auf das WriteRead-Betriebssystem-Primitiv.

[0052] Wie bereits bemerkt, ist der erste Teil des Protokolls die Reaktion eines Servers auf ein Dialog Begin oder -Send. Meistens ist die Reaktion des Servers „FeContinue" (wodurch eine Sitzung beibehalten wird). Wenn ein FeContinue gesendet wird, werden nur Verknüpfungen **605** zwischen dem aktuellen Anfragesteuerblock REQCB **604** und den Steuerblöcken in der Server-Anwendung unterbrochen (aufgehoben bzw. dealloziert) und jeglicher betroffener Speicher wird freigestellt. Jedoch werden weder die Verknüpfungen **607** unterbrochen, noch wird der Dialogsteuerblock **650** zerstört (aufgehoben), der die Information enthält, welche zum Beibehalten der zeitlich zugewiesenen Kommunikation zwischen Server und Client wichtig ist. Das heißt, daß ein Rückgabecode FeContinue die Laufzeitumgebung dazu veranlaßt, die Verbindung zwischen dem Ursprungsprozeß und dem Empfangsprozeß beizubehalten, so daß die darauffolgenden SERVERCLASS_DIALOG_SEND-Anfragen in einer schnellen und effizienten Weise an die gleichen Empfangsprozeß-Instanz weitergeleitet werden, die den dialogbetreffenden Kontext beibehält.

[0053] In gleicher Weise würde in Reaktion auf eine FeContinue-Nachricht nach einem Dialog-Begin der Client ein Dialog-Send und die zugeordnete Nachricht an den Linkmon weitergeben. Linkmon würde die empfangene Dialog-Identifikation suchen, um diese mit einem bestimmten Dialog und Server zu assoziieren (die Identifikationen aller aktiven Dialoge liegen in dem Speicher in dem RAM vor). Daraufhin würde ein Anfragesteuerblock REQCB, beispielsweise Block **604**, gebildet (alloziert) werden und die Verknüpfungen **605** für diese bestimmte Nachricht würden aufgebaut werden.

[0054] Wenn und falls jedoch eine FeOK-Fehlercode-Nachricht von dem Server ausgestellt wird und von dem Client (über Linkmon) empfangen wird, werden nicht nur die Verknüpfungen (oder Verbindungen) **603** und **605** unterbrochen, sondern es werden ferner Verknüpfungen **607** unterbrochen, und zudem wird der Dialogsteuerblock aufgehoben und der LACB wird als zur Verfügung stehend markiert. Dadurch wird der kontextabhängige Dialog effektiv aufgelöst (kill).

[0055] Zuletzt verursacht ein DIALOG END, daß der Linkmon und das Betriebssystem die Systemressourcen säubern (clean up). Ein DIALOG END kann auf eine FeOK-Nachricht folgen.

[0056] In dem obengenannten Prozeß muß der Server selbstverständlich verschiedene Sitzungen verfolgen und muß über diese informiert sein, die mit verschiedenen Clients offen sind, und jeder Sitzung eindeutige Dialogsteuerblöcke zuordnen. Dies ist insbesondere der Fall, wenn der Server mit mehreren Verarbeitungslinien arbeitet (multi-threaded). Tatsächlich wird das automatisch von dem Guardian-Betriebssystem von Tandem ausgeführt, indem Segmente im RAM der DCBs jedesmal verwendet werden, wenn eine Datei geöffnet wird, und Information in Form von Flags, die sich auf das obengenannte beziehen, werden von dem Linkmon zusammen mit Nachrichten von dem Server weitergegeben. Ferner würden Identifikationen (IDs) in dem ersten Dialog-Begin und in der Bestätigung zwischen dem Client und dem Server ausgetauscht werden. Die Eindeutigkeit der Identifikationen wird von dem Linkmon bestimmt, der mit dem Betriebssystem arbeitet (das ferner Dateienummer-Identifikationen ermittelt, die sich ebenfalls auf die Nachrichten beziehen). In der bevorzugten Ausführung ist das Betriebssystem das Guardian Tandem-Betriebssystem (Non-Stop Kernel OS, Version B30.02).

[0057] Dadurch ist die Server-Anwendung durch das Überprüfen des Vorliegens von Verknüpfungen **607** und durch Durchlaufen der Verknüpfungen **607**, um die in dem Dialogsteuerblock **650** vorliegende Information zu lesen, darüber informiert, ob die nächste Nachricht nach einem kontextabhängigen „Send" von einer bestimmten bekannten kontextabhängigen Pathsend-Client-Anwendung stammt (und daß höchstwahrscheinlich die Nachricht entweder eine DIALOG SEND- oder DIALOG ABORT-Nachricht ist, insbesondere, wenn es sich um die Übertragung handelt, die mit Bezug auf die Fig. 2 oben beschrieben ist). Der Server ist ferner darüber informiert, daß die Nachricht nicht von einer anderen Client-Anwendung oder einer nicht kontextabhängigen Pathsend-An-

wendung stammt. Dadurch werden in Kombination mit dem Huckepack-Hinzufügen von Daten eine oder mehrere zusätzliche Einheiten von Rechenzeit und Systemressourcen eingespart.

[0058] In der vorliegenden Erfindung muß bei der Anwendung auf OLTP-Transaktionen jede Transaktion unteilbar (atomic) sein, wobei dadurch das Tandem-Betriebssystem, welches die vorliegende Erfindung implementiert, gewährleistet, daß jede Operation nach BEGIN TRANSACTION scheitert, wenn das Ende der Transaktion nicht erfolgreich abgeschlossen wurde.

[0059] Als weiteres Beispiel (nicht dargestellt) soll angenommen werden, daß die SQL-Anfrage, als Antwort Tausende von Zeilen auf die SQL-Anfrage zurückgeben kann. Da typischerweise nur 32 k Daten gesendet werden können (wobei dies lediglich 100 Zeilen sein können), sind mehrere Send-Operationen erforderlich, um die folgenden Zeilen zu senden. Jedoch werden diese darauffolgenden Zeilen an die gesamten Serverklassen gesendet, und der erste Server, der die Zeilen empfängt, wird zweifellos nicht der richtige Server sein. Die Lösung zu diesem Problem ist das Erstellen eines kontextabhängigen Pathsends, analog zu den Belastungs-/Guthaben-Transaktionen, in denen derselbe Server gewählt werden muß. Daher gäbe es keine Notwendigkeit, wie sie bei vorheriger Technik besteht, eine Warteschlange von Servern in einer Serverklasse abzufragen. Sobald eine kontextabhängige Sitzung (Dialog) aufgebaut wurde und die Nachrichten huckepack-hinzugefügt wurden, wie oben beschrieben ist, wird die Bestätigung der Nachricht mit FeContinue durch den Server einem bestimmten Client fest zugeordnet, in der Form einer zeitlich zugeordneten oder pseudo-synchronen Kommunikation. In gleicher Weise können huckepack-hinzugefügte Kommunikationen verwendet werden, um den Dialog abzubrechen bzw. aufzulösen („kill").

[0060] Ferner ist dem Fachmann ersichtlich, daß zum Umsetzen der vorliegenden Erfindung jegliche Softwarebefehle verwendet werden können, um die oben beschriebene Erfindung zu implementieren, d.h. Code, Daten, Datenstrukturen und jegliche Computerhardware, die die gleiche Form als Gesamtheit umfaßt, in Kombination mit anderen ähnlichen Strukturen in einem Netzsystem.

[0061] Die oben genannte Beschreibung dient nur darstellerischen Zwecken. Dem Fachmann sind zahlreiche Modifikationen und Änderungen leicht ersichtlich, die in den Umfang der Erfindung fallen, der wie folgt beansprucht ist.

Patentansprüche

1. Verfahren zum Senden von Information in einem Server-Client-Netz zwischen einem Client und einem Server, wobei das Verfahren die folgenden Schritte umfaßt, welche von einem Datenverarbeitungssystem ausgeführt werden:
Senden einer anfänglichen Anfrage durch den Client, die Daten, welche sich auf die anfängliche Anfrage beziehen, sowie Daten umfaßt, die sich auf den Zustand einer Client-Server-Beziehung beziehen;
Aufbauen eines Dialogs zwischen dem Client und dem Server mittels eines Betriebssystems des Datenverarbeitungssystems, in Reaktion auf die anfängliche Anfrage;
Senden einer Antwort auf die anfängliche Anfrage durch den Server, wobei die Antwort Daten, die sich auf die anfängliche Anfrage beziehen, sowie einen Dialogzustand umfaßt, der sich auf den Zustand der Client-Server-Beziehung bezieht;
Senden, durch den Client, einer zweiten Anfrage gemäß eines Werts des Dialogzustands; und
Senden einer zweiten Antwort auf die zweite Anfrage, wobei die zweite Anfrage den Dialogzustand sowie Information umfaßt, die kontextabhängig von der Information in der Antwort ist.

2. Verfahren nach Anspruch 1, wobei der Schritt des Aufbauens eines Dialogs die Schritte des Einrichtens einer Dialog-Identifikation umfaßt.

3. Verfahren nach Anspruch 2, wobei die zweite Anfrage die Dialog-Identifikation umfaßt.

4. Verfahren nach Anspruch 2, wobei der Schritt des Einrichtens einer Dialog-Identifikation die Schritte des Zuordnens der Dialog-Identifikation zu der anfänglichen Anfrage und des Sendens der anfänglichen Anfrage an den Server umfaßt.

5. Verfahren nach einem der vorangegangenen Ansprüche, wobei der Dialogzustand einen der folgenden Werte hat: FeOK, der angibt, daß der Dialog beendet werden sollte, und FeContinue, der angibt, daß der Client eine weitere Anfrage senden sollte.

6. Verfahren nach Anspruch 5, wobei der Schritt des Sendens einer zweiten Anfrage den Schritt des Sendens der zweiten Anfrage durch den Client umfaßt, wenn der Wert des Dialogzustands FeContinue ist, wodurch angezeigt wird, daß der Server weitere zu sendende Information hat.

7. Verfahren nach einem der vorangegangenen Ansprüche, das ferner den Schritt des Sendens einer zweiten Antwort durch den Server umfaßt, die Information enthält, welche sich logisch auf Information in der ersten Antwort bezieht.

8. Verfahren nach einem der vorangegangenen Ansprüche, das ferner den Schritt umfaßt: Senden einer Abbruchnachricht, die den Dialog beendet, durch entweder den Client, den Server oder das Betriebs-

system.

9. Verfahren nach einem der vorangegangenen Ansprüche, wobei der Schritt des Aufbauens eines Dialogs den Schritt des Einrichtens eines Dialogssteuerungsblocks umfaßt, welcher eine Dialog-Identifikation enthält.

10. Verfahren nach einem der vorangegangenen Ansprüche, wobei die anfängliche Anfrage ein Feld umfaßt, das angibt, daß ein kontextabhängiger Dialog aufgebaut werden sollte.

11. Verfahren nach einem der vorangegangenen Ansprüche, das ferner den Schritt umfaßt: Senden einer Dialogbeendigungsnachricht durch den Client, wenn der Dialogzustand FeOK ist.

12. Verfahren nach Anspruch 1, wobei der Client und der Server aus einer jeweiligen Vielzahl von Client-Knoten und Server-Knoten stammen, die ein Server-Client-Knotennetz umfassen, wobei ein kontextabhängiger Pfad erzeugt wird, um den Aufwand des Aufrechterhaltens eines Dialogs zwischen dem Server und dem Client zu minimieren, wobei das Verfahren umfaßt:
Senden von Client-Nachrichten ausgehend von einer Vielzahl von Client-Knoten, wobei jede Client-Nachricht spezielle Client-Anfragen an einen bestimmten Server-Knoten aufweist;
Huckepack-Hinzufügen von Daten auf die von der Vielzahl von Client-Knoten stammenden Client-Nachrichten, wobei die huckepack-hinzugefügten Daten sowohl die Daten, welche sich auf die bestimmten Client-Anfragen beziehen, als auch Daten umfaßt, die sich auf den Zustand der Client-Server-Beziehung beziehen;
Antworten auf die huckepack-hinzugefügten Daten, indem eine Vielzahl von Servernachrichten an die Vielzahl von Clients gesendet wird; und
Huckepack-Hinzufügen von Daten auf die Servernachrichten, wobei die huckepack-hinzugefügten Daten sowohl Daten, welche sich auf die bestimmten Client-Anfragen beziehen, als auch Daten umfaßt, die sich auf den Zustand der Client-Server-Beziehung beziehen,
wobei die huckepack-zugefügten Daten den Aufwand des Aufrechterhaltens eines Dialogs zwischen den Clients und den Server-Knoten verringern können.

13. Verfahren nach Anspruch 12, wobei eine vorübergehend zugewiesene Kommunikation zwischen zumindest einem der Vielzahl von Client-Knoten und dem Server-Knoten erzeugt wird, wobei die zugewiesene Kommunikation von einer Vielzahl von Zuständen zwischen dem Client-Knoten und dem Server-Knoten definiert wird und diese erzeugt, wobei die Zustände von einer Betriebssystem-Prozeßanwendung überwacht werden, die für den Client-Knoten und den Server-Knoten gemeinsam ist, wobei die Betriebssystem-Prozeßanwendung in einem Betriebssystem-Prozeßknoten mit einem primären Speicher vorgesehen ist, und das Verfahren ferner die Schritte umfaßt:
Erzeugen einer Vielzahl von objektorientierten Datenstrukturen in dem primären Speicher des Betriebssystem-Prozeßknotens; und
Verknüpfen der Vielzahl von Strukturen in einer vorbestimmten Weise, die davon abhängt, welche Zustände zwischen dem Client-Knoten und dem Server-Knoten vorliegen,
wobei der Zustand der zugewiesenen Kommunikation nachgeprüft werden kann und somit die einzelnen Client- und Server-Knoten nachgeprüft werden können, indem die Verknüpfungen der Vielzahl von Datenstrukturen durchlaufen werden.

14. Verfahren nach Anspruch 12 oder 13, wobei der Schritt des Huckepack-Hinzufügens von Daten den Schritt des Erstellens einer Dialog-Identifikation umfaßt.

15. Verfahren nach einem der Ansprüche 12 oder 13, wobei der Schritt des Huckepack-Hinzufügens von Daten den Schritt des Erstellens eines Dialogzustands umfaßt.

16. Verfahren nach Anspruch 15, wobei der Dialogzustand einen der folgenden Werte aufweist:
FeOK, der angibt, daß der Dialog beendet werden sollte, und
FeContinue, der angibt, daß der Client eine weitere Anfrage senden sollte.

17. Verfahren nach einem der Ansprüche 12 bis 16, das ferner den Schritt des Sendens einer den Dialog beendende Beendigungsnachricht, durch entweder den Client, durch den Server oder durch das Betriebssystem umfaßt.

18. Verfahren nach Anspruch 12 oder 13, wobei der Schritt des Huckepack-Hinzufügens von Daten den Schritt des Einrichtens eines Dialogsteuerungsblocks umfaßt, der eine Dialog-Identifikation umfaßt.

19. System in einem Server-Client-Netz, umfassend:
einen Client und zumindest einen Server, wobei der Client umfaßt:
Mittel zum Senden einer anfänglichen Anfrage, die Daten, welche sich auf die anfängliche Anfrage beziehen, und Daten umfaßt, die sich auf den Zustand einer Client-Server-Beziehung beziehen;
Mittel zum Senden einer zweiten Anfrage gemäß eines ersten Werts eines Dialogzustands der Client-Server-Beziehung; und
Mittel zum Senden einer Dialog-Beendigungsnachricht gemäß eines zweiten Werts des Dialogzustands, wobei der Server umfaßt:

Mittel zum Senden einer ersten Antwort auf die anfängliche Anfrage, wobei die erste Antwort Daten, welche sich auf die anfängliche Anfrage beziehen, und einen Dialogzustand umfaßt, der den ersten Wert aufweist; und
Mittel zum Senden einer zweiten Antwort auf die zweite Anfrage, wobei die zweite Anfrage Information, die kontextabhängig von Information in der ersten Nachricht ist, sowie den Dialogzustand umfaßt, der den zweiten Wert aufweist;
Mittel zum Aufbauen eines Dialogs zwischen dem Client und dem Server in Reaktion auf die anfängliche Anfrage.

20. System nach Anspruch 19, wobei die jeweiligen ersten und zweiten Dialogzustandswerte sind: FeContinue und FeOK, die angeben, daß der Client eine weitere Anfrage senden sollte beziehungsweise daß der Dialog beendet werden sollte.

21. System nach Anspruch 19 oder 20, das umfaßt:
einen ersten Sendeschaltkreis, der vorgesehen ist, die anfängliche Anfrage an den Server zu senden;
einen Empfängerschaltkreis, der vorgesehen ist, den Dialogzustand von dem Client zu empfangen; und
einen zweiten Sendeschaltkreis, der vorgesehen ist, gemäß dem Dialogzustand an den Client entweder 1) eine zweite Anfrage an den Server zu senden, wobei die zweite Anfrage logisch mit der ersten Anfrage verbunden ist, oder 2) eine Dialogbeendigungsnachricht zu senden.

22. System nach Anspruch 19, 20 oder 21, wobei der Server umfaßt:
einen ersten Empfangsschaltkreis, der vorgesehen ist, die anfängliche Anfrage von dem Client zu empfangen;
einen ersten Sendeschaltkreis, der vorgesehen ist, in Reaktion auf die anfängliche Anfrage erste Daten an den Client zu senden, wobei die ersten Daten den Dialogzustandswert umfassen, der angibt, daß der Server in Reaktion auf die anfängliche Anfrage weitere Daten zu senden hat;
einen zweiten Empfangsschaltkreis, der vorgesehen ist, die zweite Anfrage von dem Client zu empfangen; und
einen zweiten Sendeschaltkreis, der vorgesehen ist, in Reaktion auf die zweite Anfrage zweite Daten an den Empfänger zu senden, wobei die zweiten Daten einen Dialogzustandswert umfassen, der angibt, daß der Server das Senden seiner Daten abgeschlossen hat.

23. System nach Anspruch 22, wobei der zweite Sendeschaltkreis einen Sendeschaltkreis umfaßt, der vorgesehen ist, in Reaktion auf die zweite Anfrage die zweiten Daten an den Client zu senden, wobei die ersten und zweiten Daten logisch in Beziehung sind.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen

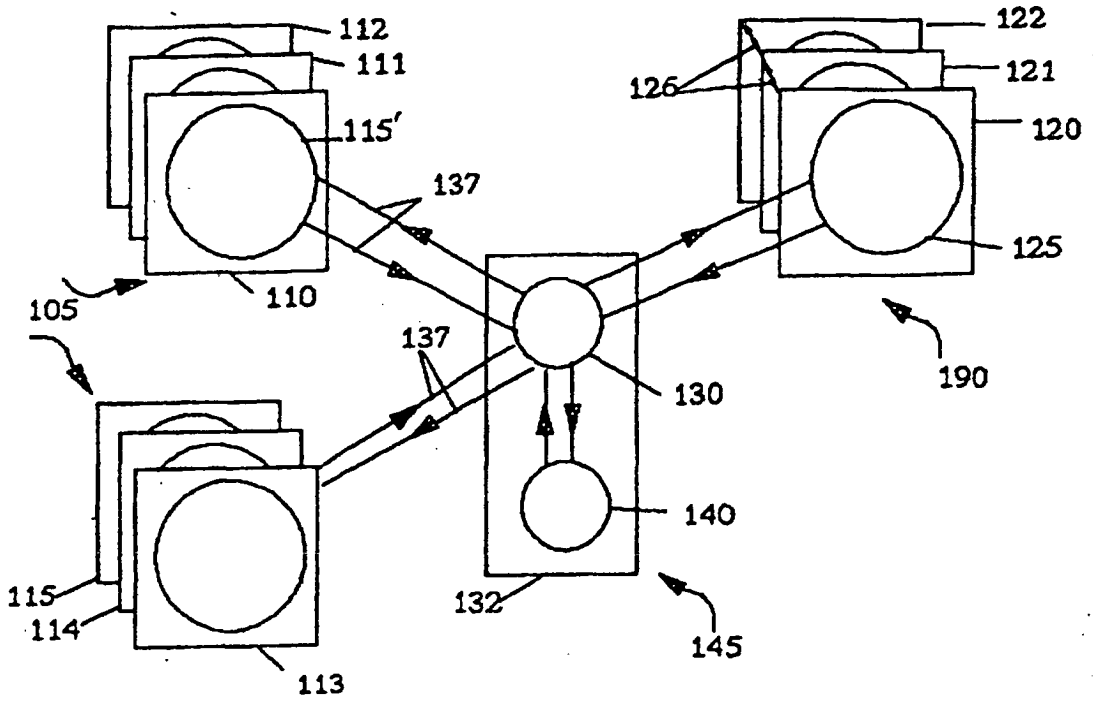


Fig. 1

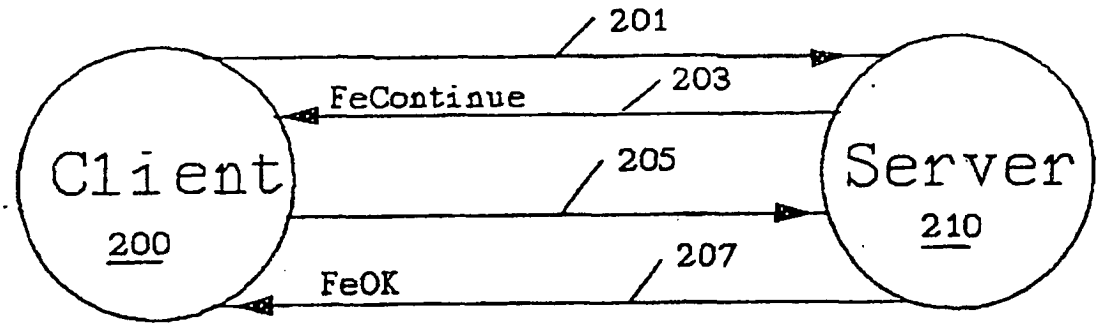


Fig. 2

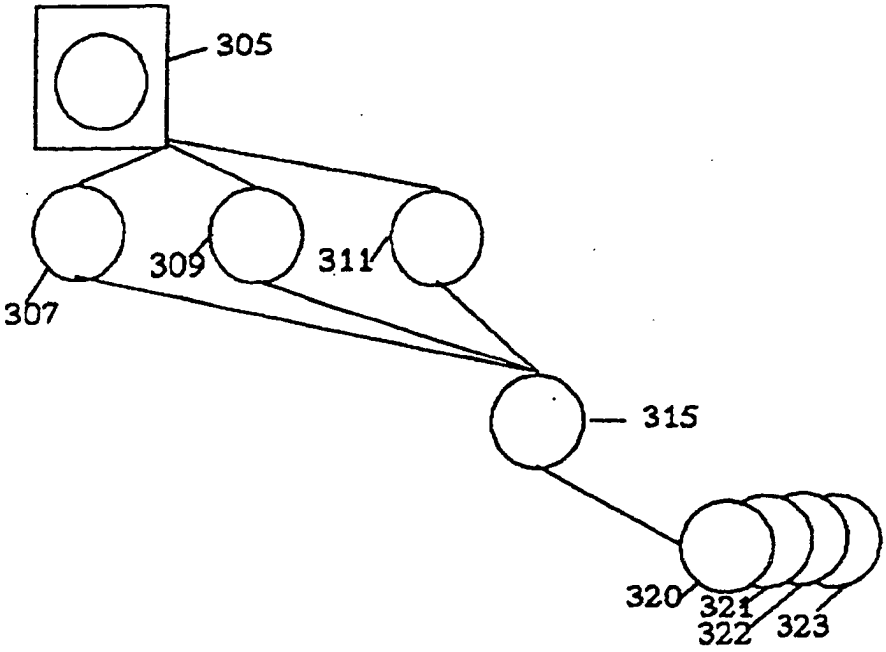


Fig. 3

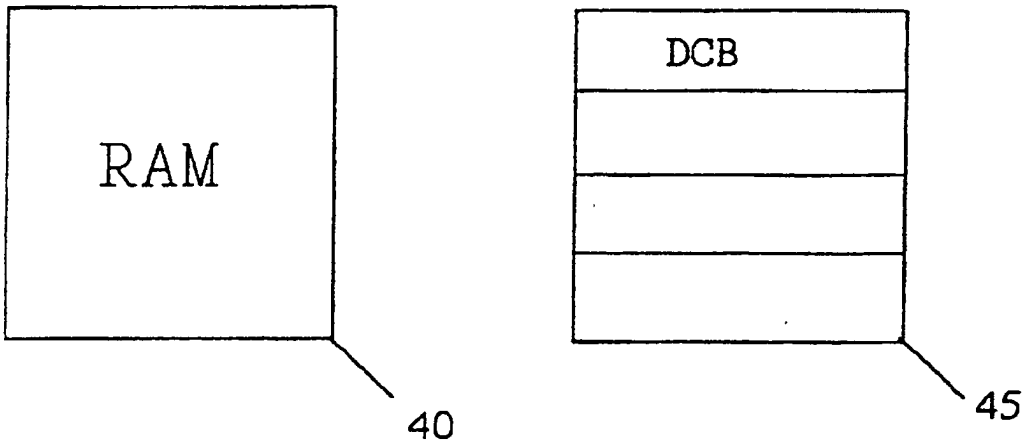


Fig. 4

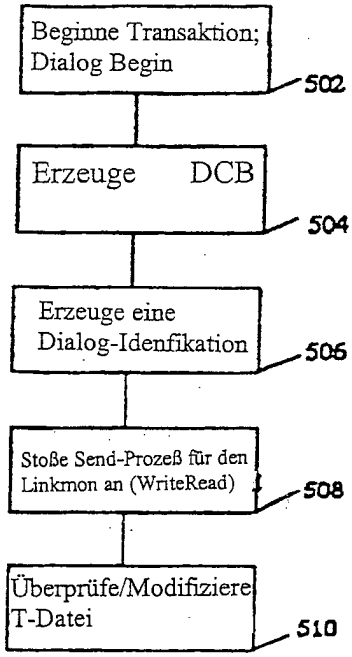

Beginne Transaktion;
Dialog Begin
502
Erzeuge DCB
504
Erzeuge eine
Dialog-Idenfikation
506
Stoße Send-Prozeß für den
Linkmon an (WriteRead)
508
Überprüfe/Modifiziere
T-Datei
510

Fig. 5

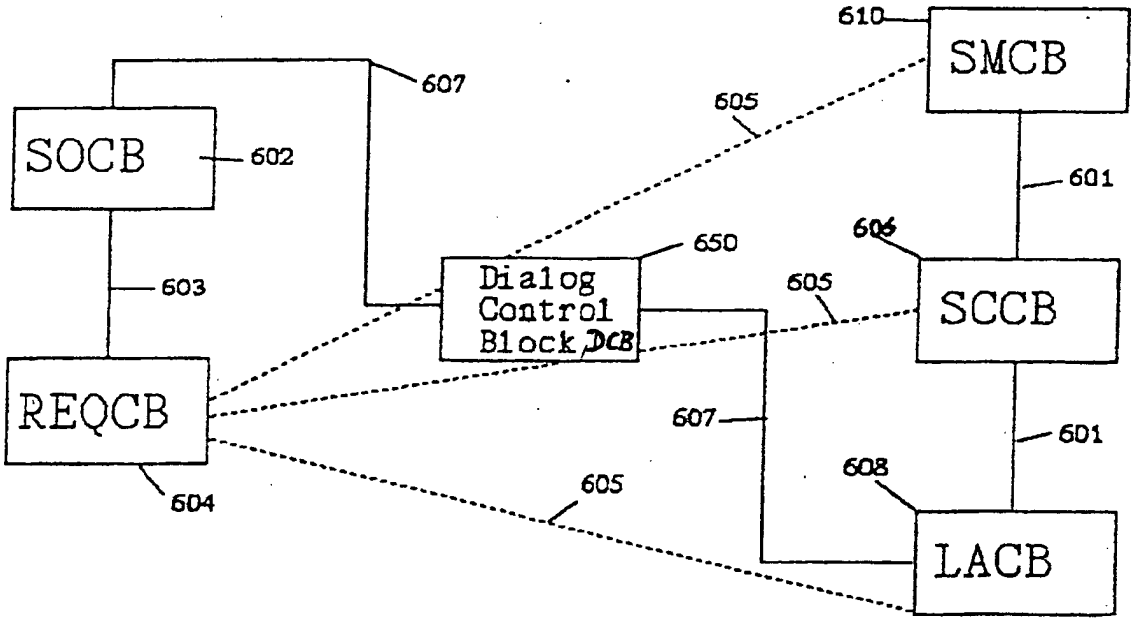

610
SMCB
607
605
SOCB
602
601
606
650
603
Dialog
Control
Block DCB
605
SCCB
REQCB
607
601
608
605
604
LACB

Fig. 6