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

(10) **DE 695 32 616 T2** 2005.01.05

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 989 489 B1**
(21) Deutsches Aktenzeichen: **695 32 616.3**
(96) Europäisches Aktenzeichen: **99 125 601.7**
(96) Europäischer Anmeldetag: **19.12.1995**
(97) Erstveröffentlichung durch das EPA: **29.03.2000**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **25.02.2004**
(47) Veröffentlichungstag im Patentblatt: **05.01.2005**

(51) Int Cl.[7]: **G06F 9/455**
G06F 9/44

(30) Unionspriorität:
359884 **20.12.1994** **US**

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

(73) Patentinhaber:
Sun Microsystems, Inc., Mountain View, Calif., US

(74) Vertreter:
Sparing · Röhl · Henseler, 40237 Düsseldorf

(72) Erfinder:
Gosling, James A., Woodside, California 94062, US

(54) Bezeichnung: **Plattformunabhängiges Verfahren und System zum Laden eines Objekts und Objektdarstellers**

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

1. Gebiet der Erfindung

[0001] Die vorliegende Erfindung betrifft allgemein die Verwendung von Computersoftware auf Mehrfach-Computer-Plattformen, die unterschiedliche zugrundeliegende Maschinenbefehlssätze verwenden, und insbesondere ein Verfahren zum Verifizieren der Integrität der Computersoftware, die von einem Netzserver oder einer anderen Quelle erhalten wird.

2. Stand der Technik

[0002] JP-A-63282867 beschreibt ein Datendatei-Übertragungssystem, bei dem die Datentypinformationen zusammen mit einer Datendatei übertragen werden, um der Client-Software zu ermöglichen, die Daten in der Datendatei unter Verwendung dieser Datentypinformationen anzuzeigen.

[0003] Wie in **Fig.** 1 allgemein gezeigt, kann in einem typischen vernetzten Computersystem **100** des Standes der Technik ein erster Computer **102** ein Computerprogramm **103** herunterladen, das sich auf einem zweiten Computer **104** befindet. In diesem Beispiel ist der erste Anwenderknoten **102** typischerweise eine Anwender-Workstation mit einer Zentraleinheit **106**, einer Anwenderschnittstelle **108**, einem Primärspeicher **110** (z. B. einem Schreib/Lese-Speicher) für die Programmausführung, einem Sekundärspeicher **112** (z. B. einer Festplatte) zum Speichern eines Betriebssystems **113**, von Programmen, Dokumenten und anderen Daten, sowie einem Modem oder einer anderen Kommunikationsschnittstelle **114** zum Verbinden mit einem Computernetz **120**, wie z. B. dem Internet, einem lokalen Netz oder einem Weitverkehrsnetz. Die Computer **102** und **104** werden häufig als "Knoten im Netz" oder "Netzknoten" bezeichnet.

[0004] Der zweite Computer **104** ist häufig ein Netzserver, kann jedoch eine zweite Anwender-Workstation sein, und enthält typischerweise die gleiche grundsätzliche Anordnung von Computerkomponenten wie der erste Computer.

[0005] Nachdem im Stand der Technik der erste Computer **102** eine Kopie eines Computerprogramms **103** vom zweiten Computer **104** heruntergeladen hat, stehen im Wesentlichen keine genormten Werkzeuge zur Verfügung, um den Anwender des ersten Computers **102** zu unterstützen, die Integrität der heruntergeladenen Programms **103** zu verifizieren. Insbesondere solange der Anwender des ersten Computers nicht den Quellcode des heruntergeladenen Programms untersucht, ist es praktisch unmöglich, Werkzeuge des Standes der Technik zu verwenden, um zu ermitteln, ob das heruntergeladene Programm **103** einen Stapelunterlauf oder einen Stapelüberlauf hervorruft, oder ob das heruntergeladene Programm **103** Dateien und andere Betriebsmittel auf dem Computer des Anwender beschädigt.

[0006] Ein zweites Problem bezüglich des Herunterladens von Computersoftware von einem Computer auf einen weiteren betrifft die Übertragung von Computersoftware zwischen Computerplattformen, die unterschiedliche zugrundeliegende Maschinenbefehlssätze verwenden. Es gibt einige Beispiele des Standes der Technik von plattformunabhängigen Computerprogrammen und von plattformunabhängigen Computerprogrammiersprachen. Was im Stand der Technik fehlt sind zuverlässige und automatisierte Softwareverifikationswerkzeuge, um den Empfängern solcher Software zu ermöglichen, die Integrität der übertragenen plattformunabhängigen Computersoftware zu verifizieren, die von einem Netzserver oder einer anderen Quelle erhalten wird.

[0007] Ein weiterer Aspekt der vorliegenden Erfindung betrifft Verfahren zum automatischen Herunterladen von Software, die einem Objekt oder einer Datei zugeordnet ist, nachdem ein Anwender ein Objekt oder eine Datei zum Herunterladen von einem entfernten Ort ausgewählt hat. Zum Beispiel gibt es das weit verbreitete Merkmal des Internets, das bekannt ist als das "World Wide Web" (WWW). Wenn ein Dokument auf dem World Wide Web (WWW) des Internets betrachtet wird, kann eine Seite des Dokuments Referenzen auf andere Dokumente oder Objekte enthalten. Ein Anwender kann auf solche anderen Dokumente oder Objekte zugreifen, indem er ein gegebenes Objekt über einen zugeordneten Hyperlink (Hyperverknüpfung) auswählt. Ein solches Auswählen wird üblicherweise von einem Anwender durchgeführt in Verbindung mit einer graphischen Anwenderschnittstelle auf einem Workstation-Knoten durch Niederdrücken eines Knopfes auf einer Zeigevorrichtung, während die Zeigevorrichtung auf ein graphisches Abbild zeigt, das die Hyperlink-Auswahl darstellt. In Reaktion auf die Auswahl eines Hyperlinks öffnet anschließend das Netzzugriffsprogramm des Anwenders eine Ver-

bindung zu dem Server, auf dem das bezeichnete Dokument des Objekts liegt (angegeben durch Daten, die im Hyperlink oder im Dokument oder Objekt eingebettet sind, das derzeit betrachtet wird), und lädt das bezeichnete Dokument oder das Objekt herunter. Wenn jedoch das heruntergeladene Dokument oder Objekt einen für das Netzzugriffsprogramm des Anwenders unbekannten Datentyp aufweist, ist der Anwender nicht fähig, das heruntergeladene Dokument zu betrachten oder anderweitig zu nutzen.

[0008] Wenn dies geschieht, versucht der Anwender häufig, einen Betrachter für das heruntergeladene Dokument oder Objekt aufzuspüren, indem er Verzeichnisse von Programmen auf dem Server, von dem das Dokument oder Objekt erhalten wurde, oder auf anderen Servern durchsucht. Wenn ein Betrachter gefunden worden ist, der mit der Computerplattform des Anwenders kompatibel ist, kann der Anwender den Betrachter herunterladen und anschließend diesen ausführen, um das vorher heruntergeladene Objekt zu betrachten. Es gibt jedoch einige signifikante Risiken für den Anwender, die mit dem Ausführen eines Betrachters von unbekanntem Ursprung verbunden sind. Zum Beispiel kann das heruntergeladene Betrachterprogramm eingebettete "Virus"-Programme enthalten, die die Integrität des Computers des Anwenders beeinträchtigen, oder das heruntergeladene Programm selbst kann auf Betriebsmittel zugreifen und/oder Informationen auf dem Computer des Anwenders zerstören, was den Wünschen des Anwenders entgegensteht. Die vorliegende Erfindung beseitigt diese Schwierigkeiten, indem sie für ein automatisches Herunterladen von Betrachtern für Dokumente und Objekte und für die automatische Integritätsverifizierung dieser Programme sorgt, bevor der heruntergeladene Betrachter ausgeführt werden kann.

ZUSAMMENFASSUNG DER ERFINDUNG

[0009] Die vorliegende Erfindung ist ein "Klassenlader" zum Erhalten (d. h. Herunterladen) von Objekten und Objektbetrachtern von entfernten Computerknoten und zum Aktivieren lokal gespeicherter Objektbetrachter, um Objekte zu betrachten. Wenn ein Anwender ein zu betrachtendes Objekt auswählt, wie z. B. unter Verwendung des Hyperlink-Merkmals des World Wide Web, wird ein herkömmliches Herunterladen des entsprechenden Objekts eingeleitet. Der Klassenlader der vorliegenden Erfindung nutzt jedoch die am Anfang des Objektherunterladeprozesses gewonnenen Datentypinformationen, um zu ermitteln, ob ein Betrachter für das bezeichnete Objekt auf der Workstation des Anwenders verfügbar ist.

[0010] Wenn kein geeigneter Betrachter lokal verfügbar ist, lokalisiert der Klassenlader automatisch einen geeigneten Betrachter auf dem Server, von dem das Objekt heruntergeladen wird, oder auf irgendeinem anderen geeigneten Server, der der Workstation des Anwenders bekannt ist. Der Klassenlader lädt den lokalisierten Betrachter herunter und aktiviert anschließend eine Programmverifikationsprozedur, um die Integrität des heruntergeladenen Betrachters zu verifizieren, bevor der Betrachter ausgeführt wird. Sobald ein Betrachter verifiziert worden ist, wird der Betrachter zur lokalen Betrachterbibliothek des Anwenders hinzugefügt, das Herunterladen des betreffenden Objekts abgeschlossen und die Ausführung des Betrachters zum Betrachten des heruntergeladenen Objekts freigegeben.

[0011] Wenn kein geeigneter Betrachter lokalisiert werden kann, oder der einzige lokalisierte Betrachter nicht die Verifikationsprozedur passiert, wird das Herunterladen des betreffenden Objekts abgebrochen.

[0012] Die vorliegende Erfindung verifiziert die Integrität der Computerprogramme, die in einer Bytecode-Sprache geschrieben sind, die als die OAK-Sprache vertrieben wird und einen beschränkten Satz von datentypspezifischen Bytecodes verwendet. Alle verfügbaren Quellcode-Bytecodes in der Sprache sind entweder (A) Stapeldaten beanspruchende Bytecodes, die zugeordnete Datentypbeschränkungen für die Typen von Daten aufweisen, die von jedem solchen Bytecode verarbeitet werden können, oder (B) verwenden keine Stapeldaten, beeinflussen jedoch den Stapel, indem sie entweder Daten eines bekannten Datentyps auf den Stapel legen oder Daten vom Stapel ohne Rücksicht auf den Datentyp entfernen, oder (C) verwenden weder Stapeldaten noch fügen sie Daten zum Stapel hinzu.

[0013] Die vorliegende Erfindung schafft ein Verifikationswerkzeug und ein Verfahren zum Identifizieren irgendeiner Befehlssequenz vor der Ausführung eines Bytecode-Programms, die versucht, Daten des falschen Typs für einen solchen Bytecode zu verarbeiten, oder zum Feststellen, ob die Ausführung irgendwelcher Bytecode-Befehle im spezifizierten Programm einen Unterlauf oder Überlauf des Operandenstapels hervorrufen würde, und zum Verhindern der Verwendung eines solchen Programms.

[0014] Die Bytecode-Programmverifikationseinrichtung der vorliegenden Erfindung enthält einen virtuellen Operandenstapel zum vorübergehenden Speichern von Stapelinformationen, die die in einem Programmoperandenstapel gespeicherten Daten angeben, während der Ausführung eines spezifizierten Bytecode-Pro-

gramms. Die Verifikationseinrichtung verarbeitet das spezifizierte Programm durch sequentielles Verarbeiten jedes Bytecode-Befehls des Programms, Aktualisieren des virtuellen Operandenstapels, um die Anzahl, die Sequenz und die Datentypen der Daten anzuzeigen, die im Operandenstapel am jeweiligen Punkt im Programm gespeichert wurden. Die Verifikationseinrichtung vergleicht ferner die virtuellen Stapelinformationen mit Datentypbeschränkungen, die jedem Bytecode-Befehl zugeordnet sind, um zu ermitteln, ob während der Programmausführung der Operandenstapel Daten enthalten würde, die mit den Datentypbeschränkungen des Bytecode-Befehls inkonsistent sind, und ermittelt ferner, ob irgendwelche Bytecode-Befehle im spezifizierten Programm einen Unterlauf oder Überlauf des Operandenstapels hervorrufen würden.

[0015] Um eine genaue Analyse des Befehlssequenzflusses des Bytecode-Programms zu vermeiden und um das mehrfache Verifizieren von Bytecode-Befehlen zu vermeiden, werden alle Punkte im spezifizierten Programm, denen in der Ausführung unmittelbar zwei oder mehr unterschiedliche Bytecodes im Programm vorausgehen können, identifiziert (als Mehrfacheinsprungpunkte bezeichnet). Im Allgemeinen ist wenigstens einer der zwei oder mehr unterschiedlichen Bytecodes im Programm ein Sprung/Verzweigungs-Bytecode. Während der Verarbeitung des spezifizierten Programms nimmt die Verifikationseinrichtung einen "Schnappschuss" des virtuellen Operandenstapels unmittelbar vor jedem Mehrfacheinsprungpunkt auf (d. h. nach irgendeinem der vorangehenden Bytecode-Befehle), vergleicht diesen Schnappschuss mit dem Zustand des virtuellen Operandenstapels nach der Verarbeitung jedes der anderen vorangehenden Bytecodebefehle für denselben Mehrfacheinsprungpunkt, und erzeugt einen Programmfehler, wenn die Zustände des virtuellen Stapels nicht identisch sind.

KURZBESCHREIBUNG DER ZEICHNUNGEN

[0016] Die beigefügten Zeichnungen, die in diese Beschreibung eingefügt sind und einen Teil derselben bilden, zeigen Ausführungsformen der Erfindung und dienen zusammen mit der Beschreibung zur Erläuterung der Prinzipien der Erfindung, wobei:

[0017] **Fig.** 1 zwei über ein Netz verbundene Computer zeigt;

[0018] **Fig.** 2 zwei über ein Netz verbundene Computer zeigt, wobei wenigstens einer der Computer eine Sekundärspeichervorrichtung zum Speichern mehrerer Kopien eines Quellprogramms in unterschiedlichen ausführbaren Formen enthält;

[0019] **Fig.** 3 zwei über ein Netz verbundene Computer zeigt, wobei wenigstens einer der Computer eine Bytecode-Programmverifikationseinrichtung und einen Klassenlader entsprechend der vorliegenden Erfindung enthält;

[0020] **Fig.** 4 ein Flussdiagramm des Ladeprozesses zum Zugreifen auf ein Bytecode-Programm und einen in einem entfernten Server gespeicherten Betrachter gemäß der bevorzugten Ausführungsform der vorliegenden Erfindung zeigt;

[0021] **Fig.** 5 Datenstrukturen zeigt, die von einer Bytecode-Verifikationseinrichtung während der Verifizierung eines Bytecode-Programms gemäß der vorliegenden Erfindung gehalten werden;

[0022] **Fig.** 6 ein Flussdiagramm des Bytecode-Programmverifikationsprozesses in der bevorzugten Ausführungsform der vorliegenden Erfindung zeigt; und

[0023] **Fig.** 7 ein Flussdiagramm des Bytecode-Programminterpretationsprozesses in der bevorzugten Ausführungsform der vorliegenden Erfindung zeigt.

GENAUE BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGSFORMEN

[0024] Im Folgenden wird auf bevorzugte Ausführungsformen der Erfindung genauer Bezug genommen, von denen Beispiele in den beigefügten Zeichnungen gezeigt sind. Obwohl die Erfindung in Verbindung mit bevorzugten Ausführungsformen beschrieben wird, ist klar, dass nicht beabsichtigt ist, dass die Erfindung auf diese Ausführungsformen beschränkt wird. Im Gegenteil soll die Erfindung Alternativen, Abwandlungen und Äquivalente abdecken, die im Umfang der beigefügten Ansprüche enthalten sein können.

[0025] Wie bei einem verteilten Computersystem **200** in **Fig.** 2 gezeigt, ist ein erster Computerknoten **202** mit einem zweiten Computerknoten **204** über ein Computerkommunikationsnetz **216**, wie z. B. das Internet, ver-

bunden. Der erste Computerknoten **202** enthält eine Zentraleinheit **206**, eine Anwenderschnittstelle **208**, einen Primärspeicher (RAM) **210**, einen Sekundärspeicher (Plattenspeicher) **212** und ein Modem oder eine andere Kommunikationsschnittstelle **214**, die den ersten Computerknoten **202** mit dem Computerkommunikationsnetz **216** verbinden. Der Plattenspeicher **212** speichert Programme zur Ausführung durch den Prozessor **206** sowie Datendateien und andere Informationen.

[0026] Der zweite Computerknoten **204**, der hier als ein Dateiserver oder als ein anderer Informationsserver konfiguriert angenommen wird, enthält eine Zentraleinheit **218**, eine Anwenderschnittstelle **220**, einen Primärspeicher (RAM) **222**, einen Sekundärspeicher (Plattenspeicher) **224** und ein Modem oder eine andere Kommunikationsschnittstelle **226**, die den zweiten Computerknoten mit dem Computerkommunikationsnetz **216** verbinden. Der Plattenspeicher **224** enthält ein Datei- und/oder Objektverzeichnis **228** (manchmal als Plattenverzeichnis oder Katalog bezeichnet) zum Lokalisieren von Informationen, die im Sekundärspeicher **224** gespeichert sind, Objekte **230**, eine Betrachterbibliothek **232** und andere Programme **234** für die Ausführung durch den Prozessor **218** und/oder die Verteilung auf andere Computerknoten.

[0027] Der erste und der zweite Computerknoten **202** und **204** können unterschiedliche Computerplattformen und Betriebssysteme **236**, **237** verwenden, so dass Objektcodeprogramme, die auf einem der beiden Computerknoten ausgeführt werden, auf dem anderen nicht ausgeführt werden können. Zum Beispiel kann der Serverknoten **204** ein Sun-Microsystems-Computer sein, der ein Unix-Betriebssystem verwendet, während der Anwender-Workstation-Knoten **202** ein IBM-kompatibler Computer sein kann, der einen 80486-Mikroprozessor und ein Microsoft-DOS-Betriebssystem verwendet. Ferner können andere Anwender-Workstations, die mit demselben Netz verbunden sind und denselben Server **204** nutzen, eine Vielzahl unterschiedlicher Computerplattformen und unterschiedlicher Betriebssysteme verwenden.

[0028] In der Vergangenheit hat ein Server **204**, der für die Verteilung von Software auf einem Netz mit Computern vieler Typen verwendet worden ist, mehrere unterschiedliche Bibliotheken (z. B. mehrere unterschiedliche Betrachterbibliotheken **232**) von Software für alle unterschiedlichen Computerplattformtypen (z. B. Unix, Windows, DOS, Macintosh usw.) gespeichert. Um die Bedürfnisse der verschiedenen Systemanwender zu unterstützen, müsste ein Server dementsprechend sowohl mehrere Versionen desselben Computerprogramms (**238** und **239**) sowie mehrere Objektbetrachter (**241** und **243**) speichern, einen für jeden Computerplattformtyp. Unter Verwendung der vorliegenden Erfindung jedoch können viele verschiedene Anwender unterstützt werden durch die Verteilung einer einzelnen Bytecode-Version des Programms.

[0029] In **Fig.** 3 ist ein verteiltes Computersystem **250** gezeigt, das die Lehre der vorliegenden Erfindung verkörpert. Ein erster Computerknoten **252** ist mit einem zweiten Computerknoten **254** über ein Computerkommunikationsnetz **266**, wie z. B. das Internet, verbunden. Wie beim Stand der Technik können wiederum ein erster und ein zweiter Computerknoten **252** und **254** unterschiedliche Computerplattformen und Betriebssysteme **255**, **256** verwenden, so dass Objektcodeprogramme, die auf einem der beiden Computerknoten ausgeführt werden, auf dem anderen nicht ausgeführt werden können. Zum Beispiel kann der Serverknoten **254** ein Sun-Microsystems-Computer sein, der ein Unix-Betriebssystem verwendet, während der Anwender-Workstation-Knoten **252** ein IBM-kompatibler Computer sein kann, der einen 80486-Mikroprozessor und ein Microsoft-DOS-Betriebssystem verwendet, wie oben in Verbindung mit **Fig.** 2 beschrieben worden ist. Der erste Computerknoten **252** enthält eine Zentraleinheit **257**, eine Anwenderschnittstelle **258**, einen Primärspeicher (RAM) **260**, einen Sekundärspeicher (Plattenspeicher) **262** und ein Modem oder eine andere Kommunikationsschnittstelle **264**, die den ersten Computerknoten **252** mit dem Computerkommunikationsnetz **266** verbinden. Der Plattenspeicher **262** speichert Programme für die Ausführung durch den Prozessor **257**, wobei wenigstens eines der Programme ein Bytecode-Programm **267** ist, das eine ausführbare Form aufweist. Zum Zweck dieser Beschreibung wird angenommen, dass der erste Computerknoten **252** das Bytecode-Programm **267** vom zweiten Computerknoten **254** über das Computerkommunikationsnetz **266** empfängt, dessen Einzelheiten im Folgenden in Verbindung mit dem Klassenlader genauer beschrieben werden.

[0030] In der bevorzugten Ausführungsform ist das Bytecode-Programm als eine OAK-Anwendung geschrieben, die dann, wenn sie kompiliert oder interpretiert wird, eine Serie ausführbarer Befehle ergibt. Eine Auflistung aller Quellcode-Bytecode-Befehle im OAK-Befehlssatz ist in Tabelle 1 gezeigt. Der OAK-Befehlssatz ist gekennzeichnet durch Bytecode-Befehle, die datentyp-spezifisch sind. Genauer, der OAK-Befehlssatz unterscheidet dieselbe Basisoperation für unterschiedliche primitive Datentypen durch Zuweisen separater Opcodes. Dementsprechend sind mehrere Bytecodes innerhalb des Befehlssatzes enthalten, um dieselbe Grundfunktion auszuführen (um z. B. zwei Zahlen zu addieren), wobei jeder solcher Bytecodes verwendet wird, um nur Daten eines entsprechenden verschiedenen Datentyps zu verarbeiten. Außerdem ist der OAK-Befehlssatz gekennzeichnet durch nicht enthaltene Befehle. Es gibt z. B. keine "berechneten GOTO-Befehle" im

OAK-Sprache-Befehlssatz, und es gibt keine Befehle zum Modifizieren von Objektreferenzen oder zum Erzeugen neuer Objektreferenzen (außer dem Kopieren einer existierenden Objektreferenz). Diese zwei Einschränkungen des OAK-Befehlssatzes sowie andere tragen dazu bei, sicherzustellen, dass ein beliebiges Bytecode-Programm, das Daten in einer Weise verwendet, die mit datentypspezifischen Befehlen im OAK-Befehlssatz konsistent ist, nicht die Integrität des Computersystems eines Anwenders verletzt.

[0031] In der bevorzugten Ausführungsform sind die verfügbaren Datentypen Ganzzahl, lange Ganzzahl, kurze Ganzzahl (vorzeichenbehaftete 16-Bit-Ganzzahl), Gleitkommazahl mit einfacher Genauigkeit, Gleitkommazahl mit doppelter Genauigkeit, Byte, Zeichen und Objektzeiger (hier manchmal als Objektreferenz bezeichnet). Der "Objektreferenz"-Datentyp enthält eine virtuell unbegrenzte Anzahl von Datenuntertypen, da jeder "Objektreferenz"-Datentyp eine Objektklassenspezifikation als Teil des Datentyps enthalten kann. Außerdem sind die in Programmen verwendeten Konstanten ebenfalls datentypisiert, wobei die verfügbaren Konstanten-Datentypen in der bevorzugten Ausführungsform die oben erwähnten Datentypen umfassen, zusätzlich zu Klasse, Feldreferenz, Verfahrensreferenz, Kette und Asciz, die alle zwei oder mehr Bytes mit einem speziellen Zweck darstellen.

[0032] Die wenigen Bytecodes, die datentypunabhängig sind, führen Stapelmanipulationsfunktionen aus, wie z. B. (A) Duplizieren eines oder mehrerer Wörter auf dem Stapel und Platzieren derselben an speziellen Orten innerhalb des Stapels, wodurch mehr Stapelelemente des bekannten Datentyps erzeugt werden, oder (B) Löschen eines oder mehrerer Elemente vom Stapel. Einige andere datentypunabhängige Bytecodes verwenden keine Wörter auf dem Stapel und lassen den Stapel unverändert, oder fügen dem Stapel Wörter hinzu, ohne irgendwelche vorher auf dem Stapel befindlichen Wörter zu verwenden. Diese Bytecodes besitzen keine Datentypbeschränkungen bezüglich der Stapelinhalte vor ihrer Ausführung, modifizieren jedoch alle Stapelinhalte in einer vollständig vorhersagbaren Weise bezüglich der Datentypen der Elemente im Stapel. Als Ergebnis kann die Anzahl der Operanden im Stapel und der Datentyp aller Operanden im Stapel zu allen Zeitpunkten mit 100% Zuverlässigkeit vorhergesagt (d. h. berechnet) werden.

[0033] Der zweite Computerknoten **254**, der hier als ein Dateiserver oder als ein anderer Informationsserver konfiguriert angenommen wird, enthält eine Zentraleinheit **268**, eine Anwenderschnittstelle **270**, einen Primärspeicher (RAM) **272**, einen Sekundärspeicher (Plattenspeicher) **274** und einen Modem oder eine andere Kommunikationsschnittstelle **276**, die den zweiten Computerknoten mit dem Computerkommunikationsnetz **266** verbinden. Der Plattenspeicher **274** umfasst ein Verzeichnis **280**, Objekte **282**, die ein erstes Objekt **283**, eine Betrachterbibliothek **284** und Programme **286** enthalten zur Ausführung durch den Prozessor **268** und/oder zur Verteilung an andere Computerknoten, wobei wenigstens eines von diesen das Bytecode-Programm **267** für die Übertragung zum Computerknoten **252** ist.

[0034] Wie in **Fig.** 3 gezeigt, speichert der erste Computerknoten **252** in seinem Sekundärspeicher **262** ein Klassenladerprogramm **296** zum Wiedergewinnen (d. h. Herunterladen) von Objekten und Objektbetrachtern von anderen Computerknoten und zum Aktivieren lokal gespeicherter Objektbetrachter zum Betrachten von Objekten. Der Klassenlader **296** prüft ferner automatisch (am Ort des Workstation-Knotens des Endanwenders) die heruntergeladenen Objektbetrachter, um die Integrität jedes Betrachters zu prüfen, bevor er vom jeweiligen Anwender ausgeführt wird.

[0035] Zum Zweck dieses Dokuments kann ein "Objekt", das unter Verwendung eines zugeordneten Betrachters "betrachtet" werden kann, entweder (A) ein Nur-Daten-Typ von Objekt sein, wie z. B. eine Datei oder eine andere Datenstruktur, die Daten eines spezifischen Typs oder Formats enthält, wie z. B. JPEG-, GIF-, MPEG- oder MPEG2-Daten, ohne irgendein eingebettetes Verfahren oder eine Software aufzuweisen, oder (B) ein ein Verfahren speicherndes Objekt sein, wie z. B. eine Datei oder eine andere Datenstruktur, die ein oder mehrere eingebettete Verfahren sowie optional Daten enthält. Zum Beispiel können unterschiedliche Betrachter erforderlich sein zum Betrachten von Nur-Daten-Objekten, die unterschiedliche Bilddatentypen speichern, wie z. B. JPEG und GIF, und zum Betrachten von Nur-Daten-Objekten, die unterschiedliche Videoprogrammtypen speichern, wie z. B. MPEG und MPEG2. Andere Beispiele können unterschiedliche Betrachter sein zum Betrachten von Schaubildern von Daten, Betrachter mit eingebauter Datenentschlüsselungssoftware zum Betrachten verschlüsselter Daten (wenn der Entschlüsselungsschlüssel dem Anwender bekannt ist) und dergleichen.

[0036] Außerdem können unterschiedliche Betrachter erforderlich sein für ein Verfahren speichernde Objekte, die unterschiedliche interne Programmtypen verwenden. Zum Beispiel können unterschiedliche interne Programmtypen in verschiedenen ein Verfahren speichernde Objekte unterschiedliche Skriptsprachen verwenden oder können die Verfügbarkeit unterschiedlicher Bibliotheken von Hilfsprogrammen annehmen, wodurch unterschiedliche Betrachter erforderlich sind.

[0037] Ein "Betrachter" (manchmal als Interpreter bezeichnet) decodiert Daten und/oder Befehle in einem spezifizierten Objekt und führt im Allgemeinen alle Berechnungen und Operationen aus, die erforderlich sind, um Objekte eines bestimmten Datentyps oder einer Klasse nutzbar zu machen. In der vorliegenden Erfindung sind solche Objektbetrachter Bytecode-Programme, die in einer Quellcode-Bytecode-Sprache geschrieben sind, so dass die Integrität jedes Objektbetrachters unabhängig von einem Endanwender geprüft werden kann durch Ausführen einer Bytecode-Programmverifikationseinrichtung **240**. Die Bytecode-Programmverifizierung wird im Folgenden genauer beschrieben.

[0038] Es ist zu beachten, dass ein verteiltes Computersystem **250** plattformunabhängige Objektbetrachter gemäß der vorliegenden Erfindung sowie andere Objektbetrachter enthalten kann, die nicht plattformunabhängig sind und die nicht unter Verwendung der Bytecode-Programmverifikationseinrichtung **240** und des Klassenladers **296**, die Werkzeuge der vorliegenden Erfindung sind, verifiziert werden können. In einem solchen Hybridsystem werden die Vorteile der automatisierten Betrachterintegritätsverifizierung der vorliegenden Erfindung für Bytecode-Betrachterprogramme verwirklicht, jedoch nicht für die anderen Betrachterprogramme.

[0039] Der Klassenlader **296** ist ein ausführbares Programm zum Laden und Verifizieren von Objekten und Objektbetrachtern von einem entfernten Server. Wenn z. B. ein Dokument auf dem World Wide Web (WWW) des Internets betrachtet wird, kann eine Seite des Dokuments Referenzen auf andere Dokumente oder Objekte enthalten. Ein Anwender kann auf solche andere Dokumente oder Objekte zugreifen durch Auswählen eines gegebenen Objekts über einen zugehörigen Hyperlink. Eine solche Auswahl wird üblicherweise von einem Anwender in Verbindung mit einer graphischen Anwenderoberfläche auf einem Workstation-Knoten durchgeführt, indem ein Knopf auf einer Zeigevorrichtung niedergedrückt wird, während die Zeigevorrichtung verwendet wird, um auf ein graphisches Abbild zu zeigen, das die Hyperlink-Auswahl darstellt.

[0040] Während des Auswahlprozesses kann das Dokument oder Objekt, das derzeit betrachtet wird, Referenzen auf andere Dokumente oder Objekte enthalten, einschließlich solcher mit einem Datentyp, der für die Workstation des Anwenders unbekannt ist. Der Klassenlader der vorliegenden Erfindung wird verwendet, um sowohl einen Betrachter zu lokalisieren, der einem "fremden" Datentyp zugeordnet ist, als auch um die Programmintegrität aller heruntergeladenen Bytecode-Programme vor ihrer Ausführung durch den Anwender zu verifizieren.

[0041] Der Klassenlader **296** führt drei Hauptfunktionen aus. Zuerst prüft der Klassenlader die Datentypen der heruntergeladenen Objekte (und ihrer zugeordneten Bytecode-Programme), um zu ermitteln, ob die Anwender-Workstation einen zugeordneten Betrachter in einer "Betrachterbibliothek" **298** in ihrem eigenen lokalen Speicher **262** besitzt. Zweitens, wenn der Klassenlader keinen geeigneten Betrachter lokalisieren kann, führt er eine Suchroutine sowohl auf dem Quellenserver als auch auf anderen Servern aus, die er kennt, um den geeigneten Betrachter zu lokalisieren und herunterzuladen. Wenn kein Betrachter lokalisiert werden kann, wird das Objekt und/oder das Bytecode-Programm, die heruntergeladen worden sind, für die Suche nach einem geeigneten Betrachter verworfen. Schließlich aktiviert der Klassenlader nach dem Lokalisieren des geeigneten Betrachters auf einer entfernten Quelle die Ausführung einer Bytecode-Verifikationseinrichtung **240**, um den heruntergeladenen Betrachter vor der Ausführung des Betrachters in Verbindung mit einem Bytecode-Programminterpreter **242** oder der Kompilierung durch einen Bytecode-Programmcompiler **244** zu prüfen. Nach der Verifizierung kann der heruntergeladene Betrachter in der lokalen Betrachterbibliothek **298** des Anwenders gespeichert werden.

[0042] Mit Bezug auf die **Fig.** 3 und 4 und den Anhang 1 wird die Ausführung des Klassenladerprogramms **296** genauer beschrieben hinsichtlich der Wiedergewinnung eines Bytecode-Programms über ein zugehöriges Objekt. Der Anhang 1 listet eine Pseudocode-Darstellung des Klassenladerprogramms auf. Der im Anhang 1 verwendete Pseudocode ist im Wesentlichen eine Computersprache, die universale Computersprachenkonventionen verwendet. Obwohl der hier verwendete Pseudocode nur für die Zwecke dieser Beschreibung erfunden worden ist, ist er so beschaffen, dass er leicht von einem beliebigen Computerprogrammierer vom Fach verstanden werden kann.

[0043] Wie in **Fig.** 4 gezeigt, beginnt die Anwender-Workstation **252** einen Herunterladeprozess durch Öffnen (**304**) einer Verbindung zu einem Server **254**, der ein herunterzuladendes Objekt **283** enthält. Der Klassenlader **296** leitet die Übertragung des Objekt-Bytecode-Programms ein (**306**) durch Hyperlink-Auswahl des Objekts, woraufhin der Server **254** einen "Hantierer" für das bezeichnete Objekt zur Anwender-Workstation **252** überträgt. Der Hantierer wird vor dem Körper des bezeichnenden Objekts wiedergewonnen und enthält Informationen bezüglich der Eigenschaften des bezeichneten Objekts, einschließlich des Objektdatentyps (manchmal als Objektklasse bezeichnet).

[0044] Eine erste Prüfung (**308**) wird durchgeführt, um zu ermitteln, ob der dem wiederzugewinnenden Objekt zugeordnete Datentyp dem System des Anwenders bekannt ist. Genauer, der Klassenlader durchsucht eine Betrachterbibliothek **298**, die im Sekundärspeicher **262** der Anwender-Workstation **252** vorhanden ist, um zu ermitteln, ob ein geeigneter Betrachter für die Objekte des ermittelten Datentyps verfügbar ist. Die Betrachterbibliothek **298** enthält eine Auflistung aller Datentyp-Betrachter, die derzeit für die Anwender-Workstation zur Verfügung stehen, sowie deren entsprechende Orte im Speicher. Auf diese Weise führt der Klassenlader eine Vorverarbeitung für das herunterzuladende Objekt während der anfänglichen Quittierung durch, um eine Kompatibilität mit der Anwender-Workstation-Plattform vor dem wirklichen Herunterladen des Körpers des bezeichneten Objekts festzustellen. Wenn ein geeigneter Betrachter lokalisiert wird, schließt der Klassenlader (**310**) das Herunterladen des bezeichneten Objekts ab.

[0045] Wenn kein geeigneter Betrachter innerhalb der Betrachterbibliothek **298** lokalisiert wird, was anzeigt, dass das ausgewählte Objekt einen Datentyp aufweist, der der Anwender-Workstation **252** nicht vertraut ist, führt der Klassenlader eine Suche nach einem geeigneten Betrachter durch. In den meisten Fällen ist der erste Ort zum Suchen nach einem geeigneten Betrachter derselbe Server, auf dem das ausgewählte Objekt gespeichert ist. Somit öffnet der Klassenlader (**312**) eine zweite Verbindung zu demselben Server, der die Quelle des bezeichneten Objekts ist, und fordert einen Betrachter für den angegebenen Datentyp an (**314**). Wenn der Server den geeigneten Betrachter enthält, wird der Betrachter in die Workstation des Anwenders heruntergeladen (**315**).

[0046] Nach Abschluss des Herunterladens, wenn der heruntergeladene Betrachter ein Bytecode-Programm (**316**) ist, leitet der Klassenlader eine Verifizierung (**317**) des Betrachterprogramms ein durch Aktivieren der Bytecode-Programmverifikationseinrichtung **240**. Die Bytecode-Programmverifikationseinrichtung **240** ist ein ausführbares Programm, das die Operandendatentypkompatibilität und richtige Stapelmanipulationen in einem spezifizierten Bytecode-(Quell)-Programm vor der Ausführung des Bytecode-Programms durch den Prozessor **257** prüft. Die Operation des Bytecode-Verifikationseinrichtungsprogramms **240** wird im Folgenden genauer beschrieben. Wenn die Verifizierung erfolgreich ist (**318**), speichert (**319**) die Betrachtersuchvorrichtung den verifizierten Objektbetrachter in der Betrachterbibliothek **298** und aktualisiert das Verzeichnis in der Bibliothek, um die Verfügbarkeit des neuen Datentypbetrachters wiederzugeben. Wenn die Verifizierung nicht erfolgreich ist, wird der heruntergeladene Betrachter gelöscht (**320**).

[0047] Bestimmte Ausführungsformen der vorliegenden Erfindung erlauben das automatische Herunterladen unter Verwendung sowohl verifizierbarer als auch nicht verifizierbarer Objektbetrachter. In diesen Ausführungsformen wird nach dem Herunterladen eines Objektbetrachters (**315**) dann, wenn der heruntergeladene Betrachter kein Bytecode-Programm (**316**) ist, eine Ermittlung durchgeführt (**321**), ob der Objektbetrachter akzeptiert werden soll. Zum Beispiel kann der Anwender gefragt werden, ob er den Objektbetrachter akzeptiert, oder eine Vorgabeentscheidung zum Akzeptieren oder Nichtakzeptieren solcher Objektbetrachter kann in einer Konfigurationsdatei enthalten sein. Wenn der nicht verifizierbare Objektbetrachter akzeptiert wird, wird er in der Betrachterbibliothek (**319**) gespeichert, wobei dann, wenn er nicht akzeptiert wird, der heruntergeladene Betrachter gelöscht wird (**320**).

[0048] Wenn die Schritte **308** und **314** keinen für die Verwendung mit dem ausgewählten Objekt geeigneten Betrachter lokalisieren können, da weder der Server noch die Anwender-Workstation einen geeigneten Betrachter enthalten, erweitert der Klassenlader seine Suche auf andere Server-Sites oder entfernte Anwender-Workstations (z. B. eine bekannte Serverliste **327**), die der Workstation des Anwenders bekannt sind (Schritt **322** und **323**). In **Fig.** 3 ist ein zweiter Server **324** gezeigt, der einen Sekundärspeicher **325** mit einer Betrachterbibliothek **326** enthält. Wenn der geeignete Betrachter in der Betrachterbibliothek **326** des zweiten Servers **324** lokalisiert wird, lädt der Klassenlader das Betrachterprogramm herunter und verifiziert es, entsprechend den oben erwähnten Schritten **315**–**321**. Der Klassenlader wiederholt diesen Prozess und prüft abwechselnd die Server, bis alle bekannten Betriebsmittel erschöpft sind oder ein geeigneter Betrachter lokalisiert und verifiziert ist. Wenn schließlich kein geeigneter Betrachter lokalisiert werden kann, wird das Herunterladen des betreffenden Objekts abgebrochen und eine Anwendernachricht erzeugt, um den Anwender darüber zu informieren, dass kein Betrachter für das bezeichnete Objekt lokalisiert werden konnte (**328**).

[0049] Wie oben angegeben ist, wird in dem Fall, dass ein geeigneter Objektbetrachter bereits in der Betrachterbibliothek **298** auf der Workstation des Anwenders gespeichert war (**308**) oder erfolgreich heruntergeladen, verifiziert und der Betrachterbibliothek des Anwenders hinzugefügt wurde, das Laden des ausgewählten Objekts abgeschlossen (**310**). Wenn das heruntergeladene Objekt eines oder mehrere eingebettete Bytecode-Programme (**330**) enthält und daher ein ein Verfahren speicherndes Objekt ist, werden die Bytecode-Programme im heruntergeladenen Objekt verifiziert (**332**) durch Aktivieren der Ausführung der Bytecode-Verifika-

tionseinrichtung für diese eingebetteten Programme. Wenn die Verifikationseinrichtung einen "Erfolg"-Rückgabecode nach der Verarbeitung der eingebetteten Programme erzeugt (**334**), wird das heruntergeladene Objekt mit dem zugehörigen Objektbetrachter betrachtet (**335**). Wenn die Verifikationseinrichtung die Verarbeitung des eingebetteten Programms abbricht aufgrund der Erfassung eines Programms, das mit den Anforderungen der Verifikationseinrichtung nicht konform ist (**334**), wird das heruntergeladene Objekt gelöscht (**336**) und eine entsprechende Anwendernachricht erzeugt.

[0050] Für den Fall, dass das heruntergeladene Objekt keine eingebetteten Bytecode-Programme enthält (**330**), werden die Schritte **332**–**334** ausgelassen und das Objekt wird mit dem geeigneten Betrachter (**335**) betrachtet.

[0051] Wie wiederum in **Fig.** 3 gezeigt ist, speichert der erste Computerknoten **252** ferner in seinem Sekundärspeicher **262** ein Bytecode-Verifikationseinrichtungsprogramm **240** zum Prüfen der Integrität der spezifizierten Bytecode-Programme, sowie einen Bytecode-Interpreter **242** zum Ausführen spezifizierter Bytecode-Programme. Alternativ oder zusätzlich kann der erste Computerknoten **252** einen Bytecode-Compiler **244** speichern zum Umsetzen eines verifizierten Bytecode-Programms in ein Objektcodeprogramm für eine effizientere Ausführung des Bytecode-Programms als durch den Interpreter **242**.

[0052] Die Bytecode-Verifikationseinrichtung **240** ist ein ausführbares Programm, das die Operandendatentypkompatibilität und die geeigneten Stapelmanipulationen in einem spezifizierten Bytecode-(Quell)-Programm prüft vor der Ausführung des Bytecode-Programms durch den Prozessor **257** unter der Steuerung des Bytecode-Interpreters **242** (oder vor der Kompilierung des Bytecode-Programms durch den Compiler **244**). Jedes Bytecode-Programm **267** (einschließlich der heruntergeladenen Objektverifikationseinrichtung) besitzt einen zugeordneten Prüfstatuswert **302**, der anfangs auf Falsch gesetzt wird, wenn das Programm von einem anderen Ort heruntergeladen wird. Der Prüfstatuswert **302** für das Programm wird von der Bytecode-Verifikationseinrichtung **240** nur auf Wahr gesetzt, nachdem das Programm verifiziert worden ist und durch keinen der Datentyp- und Stapelnutzungstests gefallen ist, die von der Verifikationseinrichtung **240** ausgeführt werden.

Die Bytecode-Programmverifikationseinrichtung

[0053] Mit Bezug auf **Fig.** 5 wird im Folgenden die Ausführung der Bytecode-Programmverifikationseinrichtung **240** in Verbindung mit einem bestimmten Bytecode-Programm **340** erläutert. Die Verifikationseinrichtung **240** verwendet einige vorübergehende Datenstrukturen zum Speichern von Informationen, die sie während des Verifikationsprozesses benötigt. Genauer verwendet die Verifikationseinrichtung **240** einen Stapelzähler **342**, einen virtuellen Stapel **344**, ein virtuelles lokales Variablenfeld **345** und eine Stapelschnappschuss-Speicherstruktur **346**.

[0054] Der Stapelzähler **342** wird von der Verifikationseinrichtung **240** aktualisiert, während sie die virtuellen Stapelmanipulationen verfolgt, um die aktuelle Anzahl von virtuellen Stapeleinträgen **344** wiederzugeben.

[0055] Der virtuelle Stapel **344** speichert Datentypinformationen bezüglich jedes Datums, das vom Bytecode-Programm **340** im Operandenstapel während der wirklichen Ausführung gespeichert wird. In der bevorzugten Ausführungsform wird der virtuelle Stapel **344** in derselben Weise wie ein regulärer Stapel verwendet, mit der Ausnahme, dass anstelle der Speicherung wirklicher Daten und Konstanten der virtuelle Stapel **344** einen Datentypindikatorwert für jedes Datum speichert, das während der wirklichen Ausführung des Programms im Operandenstapel gespeichert wird. Wenn somit z. B. während der wirklichen Ausführung des Stapels drei Werte zu speichern sind:
Hantierer-für-ObjektA
5
1
sind die entsprechenden virtuellen Stapeleinträge
R
I
I
wobei "R" im virtuellen Stapel eine Objektreferenz anzeigt und jedes "I" im virtuellen Stapel eine Ganzzahl anzeigt. Ferner würde der Stapelzähler **342** in diesem Beispiel einen Wert gleich 3 speichern, entsprechend den drei Werten, die im virtuellen Stapel **344** gespeichert sind.

[0056] Daten jedes möglichen Datentyps sind einem entsprechenden virtuellen Stapelmarkenwert zugeordnet, z. B.: Ganzzahl (I), lange Ganzzahl (L), Gleitkommazahl mit einfacher Genauigkeit (F), Gleitkommazahl

mit doppelter Genauigkeit (D), Byte (B), Kurz (S) und Objektreferenz (R). Der Markenwert für eine Objektreferenz enthält häufig einen Objektklassenwert (z. B. R:Punkt, wobei "Punkt" eine Objektklasse ist).

[0057] Das virtuelle lokale Variablenfeld **345** dient derselben Grundfunktion wie der virtuelle Stapel **344**. Das heißt, es wird verwendet, um Datentypinformationen für lokale Variablen zu speichern, die vom spezifizierten Bytecode-Programm verwendet werden. Da Daten häufig von Programmen zwischen lokalen Variablen und dem Operandenstapel übertragen werden, können die Bytecode-Befehle, die solche Datenübertragungen durchführen und anderweitig lokale Variablen verwenden, verifiziert werden, um sicherzustellen, dass die lokalen Variablen, auf die vom jeweiligen Bytecode-Befehl zugegriffen wird, konsistent mit den Datentypbenutzungsbeschränkungen dieser Bytecode-Befehle sind.

[0058] Im Betrieb verarbeitet die Verifikationseinrichtung **240** jeden Bytecode-Befehl, der das Abziehen eines Datums vom Stapel anfordert, und zieht dieselbe Anzahl von Datentypwerten vom virtuellen Stapel **344** ab. Die Verifikationseinrichtung vergleicht anschließend die vom virtuellen Stapel **344** "abgezogenen" Datentypwerte mit den Datentypanforderungen des Bytecode-Befehls. In ähnlicher Weise schiebt die Verifikationseinrichtung für jeden Bytecode-Befehl, der ein Schieben eines Datums auf den Stapel anfordert, einen entsprechenden Datentypwert auf den virtuellen Stapel.

[0059] Ein Aspekt der Programmverifizierung gemäß der vorliegenden Erfindung ist die Verifizierung, ob die Anzahl und der Datentyp der Operanden im Operandenstapelstatus zu jedem Zeitpunkt identisch ist, zu dem ein bestimmter Befehl ausgeführt wird. Wenn einem bestimmten Bytecode-Befehl bei der Ausführung unmittelbar zwei oder mehr unterschiedliche Befehle vorausgehen können, muss der virtuelle Stapelstatus unmittelbar nach der Verarbeitung jedes dieser unterschiedlichen Befehle verglichen werden. Üblicherweise ist wenigstens einer der unterschiedlichen vorangehenden Befehle ein bedingter oder unbedingter Sprung- oder Verzweigungsbefehl. Eine logische Folge der oben erwähnten "Stapelkonsistenz"-Forderung ist, dass jede Programmschleife nicht zu einer effektiven Addition oder Reduktion der Anzahl der im Operandenstapel gespeicherten Operanden führen darf.

[0060] Die Stapelschnappschuss-Speicherstruktur **346** wird verwendet, um "Schnappschüsse" des Stapelzählers **342** und des virtuellen Stapels **344** zu speichern, um einen effizienten Vergleich des virtuellen Stapelstatus an verschiedenen Punkten im Programm zu ermöglichen. Jeder gespeicherte Stapelschnappschuss besitzt die Form:
SC, DT1, DT2, DT3, ..., DTn
wobei SC der Stapelzählerwert ist, DT1 der erste Datentypwert im virtuellen Operandenstapel ist, DT2 der zweite Datentypwert im virtuellen Operandenstapel ist usw., bis DTn, welches der Datentypwert für das letztmögliche Element im virtuellen Operandenstapel ist.

[0061] Die Stapelschnappschuss-Speicherstruktur **346** ist gegabelt in einen Verzeichnisabschnitt **348** und einen Schnappschuss-Speicherabschnitt **350**. Der Verzeichnisabschnitt **348** wird verwendet, um Zielbefehlsidentifizierer zu speichern (z. B. die absolute oder relative Adresse jedes Zielbefehls), während der Schnappschussabschnitt **350** verwendet wird, um Schnappschüsse des virtuellen Stapels **344** zu speichern, die den Zielbefehlidentifizierern zugeordnet sind.

[0062] "Ziel"-Befehle sind so definiert, dass sie alle Bytecode-Befehle umfassen, die das Ziel eines Sprung- oder Verzweigungsbefehls sein können. Zum Beispiel enthält ein bedingter Verzweigungsbefehl eine Bedingung (die erfüllt sein kann, oder nicht) und eine Verzweigung, die anzeigt, an welchen Ort (Ziel) im Programm die Ausführung "springen" soll, falls die Bedingung erfüllt ist. Beim Auswerten eines bedingten Sprungbefehls nutzt die Verifikationseinrichtung **240** die Stapelschnappschuss-Speicherstruktur **346**, um sowohl die Identität des Zielortes (im Verzeichnisabschnitt **348**) als auch den Status des virtuellen Stapels **344** (im Schnappschussabschnitt **350**) unmittelbar vor dem Sprung zu speichern. Die Operation der Stapelschnappschuss-Speicherstruktur **346** wird im Folgenden in Verbindung mit der Beschreibung der Ausführung des Bytecode-Verifikationseinrichtungsprogramms genauer beschrieben.

[0063] Wie vorher beschrieben worden ist, enthält das Bytecode-Programm **340** mehrere datentypspezifische Befehle, die jeweils von der Verifikationseinrichtung **240** der vorliegenden Erfindung ausgewertet werden. Das Bytecode-Programm **350** enthält Befehle für Stapelmanipulationen **352** und **354** (Schieben einer ganzen Zahl auf den Stapel und Ziehen einer ganzen Zahl vom Stapel), einen Vorwärtssprung **356** und dessen zugehöriges Ziel **364**, einen Rückwärtssprung **366** und dessen zugehöriges Ziel **362**, sowie eine Ausführungs-Schleife **358** und ihr zugehöriges Ende **360** (die ein unbedingter oder bedingter Verzweigungsbefehl sein kann, in Abhängigkeit vom Typ der Ausführungs-Schleife). Da die Verifikationseinrichtung **240** der bevorzugten Ausführungs-

form der vorliegenden Erfindung nur versucht, Stapelmanipulationen und Datentypkompatibilitäten zu prüfen, kann die Operation der Bytecode-Verifikationseinrichtung unter Verwendung dieses repräsentativen Satzes von Befehlen erläutert werden.

[0064] Mit Bezug auf die **Fig.** 6A–6G und den Anhang 2 wird im Folgenden die Ausführung des Bytecode-Verifikationseinrichtungsprogramms **240** genauer beschrieben. Der Anhang 2 listet eine Pseudocode-Darstellung des Verifikationseinrichtungsprogramms auf. Der im Anhang 2 verwendete Pseudocode ist im Wesentlichen eine Computersprache, die universelle Computersprachenkonventionen verwendet. Obwohl der hier verwendete Pseudocode nur für den Zweck dieser Beschreibung erfunden worden ist, ist er so beschaffen, dass er von einem beliebigen Computerprogrammierer vom Fach verstanden werden kann.

[0065] Wie in **Fig.** 6A gezeigt, wird das heruntergeladene Bytecode-Programm für die Verarbeitung in die Bytecode-Verifikationseinrichtung **240** geladen (**400**). Die Verifikationseinrichtung **240** erzeugt (**402**) den virtuellen Stapel **344** und erzeugt das virtuelle lokale Variablenfeld **345** durch Zuweisen von Feldern von Stellen im Speicher, um Operanden und lokale Variablendatentypinformationen zu speichern. In ähnlicher Weise erzeugt (**404**) die Verifikationseinrichtung die Stapelschnappschuss-Speicherstruktur durch Zuweisen eines Feldes von Stellen im Speicher, um die Schnappschussinformationen zu speichern. Schließlich legt die Verifikationseinrichtung ein Register fest (**406**), das als Stapelzähler **342** dient, um die Anzahl der virtuellen Stapeleinträge zu verfolgen.

[0066] Das Bytecode-Programm wird zum ersten Mal durchlaufen, um Zielinformationen zu extrahieren, die bedingten und unbedingten Sprüngen und Schleifenbefehlen zugeordnet sind. Bei diesem ersten Durchlauf verarbeitet die Verifikationseinrichtung **300** sequentiell alle Befehle (Schritte **408**, **410**, **412**), wobei für jeden Befehl, der ein bedingter oder unbedingter Sprung ist (Schritt **414**), eine Darstellung des Zielortes für den Sprung im Verzeichnisabschnitt **348** der Stapelschnappschuss-Speicherstruktur **346** gespeichert wird (Schritt **416**), sofern nicht der Zielort bereits im Verzeichnis **348** gespeichert worden ist (Schritt **418**). Zum Beispiel können die absolute oder die relative Adresse des Zielbefehls im nächsten verfügbaren Slot des Verzeichnisses **348** gespeichert werden. Alle anderen Typen von Bytecode-Befehlen werden bei diesem ersten Durchlauf ignoriert.

[0067] Nachdem alle Befehle im Programm verarbeitet worden sind, wird das Verzeichnis **348** vorzugsweise sortiert (**420**), um die im Verzeichnis aufgeführten Zielorte in einer sequentiellen Adressenreihenfolge anzuordnen.

[0068] Wie in **Fig.** 5 gezeigt, wurde zum Zweck der Darstellung die Stapelschnappschuss-Speicherstruktur **346** mit Informationen geladen, die im Verzeichnisabschnitt **348** gespeichert worden wären, wenn der erste Durchlauf der Verifizierung abgeschlossen worden wäre auf der Grundlage der Bytecode-Befehle, die im Bytecode-Programm **350** gezeigt sind. Genauer, der Verzeichnisabschnitt wurde mit den Adressen geladen, die allen Zielen der bedingten und unbedingten Sprünge zugeordnet sind, welche im Bytecode-Programm vorhanden sind.

[0069] Wie in **Fig.** 6B gezeigt, wird ein zweiter Durchlauf durch das Bytecode-Programm eingeleitet, um die geeignete Verwendung des Operandenstapels und der Datentypen durch das Bytecode-Programm zu verifizieren. Der erste Befehl des Bytecode-Programms wird ausgewählt (**430**), wobei die Verifikationseinrichtung zuerst prüft (**432**), ob die Adresse für den ausgewählten Befehl im Verzeichnisabschnitt **348** der Stapelschnappschuss-Speicherstruktur **346** im obenbeschriebenen ersten Durchlauf gespeichert worden ist.

[0070] Wenn die Adresse des ausgewählten Befehls sich im Verzeichnis **348** befindet, was anzeigt, dass der ausgewählte Befehl das Ziel eines bedingten oder unbedingten Sprungs ist, prüft die Verifikationseinrichtung (**434**), ob ein zugeordneter Stapelschnappschuss im Schnappschussabschnitt **350** der Stapelschnappschuss-Speicherstruktur **346** gespeichert worden ist. Wenn kein Stapelschnappschuss gespeichert worden ist (was anzeigt, dass der Befehl ein Ziel eines Rückwärtssprungs ist), werden die Inhalte des virtuellen Stapels und der Stapelzähler in der Stapelschnappschuss-Speicherstruktur **346** gespeichert (**436**). Der Schnappschuss enthält Informationen über den Status des virtuellen Stapels unmittelbar vor der Ausführung des verarbeiteten Befehls, einschließlich eines Datentypwerts für jedes Datum, das auf den Stapel geschoben worden ist. Die Verifikationseinrichtung setzt anschließend den Verifikationsprozess fort und analysiert den individuellen Befehl, beginnend mit Schritt **450**, wie im Folgenden beschrieben wird.

[0071] Wenn ein Stapelschnappschuss für den aktuell ausgewählten Befehl gespeichert worden ist (was anzeigt, dass ein Sprungbefehl, der diesem Zielbefehl zugeordnet ist, bereits verarbeitet worden ist), vergleicht

die Verifikationseinrichtung (**438**) die virtuelle Stapelschnappschussinformation, die im Schnappschussabschnitt **350** der Stapelschnappschuss-Speicherstruktur **346** für den aktuell ausgewählten Befehl gespeichert ist, mit dem aktuellen Zustand des virtuellen Stapels. Wenn der Vergleich zeigt, dass der aktuelle Zustand und der Schnappschuss nicht übereinstimmen, wird eine Fehlernachricht oder ein Signal erzeugt (**440**), das den Ort im Bytecode-Programm angibt, an dem der Stapelstatusfehler aufgetreten ist. In der bevorzugten Ausführungsform entsteht ein Fehler, wenn der aktuelle virtuelle Stapel und der Schnappschuss nicht dieselbe Anzahl oder dieselben Typen von Einträgen aufweisen. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**442**). Das Setzen des Verifikationsstatuswertes **245** für das Programm auf Falsch verhindert die Ausführung des Programms durch den Bytecode-Interpreter **242** (**Fig.** 3).

[0072] Wenn der aktuelle virtuelle Stapel und der gespeicherte Schnappschuss für den aktuellen Befehl übereinstimmen (**438**), fährt die Verifikationseinrichtung mit dem Verifikationsprozess fort und analysiert den individuellen Befehl, beginnend mit Schritt **450**, wie im Folgenden beschrieben wird.

[0073] Wenn die Adresse des aktuell ausgewählten Befehls nicht innerhalb des Verzeichnisabschnitts **348** der Stapelschnappschuss-Speicherstruktur **346** gefunden wird, oder wenn kein Stapelstatusfehler erfasst wird, führt die Verifikationseinrichtung ausgewählte Verifizierungen einer Serie von Verifizierungen für den Befehl durch, in Abhängigkeit von der jeweiligen Befehlstapelnutzung und Befehlsfunktion.

[0074] Wie in **Fig.** 6C gezeigt, betrifft die erste durchzuführende Verifizierung Befehle, die Daten vom Operandenstapel ziehen. Wenn der aktuell ausgewählte Befehl Daten vom Stapel zieht (**450**), wird der Stapelzähler untersucht (**452**), um zu ermitteln, ob ausreichend Daten im Stapel vorhanden sind, die die Datenabziehanforderungen des Befehls erfüllen.

[0075] Wenn der Operandenstapel nicht genügend Daten für den aktuellen Befehl aufweist (**452**), was als Stapelunterlauf bezeichnet wird, wird in diesem Fall ein Fehlersignal oder eine Nachricht erzeugt (**454**), die den Ort im Programm angibt, an dem der Stapelunterlauf erfasst worden ist. Außerdem setzt die Verifikationseinrichtung anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**456**).

[0076] Wenn keine Stapelunterlaufbedingung erfasst wird, vergleicht die Verifikationseinrichtung die vorher im virtuellen Stapel gespeicherten Datentypcodeinformationen mit den Datentypanforderungen (falls vorhanden) des aktuell ausgewählten Befehls (**458**). Wenn z. B. der Opcode des analysierten Befehls eine Ganzzahladdition eines vom Stapel gezogenen Wertes fordert, vergleicht die Verifikationseinrichtung die Operandeninformation des Elements im virtuellen Stapel, das abgezogen wird, um sicherzustellen, dass es den geeigneten Datentyp, nämlich Ganzzahl, aufweist. Wenn der Vergleich eine Übereinstimmung liefert, löscht die Verifikationseinrichtung (**460**) die Informationen vom virtuellen Stapel, die dem Eintrag zugeordnet sind, der abgezogen wird, und aktualisiert den Stapelzähler **342**, so dass er die Anzahl der vom virtuellen Stapel **344** abgezogenen Einträge widerspiegelt.

[0077] Wenn zwischen den gespeicherten Operandeninformationen im abgezogenen Eintrag des virtuellen Stapels **344** und den Datentypanforderungen des derzeit ausgewählten Befehls keine Übereinstimmung erfasst wird (**458**), wird eine Nachricht erzeugt (**462**), die den Ort im Bytecode-Programm angibt, an dem der Fehler aufgetreten ist. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**456**). Dies schließt den Abzug-Verifikationsprozess ab.

[0078] Wenn wie in **Fig.** 6D gezeigt der aktuell ausgewählte Befehl Daten auf den Stapel schiebt (**470**), wird der Stapelzähler untersucht (**472**), um zu ermitteln, ob ausreichend Raum im Stapel vorhanden ist, um die Daten zu speichern, die der ausgewählte Befehl auf den Stapel schiebt. Wenn der Operandenstapel nicht genügend Raum aufweist, um die vom aktuellen Befehl auf den Stapel zu schiebenden Daten zu speichern (**472**), was als Stapelüberlauf bezeichnet wird, wird in diesem Fall ein Fehlersignal oder eine Nachricht erzeugt (**474**), die den Ort im Programm angibt, an dem der Stapelüberlauf erfasst worden ist. Außerdem setzt die Verifikationseinrichtung anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**476**).

[0079] Wenn keine Stapelüberlaufbedingung erfasst wird, fügt die Verifikationseinrichtung einen Eintrag zum virtuellen Stapel hinzu (**478**), der den Typ der Daten (Operand) anzeigt, die auf den Operandenstapel geschoben werden sollen (während der wirklichen Ausführung des Programms) für das jeweilige vom aktuell ausge-

wählten Befehl auf den Stapel zu schiebende Datum. Diese Information wird aus den datentypspezifischen Opcodes abgeleitet, die im Bytecode-Programm der bevorzugten Ausführungsform der vorliegenden Erfindung verwendet werden. Die Verifikationseinrichtung aktualisiert ferner den Stapelzähler **342**, um den hinzugefügten Eintrag oder die Einträge im virtuellen Stapel wiederzugeben. Dies schließt den Stapelschiebe-Verifikationsprozess ab.

[0080] Wenn wie in **Fig.** 6E gezeigt der aktuell ausgewählte Befehl einen bedingten oder unbedingten Sprung oder eine Verzweigung vorwärts im Programm über die gewöhnliche sequentielle Schrittoperation hinaus hervorruft (Schritt **480**), prüft die Verifikationseinrichtung zuerst (**482**), ob ein Schnappschuss für den Zielort des Sprungbefehls in der Stapelschnappschuss-Speicherstruktur **346** gespeichert ist. Wenn kein Stapelschnappschuss gespeichert worden ist, wird die virtuelle Stapelkonfiguration (nach irgendwelchen virtuellen Stapelaktualisierungen, die dem Sprung zugeordnet sind) in der Stapelschnappschuss-Speicherstruktur **346** an einem Ort gespeichert (**484**), der dem Ziel-Programmort zugeordnet ist. Es ist zu beachten, dass beliebige Stapelabziehoperationen, die dem Sprung zugeordnet sind, bereits im virtuellen Stapel durch den vorher ausgeführten Schritt **460** wiedergegeben worden sind (siehe **Fig.** 6C).

[0081] Wenn ein Stapelschnappschuss gespeichert worden ist (was anzeigt, dass ein weiterer Einsprungpunkt, der diesem Zielbefehl zugeordnet ist, bereits verarbeitet worden ist), vergleicht die Verifikationseinrichtung anschließend (**486**) die virtuelle Stapelschnappschussinformation, die im Schnappschussabschnitt **350** der Stapelschnappschuss-Speicherstruktur **346** gespeichert ist, mit dem aktuellen Zustand des virtuellen Stapels. Wenn der Vergleich zeigt, dass der aktuelle Zustand und der Schnappschuss nicht übereinstimmen, wird anschließend eine Fehlernachricht erzeugt (**488**), die den Ort im Bytecode-Programm angibt, an dem der Stapelstatusfehler aufgetreten ist. In der bevorzugten Ausführungsform entsteht ein Fehler, wenn der aktuelle virtuelle Stapel und der Schnappschuss nicht dieselbe Anzahl und nicht dieselben Typen von Einträgen enthalten. Ferner entsteht ein Fehler, wenn einer oder mehrere Datentypwerte im aktuellen virtuellen Stapel nicht mit entsprechenden Datentypwerten im Schnappschuss übereinstimmen. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**490**). Wenn im Schritt **486** eine Stapelstatusübereinstimmung erfasst wird, fährt die Verifikationseinrichtung mit der Verarbeitung im Schritt **500** fort (**Fig.** 6F).

[0082] Wenn wie in **Fig.** 6F gezeigt der aktuell ausgewählte Befehl einen bedingten oder unbedingten Sprung oder eine Verzweigung rückwärts im Programm hervorruft (Schritt **500**), vergleicht anschließend die Verifikationseinrichtung die virtuelle Stapelschnappschussinformation, die im Schnappschussabschnitt **350** der Stapelschnappschuss-Speicherstruktur **346** gespeichert ist, die dem Ziel des Rückwärtssprungs zugeordnet ist (welche bereits im Schritt **436** gespeichert worden ist), mit dem aktuellen Zustand des virtuellen Stapels (**502**). Wenn der Vergleich zeigt, dass der aktuelle Zustand und der Schnappschuss nicht übereinstimmen, wird anschließend eine Fehlernachricht erzeugt (**504**), die den Ort im Bytecode-Programm angibt, an dem der Stapelstatusfehler aufgetreten ist. In der bevorzugten Ausführungsform entsteht ein Fehler, wenn der aktuelle virtuelle Stapel und der Schnappschuss nicht dieselbe Anzahl und dieselben Typen von Einträgen enthalten, oder wenn irgendein Datentypeintrag im aktuellen virtuellen Stapel nicht mit dem entsprechenden Datentypeintrag im Schnappschuss übereinstimmt. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**506**).

[0083] Wenn eine Stapelstatusübereinstimmung erfasst wird (im Schritt **502**), oder wenn der Befehl kein Rückwärtssprung ist (im Schritt **500**), setzt die Verifikationseinrichtung die Verarbeitung im Schritt **510** fort.

[0084] Wenn der aktuell ausgewählte Befehl Daten aus einer lokalen Variablen liest (**510**), vergleicht die Verifikationseinrichtung (**512**) die vorher in der entsprechenden virtuellen lokalen Variable gespeicherten Datentypcodeinformationen mit den Datentypanforderungen (falls vorhanden) des aktuell ausgewählten Befehls. Wenn keine Übereinstimmung zwischen der in der virtuellen lokalen Variablen gespeicherten Datentypinformation und den Datentypanforderungen des aktuell ausgewählten Befehls erfasst wird (**512**), wird eine Nachricht erzeugt (**514**), die den Ort im Bytecode-Programm angibt, an dem der Fehler aufgetreten ist. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**516**).

[0085] Wenn der aktuell ausgewählte Befehl keine Daten aus einer lokalen Variablen liest (**510**) oder der Datentypvergleich im Schritt **512** eine Übereinstimmung ergibt, fährt die Verifikationseinrichtung mit der Verarbeitung des aktuell ausgewählten Befehls im Schritt **520** fort.

[0086] Wenn wie in **Fig.** 6G gezeigt der aktuell ausgewählte Befehl Daten in einer lokalen Variablen speichert

(**520**), wird die entsprechende virtuelle lokale Variable untersucht (**522**), um zu ermitteln, ob sie einen Datentypwert speichert. Wenn die virtuelle lokale Variable einen Datentypwert speichert (was anzeigt, dass vorher Daten in der lokalen Variablen gespeichert worden sind), vergleicht die Verifikationseinrichtung die Datentypinformationen in der virtuellen lokalen Variablen mit dem Datentyp, der dem derzeit ausgewählten Bytecode-Befehl zugeordnet ist (**524**). Wenn zwischen der in der virtuellen lokalen Variablen gespeicherten Datentypinformation und den Datentypanforderungen des aktuell ausgewählten Befehls keine Übereinstimmung festgestellt wird (**524**), wird eine Nachricht erzeugt (**526**), die den Ort im Bytecode-Programm angibt, an dem der Fehler aufgetreten ist. Die Verifikationseinrichtung setzt anschließend einen Verifikationsstatuswert **245** für das Programm auf Falsch und bricht den Verifikationsprozess ab (**528**).

[0087] Wenn der aktuell ausgewählte Befehl keine Daten in einer lokalen Variablen speichert (**520**) ist die Verarbeitung für den aktuell ausgewählten Befehl abgeschlossen. Wenn der aktuell ausgewählte Befehl Daten in einer lokalen Variablen speichert, jedoch die virtuelle lokale Variable keinen Datentypwert enthält (was anzeigt, dass bisher kein Befehl von der Verifikationseinrichtung verarbeitet worden ist, der Daten in der lokalen Variablen speichern würde), wird der dem ausgewählten Bytecode-Befehl zugeordnete Datentyp in der virtuellen lokalen Variablen gespeichert (Schritt **530**).

[0088] Als Nächstes prüft die Verifikationseinrichtung (**540**), ob dies der letzte Befehl im Bytecode-Programm **340** ist, der zu verarbeiten ist. Wenn weitere Befehle zu verarbeiten sind, lädt die Verifikationseinrichtung (**542**) den nächsten Befehl und wiederholt den Verifikationsprozess beginnend bei Schritt **432**. Wenn keine weiteren Befehle zu verarbeiten sind, setzt die Verifikationseinrichtung den Verifikationsstatuswert **245** für das Programm auf Wahr (**544**), was den Abschluss des Verifikationsprozesses anzeigt.

Bytecode-Interpreter

[0089] Mit Bezug auf das Flussdiagramm in **Fig.** 7 und den Anhang 3 wird die Ausführungsform des Bytecode-Interpreters **242** beschrieben. Der Anhang 3 ist eine Auflistung einer Pseudocode-Darstellung des Bytecode-Interpreters.

[0090] Nachdem ein spezifiziertes Bytecode-Programm als ein auszuführendes Programm empfangen worden ist oder anderweitig ausgewählt worden ist (**560**), ruft der Bytecode-Programminterpreter **242** die Bytecode-Verifikationseinrichtung **240** auf (**562**), um die Integrität des spezifizierten Bytecode-Programms zu verifizieren. Die Bytecode-Verifikationseinrichtung ist oben beschrieben worden.

[0091] Wenn die Verifikationseinrichtung einen Wert "Verifikationsfehler" zurückgibt (**564**), wird der Versuch zum Ausführen des spezifizierten Bytecode-Programms vom Interpreter abgebrochen (**566**).

[0092] Wenn die Verifikationseinrichtung **242** einen Wert "Verifikationserfolg" zurückgibt (**564**), wird das spezifizierte Bytecode-Programm mit Betriebsmittelhilfsprogrammen und irgendwelchen anderen Programmen, Funktionen und Objekten, auf die vom Programm zugegriffen werden kann, verknüpft (**568**). Ein solcher Verknüpfungsschritt ist ein gewöhnlicher Vor-Ausführungsschritt in vielen Programminterpretern. Anschließend wird das verknüpfte Bytecode-Programm interpretiert und vom Interpreter ausgeführt (**570**). Der Bytecode-Interpreter der vorliegenden Erfindung führt während der Programmausführung keine Operandenstapelüberlauf- und Unterlaufprüfungen durch und führt ferner keine Datentypprüfung für die im Operandenstapel während der Programmausführung gespeicherten Daten durch. Diese herkömmlichen Stapelüberlauf-, Unterlauf- und Datentypprüfoperationen können von der vorliegenden Erfindung weggelassen werden, da die Verifikationseinrichtung bereits geprüft hat, dass Fehler dieses Typs während der Programmausführung nicht auftreten.

[0093] Der Programminterpreter der vorliegenden Erfindung ist insbesondere effizient für die Ausführung von Bytecode-Programmen mit Befehlsschleifen, die oft ausgeführt werden, da die Operandenstapelprüfbefehle nur einmal für jeden Bytecode in jeder solchen Befehlsschleife in der vorliegenden Erfindung ausgeführt werden. Im Gegensatz hierzu muss während der Ausführung eines Programms durch einen herkömmlichen Interpreter der Interpreter kontinuierlich den Operandenstapel auf Überläufe (d. h. Hinzufügen von mehr Daten zum Stapel, als der Stapel speichern kann) und Unterläufe (d. h. Versuch des Abziehens von Daten vom Stapel, wenn der Stapel leer ist) überwachen. Eine solche Stapelüberwachung muss normalerweise für alle Befehle ausgeführt werden, die den Stapelstatus ändern (was nahezu alle Befehle umfasst). Für viele Programme beanspruchen die Stapelüberwachungsbefehle, die vom Interpreter ausgeführt werden, ungefähr 80% der Ausführungszeit eines interpretierten Computerprogramms. Folglich führt der Interpreter der vorliegenden Erfindung Programme häufig zwei bis fünfmal schneller aus als ein herkömmlicher Programminterpreter, der auf demselben Computer läuft.

[0094] Die vorangehenden Beschreibungen spezifischer Ausführungsformen der vorliegenden Erfindung dienen der Erläuterung und der Beschreibung. Sie sollen nicht erschöpfend sein oder die Erfindung auf die genauen offenbarten Formen beschränken, wobei offensichtlich viele Abwandlungen und Veränderungen hinsichtlich der obigen Lehre möglich sind. Die Ausführungsformen wurden ausgewählt und beschrieben, um die Prinzipien der Erfindung und deren praktische Anwendung am besten zu erläutern, um somit anderen Fachleuten zu ermöglichen, die Erfindung und verschiedene Ausführungsformen mit verschiedenen Abwandlungen am besten zu nutzen, die für die beabsichtigte bestimmte Nutzung geeignet sind. Der Umfang der Erfindung soll durch die beigefügten Ansprüche und deren Äquivalente bestimmt sein.

TABELLE 1
BYTECODES IN OAK-SPRACHE

BEFEHLSNAME	KURZBESCHREIBUNG
aaload	Lade Objektreferenz aus Feld
aastore	speichere Objektreferenz in Objektreferenzfeld
aconst_null	schiebe Null-Objekt
aload	lade lokale Objektvariable
areturn	gebe Objektreferenz von Funktion zurück
arraylength	erhalte Länge des Feldes
astore	speichere Objektreferenz in lokaler Variable
astore_<n>	speichere Objektreferenz in lokale Variable
athrow	THROW-Ausnahme
bipush	schiebe vorzeichenbehaftete Ein-Byte-Ganzzahl
breakpoint	rufe Unterbrechungspunkthantierer auf
catchsetup	setze Ausnahmehantierer
catchteardown	setze Ausnahmehantierer zurück
checkcast	stelle sicher, dass Objekt von einem gegebenen Typ ist
d2f	setze Gleitkommazahl mit doppelter Genauigkeit in Gleitkommazahl mit einfacher Genauigkeit um
d2i	setze Gleitkommazahl mit doppelter Genauigkeit in Ganzzahl um
d2l	setze Gleitkommazahl mit doppelter Genauigkeit in lange Ganzzahl um
dadd	addiere Gleitkommazahlen mit doppelter Genauigkeit
daload	lade Gleitkommazahl mit doppelter Genauigkeit aus Feld
dastore	speichere Gleitkommazahl mit doppelter Genauigkeit in Feld
dcmpg	vergleiche zwei Gleitkommazahlen mit doppelter Genauigkeit (gebe 1 bei Ungleichheit zurück)
dcmpl	vergleiche zwei Gleitkommazahlen mit doppelter Genauigkeit (gebe –1 bei Ungleichheit zurück)
dconst_<d>	schiebe Gleitkommazahl mit doppelter Genauigkeit
ddiv	dividiere Gleitkommazahlen mit doppelter Genauigkeit
dload	lade Gleitkommazahl mit doppelter Genauigkeit aus lokaler Variablen

dload_<n>	lade Gleitkommazahl mit doppelter Genauigkeit aus lokaler Variablen
dmod	führe Modulofunktion für Gleitkommazahlen mit doppelter Genauigkeit aus
dmul	multipliziere Gleitkommazahlen mit doppelter Genauigkeit
dneg	negiere Gleitkommazahl mit doppelter Genauigkeit
dreturn	gebe Gleitkommazahl mit doppelter Genauigkeit von Funktion zurück
dstore	speichere Gleitkommazahl mit doppelter Genauigkeit in lokaler Variablen
dsub	subtrahiere Gleitkommazahlen mit doppelter Genauigkeit
dup	dupliziere oberstes Stapelwort
dup2	dupliziere die zwei obersten Stapelwörter
dup2_x1	dupliziere die zwei obersten Stapelwörter und lege zwei ab
dup2_x2	dupliziere die zwei obersten Stapelwörter und lege drei ab
dup_x1	dupliziere oberstes Stapelwort und lege zwei ab
dup_x2	dupliziere oberstes Stapelwort und fege drei ab
f2d	setze Gleitkommazahl mit einfacher Genauigkeit in Gleitkommazahl mit doppelter Genauigkeit um
f2i	setze Gleitkommazahl mit einfacher Genauigkeit in Ganzzahl um
f2l	setze Gleitkommazahl mit einfacher Genauigkeit in lange Ganzzahl um
fadd	addiere Gleitkommazahlen mit einfacher Genauigkeit
faload	lade Gleitkommazahl mit einfacher Genauigkeit aus Feld
fastore	speichere Gleitkommazahl mit einfacher Genauigkeit in Feld
fcmpg	vergleiche zwei Gleitkommazahlen mit einfacher Genauigkeit (gebe 1 bei Ungleichheit zurück)
fcmpl	vergleiche zwei Gleitkommazahlen mit einfacher Genauigkeit (gebe –1 bei Ungleichheit zurück)
fconst_<f>	schiebe Gleitkommazahl mit einfacher Genauigkeit
fdiv	dividiere Gleitkommazahlen mit einfacher Genauigkeit
fload	lade Gleitkommazahl mit einfacher Genauigkeit aus lokaler Variablen
fload_<n>	lade Gleitkommazahl mit einfacher Genauigkeit aus lokaler Variablen

fmod	führe Modulofunktion für Gleitkommazahlen mit einfacher Genauigkeit aus
fmul	multipliziere Gleitkommazahlen mit einfacher Genauigkeit
fneg	negiere Gleitkommazahl mit einfacher Genauigkeit
freturn	gebe Gleitkommazahl mit einfacher Genauigkeit von Funktion zurück
fstore	speichere Gleitkommazahl mit einfacher Genauigkeit in lokaler Variablen
fsub	subtrahiere Gleitkommazahlen mit einfacher Genauigkeit
getfield	hohle Feld von Objekt
getstatic	setzte statisches Feld von Klasse
goto	verzweige immer
i2d	konvertiere Ganzzahl in Gleitkommazahl mit doppelter Genauigkeit
i2f	konvertiere Ganzzahl in Gleitkommazahl mit einfacher Genauigkeit
i2l	konvertiere Ganzzahl zu langer Ganzzahl
iadd	addiere Ganzzahlen
iaload	lade Ganzzahl aus Feld
iand	boolesche UND-Verknüpung zweier Ganzzahlen
iastore	speichere Ganzzahl in Feld
iconst_<n>	schiebe Ganzzahl
iconst m1	schiebe Ganzzahlkonstante minus 1
idiv	Ganzzahldivision
if_acmpeq	verzweige, wenn Objekte gleich sind
if_acmpne	verzweige, wenn Objekte ungleich sind
if_icmpeq	verzweige, wenn Ganzzahlen gleich sind
if_icmpge	verzweige, wenn Ganzzahl größer oder gleich ist
if_icmpgt	verzweige, wenn Ganzzahl größer ist als
if_icmple	verzweige, wenn Ganzzahl kleiner oder gleich ist
if_icmplt	verzweige, wenn Ganzzahl kleiner ist als
if_icmpne	verzweige, wenn Ganzzahlen ungleich sind
ifeq	verzweige, wenn gleich 0
ifge	verzweige, wenn größer oder gleich 0

ifgt	verzweige, wenn größer als 0
ifle	verzweige, wenn kleiner oder gleich 0
iflt	verzweige, wenn kleiner als 0
ifne	verzweige, wenn ungleich 0
iinc	inkrementiere lokale Variable mit Konstante
iload	lade Ganzzahl aus lokaler Variable
iload_<n>	lade Ganzzahl aus lokaler Variable
imod	führe Modulofunktion mit Ganzzahlen aus
imul	multipliziere Ganzzahlen
ineg	negiere Ganzzahl
instanceof	ermittle, ob Objekt von einem gegebenen Typ ist
int2byte	konvertiere Ganzzahl in vorzeichenbehaftetes Byte
int2char	konvertiere Ganzzahl in Zeichen
invokeinterface	aktiviere Schnittstellenverfahren
invokemethod	aktiviere Klassenverfahren
invokesuper	aktiviere Superklassenverfahren
ior	bolesche ODER-Verknüpfung zweier Ganzzahlen
ireturn	gebe Ganzzahl von Funktion zurück
ishl	schiebe Ganzzahl nach links
ishr	arithmetische Rechtsverschiebung einer Ganzzahl
istore	speichere Ganzzahl in lokaler Variable vindex
istore_<n>	speichere Ganzzahl in lokaler Variable n
isub	subtrahiere Ganzzahlen
iushr	logische Rechtsverschiebung einer Ganzzahl
ixor	boolesche Exklusiv-ODER-Verknüpfung zweier Ganzzahlen
jsr	Sprung zu Unterroutine
l2d	konvertiere lange Ganzzahl in Gleitkommazahl mit doppelter Genauigkeit
l2f	konvertiere lange Ganzzahl in Gleitkommazahl mit einfacher Genauigkeit
l2i	konvertiere lange Ganzzahl in Ganzzahl
ladd	addiere lange Ganzzahlen
laload	lade lange Ganzzahl aus Feld
land	boolesche UND-Verknüpung zweier langer Ganzzahlen
lastore	speichere lange Ganzzahl in Feld

lcmp	vergleiche lange Ganzzahlen
lconst_<|>	schiebe lange Ganzzahlkonstante
ldc1	schiebe Element aus Konstanten-Pool
ldc2	schiebe Element aus Konstanten-Pool
ldc2w	schiebe langes oder doppeltes Element aus Konstanten-Pool
ldiv	dividiere lange Ganzzahlen
lload	lade lange Ganzzahl aus lokaler Variable
lload_<n>	lade lange Ganzzahl aus lokaler Variable
lmod	führe Modulofunktion mit langen Ganzzahlen aus
lmul	multipliziere lange Ganzzahlen
lneg	negiere lange Ganzzahl
lookupswitch	greife auf Sprungtabelle zu mittels Schlüsselvergleich und springe
lor	bolesche ODER-Verknüpfung zweier langer Ganzzahlen
leeturn	gebe lange Ganzzahl von Funktion zurück
lshl	schiebe lange Ganzzahl nach links
lshr	arithmetische Rechtsverschiebung einer langen Ganzzahl
lstore	speichere lange Ganzzahl in lokaler Variable vindex
lstore_<n>	speichere lange Ganzzahl in lokaler Variable n
lsub	subtrahiere lange Ganzzahlen
lushr	logische Rechtsverschiebung einer langen Ganzzahl
lxor	boolesche Exklusiv-ODER-Verknüpfung zweier langer Ganzzahlen
monitorenter	gehe in überwachten Bereich des Codes über
monitorexit	verlasse überwachten Bereich des Codes
new	erzeuge neues Objekt
newarray	weise neues Feld zu
newfromname	erzeuge neues Objekt aus Name
nop	tue nichts
pop	ziehe oberstes Stapelwort ab
pop2	ziehe die zwei obersten Stapelwörter ab
putfield	setze Feld in Objekt
putstatic	setze statisches Feld in Klasse
ret	Rücksprung von Unterroutine
return	Rücksprung (leer) von Prozedur
saload	lade vorzeichenbehaftetes Byte aus Feld
sastore	speichere vorzeichenbehaftetes Byte in Feld
siaload	lade vorzeichenloses kurzes Element aus Feld
siastore	speichere vorzeichenloses kurzes Element in Feld
sipush	schiebe vorzeichenbehaftete 2-Byte-Ganzzahl
tableswitch	greife auf Sprungtabelle mittels Index zu und springe
verifystack	verifiziere, ob Stapel leer ist

ANHANG 1

Pseudocode für Klassenlader

[0095] Anwender wählt ein Objekt (das "bezeichnete Objekt") zum Betrachten aus. (Zum Beispiel kann die Anwenderauswahl durchgeführt werden durch Auswählen eines Hyperlinks auf das Objekt in einem Dokument oder einem weiteren Objekt.)

[0096] Öffnen der Verbindung zu dem Server, der das bezeichnete Objekt speichert. Empfangen eines Hantierers für das bezeichnete Objekt, der den Datentyp enthält.

[0097] Prüfen, ob der Datentyp dem System des Anwenders bekannt ist (d. h., ob der Anwender einen Betrachter für Objekte des empfangenen Datentyps besitzt)

[0098] Wenn Datentyp unbekannt ist

```
{
Öffnen einer zweiten Verbindung zum selben Server
Anfordern eines Betrachters für den spezifizierten Datentyp:
Wenn erfolgreich
    /* Hybridsystem-Option: prüfe auf Nicht-Bytecode-Betrachter */
    {
    Wenn empfangener Betrachter kein Bytecode-Programm ist
        {
        Ermittle, ob Betrachter akzeptiert werden soll
        Wenn Betrachter nicht akzeptiert wird
            Lösche empfangenen Betrachter
        Sonst /* Nicht-Bytecode-Betrachter wird akzeptiert */
            Gehe zu Beende-Objekt-Herunterladen
        }
    /* Verifikations- und Registrationsprozedur */
    Sonst
        {
        Führe Bytecode-Verifikationseinrichtung für empfangenen Be-
            trachter aus
        Wenn Verifizierung erfolgreich ist
            {
```

```
                    /* Registration des Betrachters */
                    Markiere empfangenen Betrachter als verifiziert
                    Speichere Betrachter in lokaler Betrachterbibliothek
                    Füge Datentyp zur Liste bekannter Datentypen hinzu
                    }
                Sonst
                    Lösche empfangenen Betrachter
                    }
                }
        Wenn Datentyp immer noch unbekannt ist
            {
            Suche andere Serverseiten nach Betrachter für spezifizierten Datentyp
                ab
            Wenn erfolgreich
                {dieselbe Verifikations- und Registrationsprozedur wie oben}
            }
        Wenn Datentyp immer noch unbekannt ist
            {
            Breche Herunterladen des bezeichneten Objekts ab.
            Informiere Anwender, dass Betrachter für bezeichnetes Objekt nicht
                gefunden werden kann.
            }
        }
    Beende-Objekt-Herunterladen: /* verzweige zu diesem Punkt für Nicht-
    Bytecode-Betrachter */
    Schließe Herunterladen des bezeichneten Objekts ab.
Wenn heruntergeladenes Objekt eingebettete Bytecode-Programme enthält
    {
    Führe Bytecode-Verifikationseinrichtung für eingebettete Programme aus
    Wenn Verifizierung erfolgreich ist
        Markiere eingebettete Programme als verifiziert
    Sonst
        {
        Lösche bezeichnetes Objekt
        Breche Herunterladeprozedur ab
        }
    }
Betrachte bezeichnetes Objekt mit Betrachter für Datentyp, der dem bezeichneten
    Objekt zugeordnet ist.
```

ANHANG 2

Pseudocode für OAK-Bytecode-Verifikationseinrichtung

[0099] Empfange Bytecode-Programm, das verifiziert werden soll.

[0100] Erzeuge virtuelle Operandenstapel-Datenstruktur zum Speichern von Stapelstatusinformationen und ein virtuelles lokales Variablenfeld zum Speichern lokaler Variablendatentypinformationen.

[0101] Erzeuge Datenstruktur zum Speichern virtueller Stapelschnappschüsse.

[0102] Erster Durchlauf durch Bytecode-Programm:
Lokalisiere alle Befehle, die Ziele bedingter und unbedingter Sprünge oder Verzweigungen sind (d. h. von mehr als einem vorangehenden Befehl erreicht werden können).
Speichere Liste solcher Zielbefehle in virtueller Stapelschnappschuss-Datenstruktur.

[0103] Zweiter Durchlauf durch Bytecode-Programm:
Setze Verifikationserfolg auf Wahr
Führe aus bis letzter Bytecode-Befehl verarbeitet worden ist:

```
{
Wähle nächsten Bytecode-Befehl (in sequentieller Reihenfolge im Pro-
        gramm)
Wenn Befehl in Liste der Zielbefehle ist
      {
      Wenn Schnappschuss des virtuellen Stapels für diesen Befehl be-
            reits existiert
          {
          Vergleiche aktuellen Status des virtuellen Stapels mit ge-
                speichertem Schnappschuss
          Wenn Schnappschuss nicht mit aktuellem virtuellen Stapel-
                zustand übereinstimmt
              {
              Drucke Nachricht, die den Ort im Programm angibt, an
                    dem der Stapelfehler aufgetreten ist
              Breche Verifizierung ab
              Setze Verifikationserfolg auf Falsch
              Rücksprung
```

```
                }
            }
        Sonst
            Speichere Schnappschuss des aktuellen virtuellen Stapel-
                status
        }
Fall (Befehlstyp):
        {
        Fall = Befehl zieht Daten vom Operandenstapel
            {
            Prüfe auf Stapelunterlauf
            Wenn Stapelunterlauf auftritt
                {
                Drucke Nachricht, die den Ort im Programm angibt, an
                    dem der Unterlauf aufgetreten ist
                Breche Verifizierung ab
                Rücksprung
                }
            Vergleiche Datentyp jedes vom Stapel gezogenen Operan-
                den mit dem erforderlichen Datentyp (falls vorhan-
                den) vom Bytecode-Befehl
            Wenn Typ nicht übereinstimmt
                {
                Drucke Nachricht, die den Ort im Programm angibt, an
                    dem der Datentypfehler aufgetreten ist
                Setze Verifikationserfolg auf Falsch
                }
            Lösche Informationen vom virtuellen Stapel für die abgezo-
                genen Operanden
            Aktualisiere Stapelzähler
            }
        Fall = Befehl schiebt Daten auf Operandenstapel
            {
            Prüfe auf Stapelüberlauf
            Falls Stapel übergelaufen ist
```

```
        {
        Drucke Nachricht, die den Ort im Programm angibt, an
                dem der Überlauf aufgetreten ist
        Breche Verifizierung ab
        Setze Verifikationserfolg auf Falsch
        Rücksprung
        }
    Füge Informationen zum virtuellen Stapel hinzu, die den Da-
            tentyp der auf den Operandenstapel geschobenen
            Daten anzeigen
    Aktualisiere Stapelzähler
    }
Fall = Befehl ist ein Vorwärtssprung- oder Verzweigungsbefehl
    {
    Wenn Schnappschuss des virtuellen Stapels für den Zielbe-
            fehl bereits existiert
        {
        Vergleiche aktuellen Status des virtuellen Stapels mit
                gespeichertem Schnappschuss
        Wenn Schnappschuss nicht mit aktuellem virtuellen
                Stapelzustand übereinstimmt
            {
            Drucke Nachricht, die den Ort im Programm an-
                    gibt, an dem der Stapelfehler aufgetreten
                    ist
            Breche Verifizierung ab
            Setze Verifikationserfolg auf Falsch
            Rücksprung
            }
        }
    Sonst
        Speichere Schnappschuss des aktuellen virtuellen Sta-
                pelzustands als Schnappschuss für den Ziel-
                befehl;
    }
```

```
Fall = Befehl ist ein Schleifenende-Rücksprung oder ein anderer
       Rückwärtssprung- oder Verzweigungsbefehl:
    {
    Vergleiche aktuellen virtuellen Stapelzustand mit gespei-
          chertem Schnappschuss für Zielbefehl
    Wenn aktueller virtueller Stapelzustand nicht mit gespei-
          chertem Schnappschuss übereinstimmt
        {
        Drucke Nachricht, die den Ort im Programm abgibt, an
              dem der Stapelfehler aufgetreten ist
        Breche Verifizierung ab
        Setze Verifikationserfolg auf Falsch
        Rücksprung
        }
    }
Fall = Befehl liest Daten aus lokaler Variable
    {
    Vergleiche Datentyp jedes von der lokalen Variablen gelese-
          nen Datums mit dem vom Bytecode-Befehl gefor-
          derten Datentyp (falls vorhanden)
    Wenn Typ nicht übereinstimmt
        {
        Drucke Nachricht, die den Ort im Programm angibt, an
              dem der Datentypfehler aufgetreten ist
        Setze Verifikationserfolg auf Falsch
        }
    }
Fall = Befehl speichert Daten in einer lokalen Variablen
    {
    Wenn entsprechende virtuelle lokale Variable bereits einen
          Datentypwert speichert
        {
        Vergleiche den in der virtuellen lokalen Variablen ge-
              speicherten Datentypwert mit dem Datentyp
              des Datums, das in der entsprechenden lokalen
```

```
                    Variablen gespeichert werden würde (wie von
                    dem vom aktuellen Bytecode-Befehl gehand-
                    habten Datentyp ermittelt)
                Wenn Typ nicht übereinstimmt
                    {
                    Drucke Nachricht, die den Ort im Programm an-
                        gibt, an dem der Datentypfehler aufgetre-
                        ten ist
                    Setze Verifikationserfolg auf Falsch
                    }
                }
            Sonst
                Füge Informationen zur virtuellen lokalen Variablen
                    hinzu, die den Datentyp der Daten anzeigen,
                    die in der entsprechenden lokalen Variablen
                    gespeichert werden würden
            }
        }/* Ende-Fall */
    }/* Ende der Ausführungsschleife */
Rücksprung (Verifikationserfolg)
```

ANHANG 3

Pseudocode für Bytecode-Interpreter

[0104] Empfange spezifiziertes Bytecode-Programm, das ausgeführt werden soll Rufe Bytecode-Verifikationseinrichtung auf, um spezifiziertes Bytecode-Programm zu verifizieren

[0105] Wenn Verifikationserfolg

```
{
Verknüpfe spezifiziertes Bytecode-Programm mit Betriebsmittelhilfspro-
    grammen.
Interpretiere Befehle des spezifizierten Bytecode-Programms und führe sie
    aus, ohne Operandenstapelüberlauf- und Unterlaufprüfungen durch-
    zuführen und ohne Datentypprüfungen für die im Operandenstapel
    gespeicherten Operanden durchzuführen.
}
```

Patentansprüche

1. Computer (**102**) für die Verwendung zusammen mit anderen Computern (**104**), der umfasst:
einen Speicher (**112**) zum Speichern von Objektanwendungsprogrammen (**103**), wobei jedes Objektanwendungsprogramm einem Anwender des Computers ermöglicht, Objekte eines zugeordneten Datentyps zu betrachten;
ein Anwenderschnittstellen-Steuerprogramm (**108**), das dem Anwender ermöglicht, Referenzen auf Objekte einschließlich in den anderen Computern gespeicherter Objekte auszuwählen; und

ein Computerzwischenverbindungs-Steuerprogramm, das in Reaktion auf eine Anwenderauswahl einer der Objektreferenzen, die auf eines der in den anderen Computern gespeicherten Objekte Bezug nimmt, eine erste Kommunikationsverbindung (**304**) mit einem der anderen Computer herstellt, wobei der eine Computer in Übereinstimmung mit Informationen in der ausgewählten Objektreferenz identifiziert wird, und eine Wiedergewinnung des Objekts, auf das durch die ausgewählte Objektreferenz Bezug genommen wird, von dem einen Computer einschließlich der Wiedergewinnung von Datentypinformationen, die dem Objekt, auf das Bezug genommen wird, zugeordnet sind, beginnt;
wobei das Anwenderschnittstellen-Steuerprogramm (**108**) enthält:
Suchbefehle, die feststellen, ob die in dem Computer gespeicherten Objektanwendungsprogramme (**103**) ein Objektanwendungsprogramm enthalten, das dem wiedergewonnenen Datentyp (**308**) zugeordnet ist, wobei sie bei negativer Feststellung versuchen, ein dem wiedergewonnenen Datentyp zugeordnetes Objektanwendungsprogramm in dem einen Computer zu lokalisieren; und einen Klassenlader, der Herunterladebefehle enthält, die eine Kopie des dem wiedergewonnenen Datentyp zugeordneten Objektanwendungsprogramms in den Speicher des Computers laden, wenn die Suchbefehle in dem einen Computer das dem wiedergewonnenen Datentyp (**308**) zugeordnete Betrachtungsprogramm lokalisieren.

2. Computer nach Anspruch 1, der ferner umfasst:
eine Verifikationsprozedur, die feststellt, ob das kopierte Objektanwendungsprogramm vorgegebene Programmkorrektheitskriterien einschließlich im Voraus definierter Operandenstapel-Verwendungskriterien erfüllt, und
Programmfreigabebefehle, die die Ausführung des kopierten Objektanwendungsprogramms freigeben, um dem Anwender zu ermöglichen, das Objekt, auf das Bezug genommen wird, zu betrachten, wenn die Verifikationsprozedur feststellt, dass das kopierte Objektanwendungsprogramm die vorgegebenen Programmkorrektheitskriterien erfüllt.

3. Computer nach Anspruch 1, bei dem das Computerzwischenverbindungs-Steuerprogramm Befehle enthält, die eine zweite Kommunikationsverbindung mit dem einen Computer herstellen, um die Kopie des dem wiedergewonnenen Datentyp zugeordneten Objektanwendungsprogramms herunterzuladen.

4. Verfahren zum Betreiben eines ersten Computers (**102**) in einem verteilten Computersystem, das mehrere verschiedene Computer (**102**, **104**) besitzt, wobei das Verfahren die folgenden Schritte umfasst:
a) in einem ersten Computer Speichern von Objektanwendungsprogrammen (**103**), wovon jedes einem Anwender hiervon ermöglicht, Daten in Objekten eines zugeordneten Datentyps zu verwenden;
b) im ersten Computer Ermöglichen, dass ein Anwender Daten in einem ersten Objekt, das einen zugeordneten ersten Datentyp besitzt, unter Verwendung eines ersten der Objektanwendungsprogramme (**103**) verwenden kann; wobei die Daten in dem ersten Objekt Verbindungsdaten enthalten, die auf ein zweites Objekt eines zweiten Datentyps Bezug nehmen, wobei die Verbindungsdaten einen zweiten Computer identifizieren, in dem das zweite Objekt abgelegt ist;
c) im ersten Computer Ermöglichen, dass der Anwender die Verbindungsdaten auswählt;
d) Herstellen einer ersten Kommunikationsverbindung (**304**) zwischen dem ersten Computer und dem zweiten Computer durch den ersten Computer in Reaktion auf eine Anwenderauswahl der Verbindungsdaten und Beginnen einer Wiedergewinnung des zweiten Objekts von dem zweiten Computer einschließlich einer Wiedergewinnung von Datentypinformationen, die dem zweiten Objekt zugeordnet sind;
e) durch den ersten Computer Feststellen, ob die in dem ersten Computer gespeicherten Objektanwendungsprogramme (**103**) ein dem zweiten Datentyp zugeordnetes Objektanwendungsprogramm enthalten;
f) wenn die Feststellung im Schritt (e) negativ ist, Feststellen, ob ein dem zweiten Datentyp zugeordnetes Objektanwendungsprogramm in dem zweiten Computer (**314**) gespeichert ist;
g) wenn die Feststellung im Schritt (g) positiv ist, Laden (**315**) einer Kopie des dem zweiten Datentyp zugeordneten Objektanwendungsprogramms in den ersten Computer.

5. Verfahren nach Anspruch 4, das ferner die folgenden Schritte umfasst:
wenn die Feststellung im Schritt (f) negativ ist, Feststellen, ob ein dem zweiten Datentyp zugeordnetes Objektanwendungsprogramm in irgendeinem Computer einer im Voraus definierten Gruppe anderer Computer gespeichert ist, wobei bei positiver Feststellung der Schritt g ausgeführt wird.

6. Verfahren nach Anspruch 4, das ferner die folgenden Schritte umfasst:
nach der Ausführung des Schrittes (g) Ausführen einer Verifikationsprozedur an der geladenen Kopie des Objektanwendungsprogramms, um festzustellen, ob die geladene Kopie des Objektanwendungsprogramms vorgegebene Operandenstapel-Verwendungskriterien erfüllt, und Sperren der Ausführung der geladenen Kopie des Objektanwendungsprogramms, falls die Verifikationsprozedur eine negative Feststellung macht.

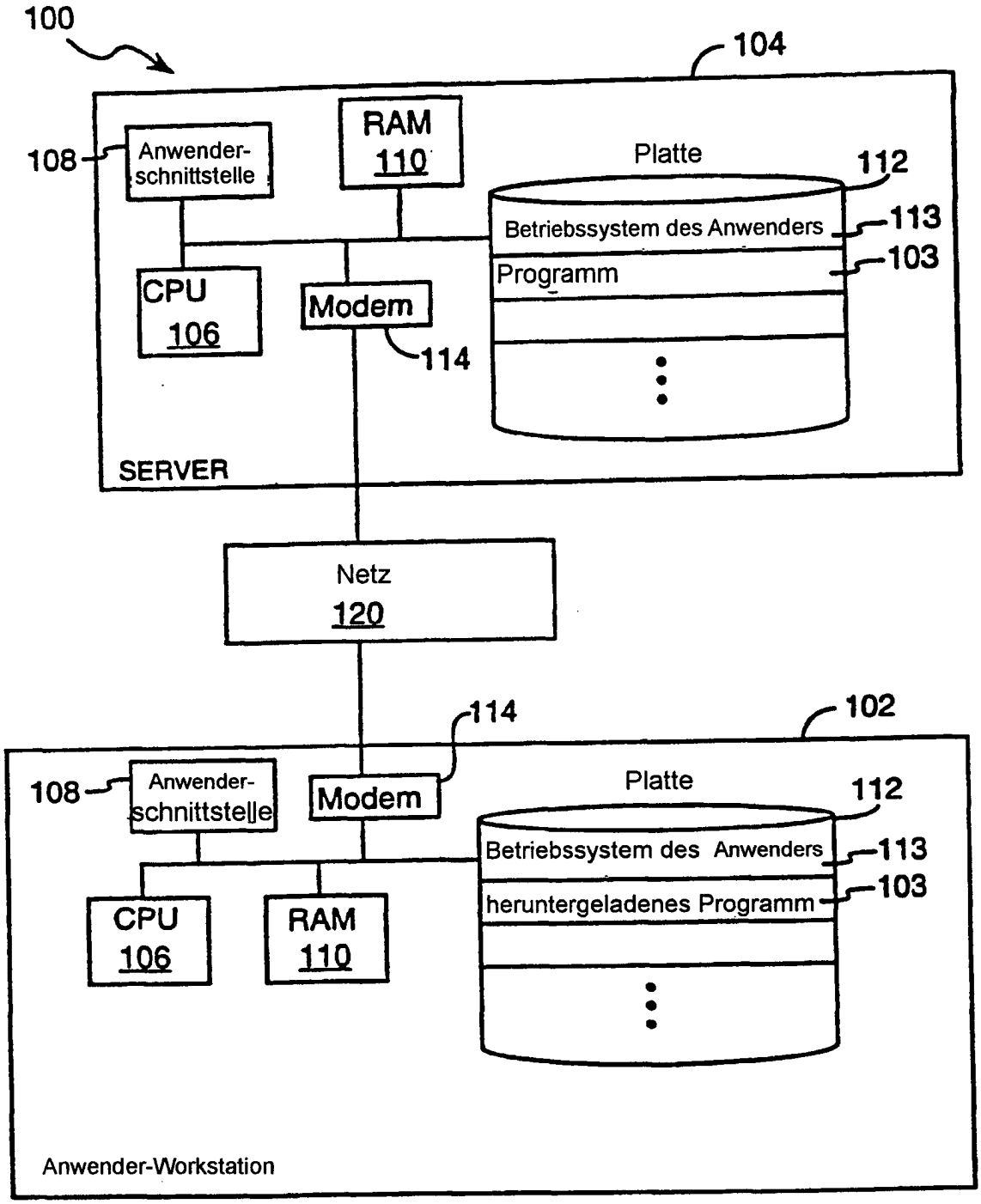
7. Computerprogrammprodukt für die Verwendung zusammen mit einer Vorrichtung, die einen Datenprozessor enthält, wobei das Computerprogrammprodukt ein computerlesbares Speichermedium und einen darin eingebetteten Computerprogrammmechanismus umfasst, wobei der Computerprogrammmechanismus umfasst:
ein erstes Programm, das eine Auswahl von Referenzen auf Objekte einschließlich in anderen Vorrichtungen gespeicherter Objekte ermöglicht; und
ein Vorrichtungszwischenverbindungs-Steuerprogramm (**304**), das in Reaktion auf die Auswahl einer der Objektreferenzen, die auf eines der in den anderen Vorrichtungen gespeicherten Objekte Bezug nimmt, eine erste Kommunikationsverbindung (**304**) mit einer der anderen Vorrichtungen herstellt, wobei die eine Vorrichtung in Übereinstimmung mit Informationen in der ausgewählten Objektreferenz identifiziert wird, und eine Wiedergewinnung des Objekts, auf das durch die ausgewählte Objektreferenz Bezug genommen wird, von der einen Vorrichtung einschließlich wiedergewonnener Datentypinformationen, die dem Objekt, auf das Bezug genommen wird, zugeordnet sind, beginnt;
Suchbefehle, die feststellen, ob die Vorrichtung ein Objektanwendungsprogramm (**103**) speichert, das dem wiedergewonnenen Datentyp zugeordnet ist, wobei sie bei negativer Feststellung versuchen, ein Objektanwendungsprogramm, das dem wiedergewonnenen Datentyp zugeordnet ist, in der einen Vorrichtung zu lokalisieren (**314**); und
einen Klassenlader, der Herunterladebefehle enthält, um eine Kopie des Objektanwendungsprogramms, das dem wiedergewonnenen Datentyp zugeordnet ist, in die Vorrichtung zu laden, wenn die Suchbefehle in der einen Vorrichtung das Objektanwendungsprogramm, das dem wiedergewonnenen Datentyp zugeordnet ist, lokalisieren.

8. Computerprogrammprodukt nach Anspruch 7, das ferner umfasst:
eine Verifikationsprozedur, die feststellt, ob das kopierte Objektanwendungsprogramm vorgegebene Programmkorrektheitskriterien einschließlich im Voraus definierter Operandenstapel-Verwendungskriterien erfüllt, und
Programmfreigabebefehle, die die Ausführung des kopierten Objektanwendungsprogramms freigeben, um dem Anwender zu ermöglichen, das Objekt, auf das Bezug genommen wird, zu betrachten, wenn die Verifikationsprozedur feststellt, dass das kopierte Objektanwendungsprogramm die vorgegebenen Programmkorrektheitskriterien erfüllt.

9. Computerprogrammprodukt nach Anspruch 7, bei dem das Vorrichtungszwischenverbindungs-Steuerprogramm Befehle enthält, die eine zweite Kommunikationsverbindung mit der einen Vorrichtung herstellen, um die Kopie des Objektanwendungsprogramms, das dem wiedergewonnenen Datentyp zugeordnet ist, herunterzuladen.

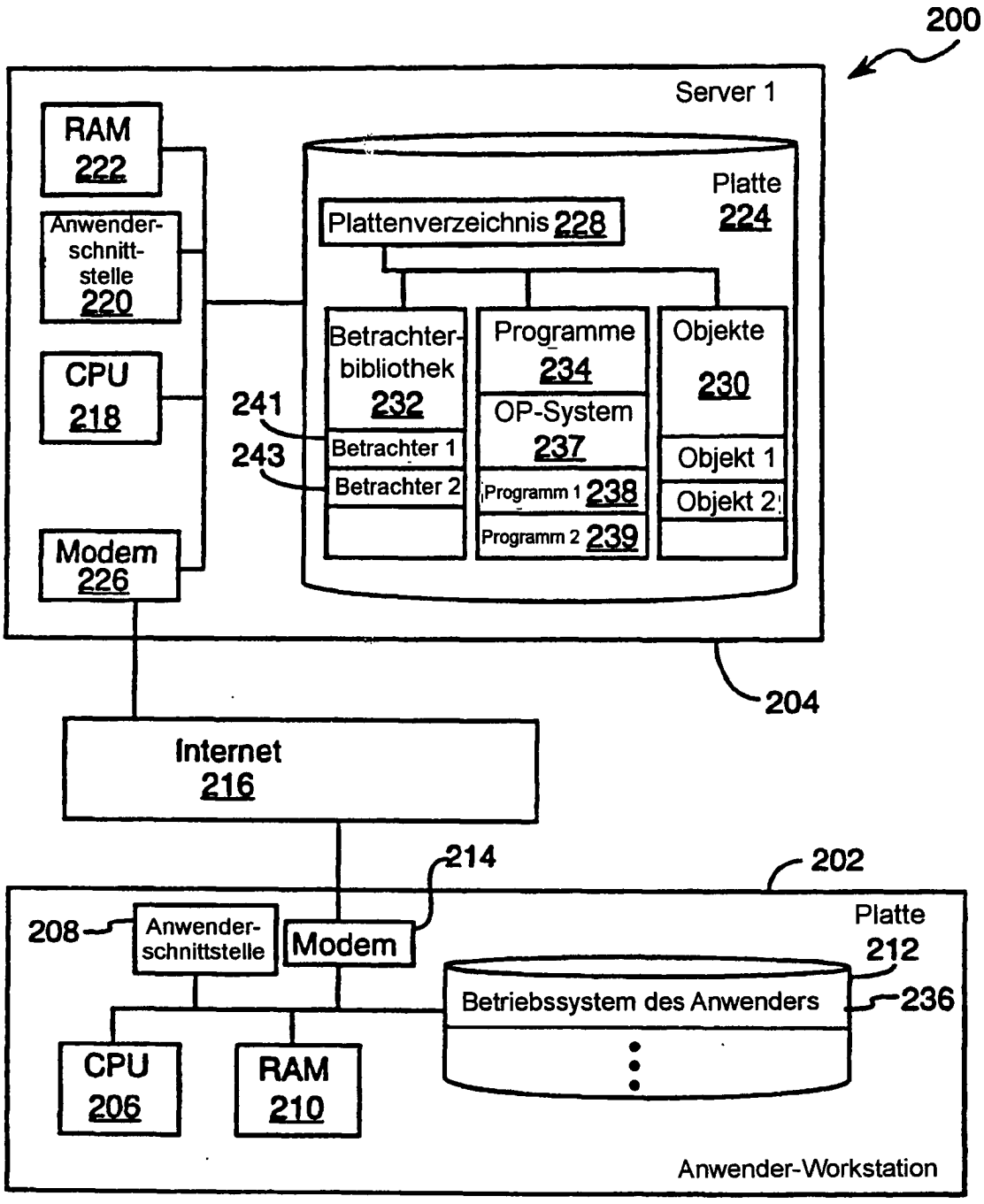
Es folgen 13 Blatt Zeichnungen

Anhängende Zeichnungen



Stand der Technik

Fig. 1



Stand der Technik

Fig. 2

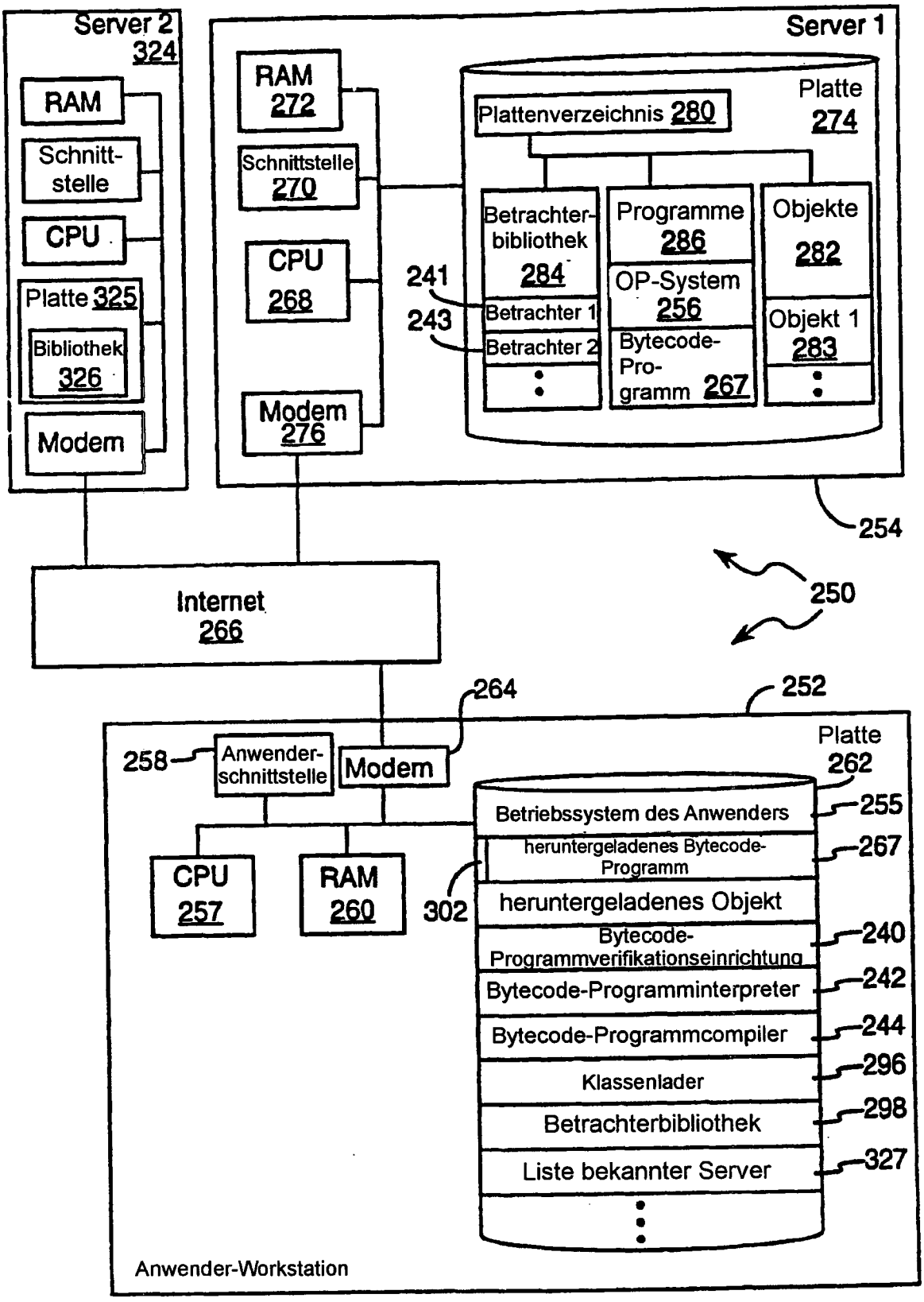

Server 2
324
RAM
Schnitt-
stelle
CPU
Platte 325
Bibliothek
326
Modem
Server 1
RAM
272
Schnittstelle
270
CPU
268
Modem
276
Platte
274
Plattenverzeichnis 280
Betrachter-
bibliothek
284
Programme
286
Objekte
282
241
243
Betrachter 1
Betrachter 2
OP-System
256
Bytecode-
Pro-
gramm 267
Objekt 1
283
254
250
Internet
266
264
252
258
Anwender-
schnittstelle
Modem
Platte
262
CPU
257
RAM
260
302
Betriebssystem des Anwenders
255
heruntergeladenes Bytecode-
Programm
267
heruntergeladenes Objekt
Bytecode-
Programmverifikationseinrichtung
240
Bytecode-Programminterpreter
242
Bytecode-Programmcompiler
244
Klassenlader
296
Betrachterbibliothek
298
Liste bekannter Server
327
Anwender-Workstation

Fig. 3

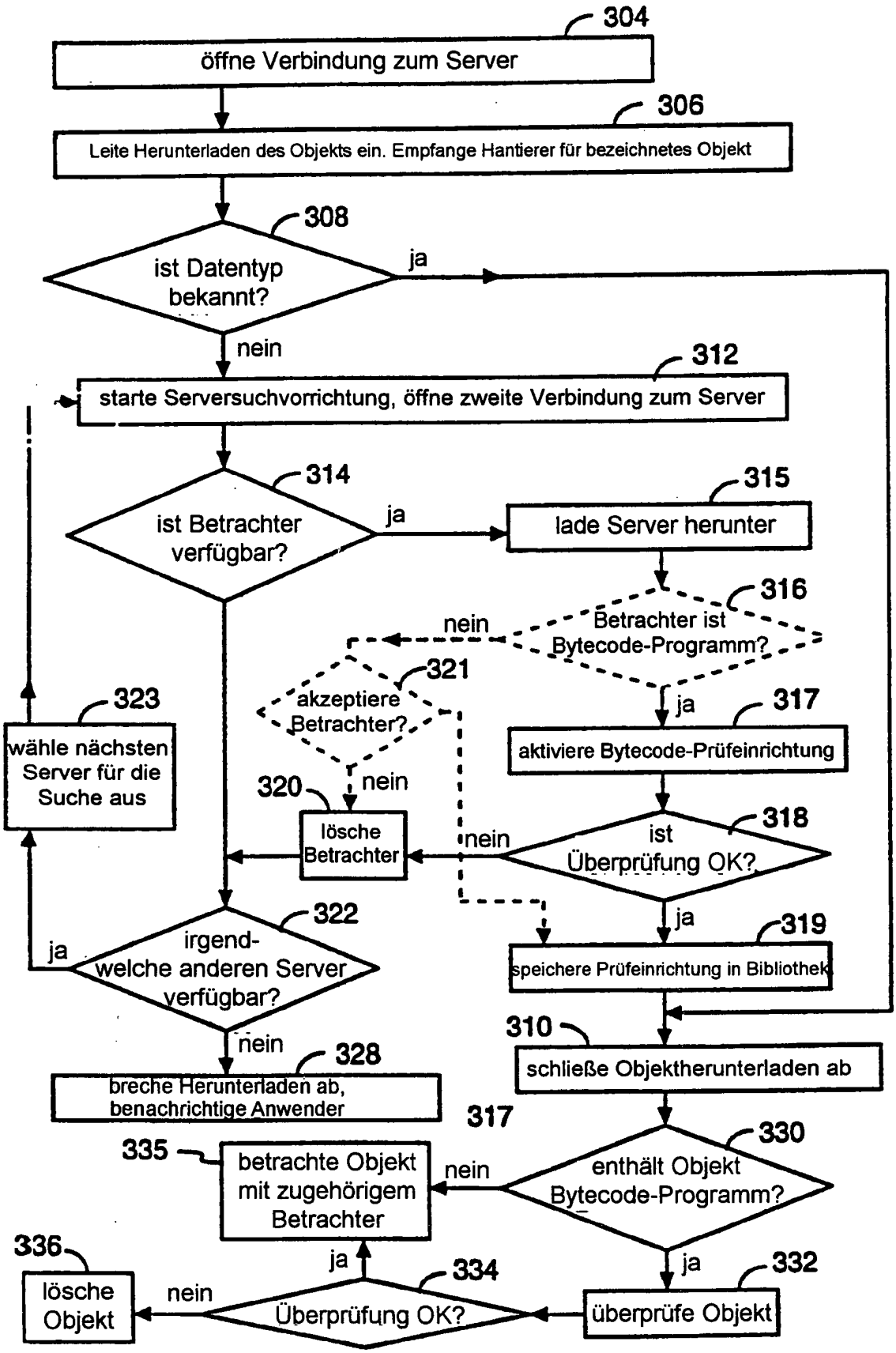

304
öffne Verbindung zum Server
306
Leite Herunterladen des Objekts ein. Empfange Hantierer für bezeichnetes Objekt
308
ist Datentyp bekannt?
ja
nein
312
starte Serversuchvorrichtung, öffne zweite Verbindung zum Server
314
ist Betrachter verfügbar?
ja
315
lade Server herunter
316
Betrachter ist Bytecode-Programm?
nein
ja
321
akzeptiere Betrachter?
nein
317
aktiviere Bytecode-Prüfeinrichtung
323
wähle nächsten Server für die Suche aus
320
lösche Betrachter
318
ist Überprüfung OK?
nein
ja
319
speichere Prüfeinrichtung in Bibliothek
322
irgend-welche anderen Server verfügbar?
ja
nein
310
schließe Objektherunterladen ab
328
breche Herunterladen ab, benachrichtige Anwender
317
330
enthält Objekt Bytecode-Programm?
335
betrachte Objekt mit zugehörigem Betrachter
nein
ja
336
lösche Objekt
nein
334
Überprüfung OK?
ja
332
überprüfe Objekt

Fig. 4

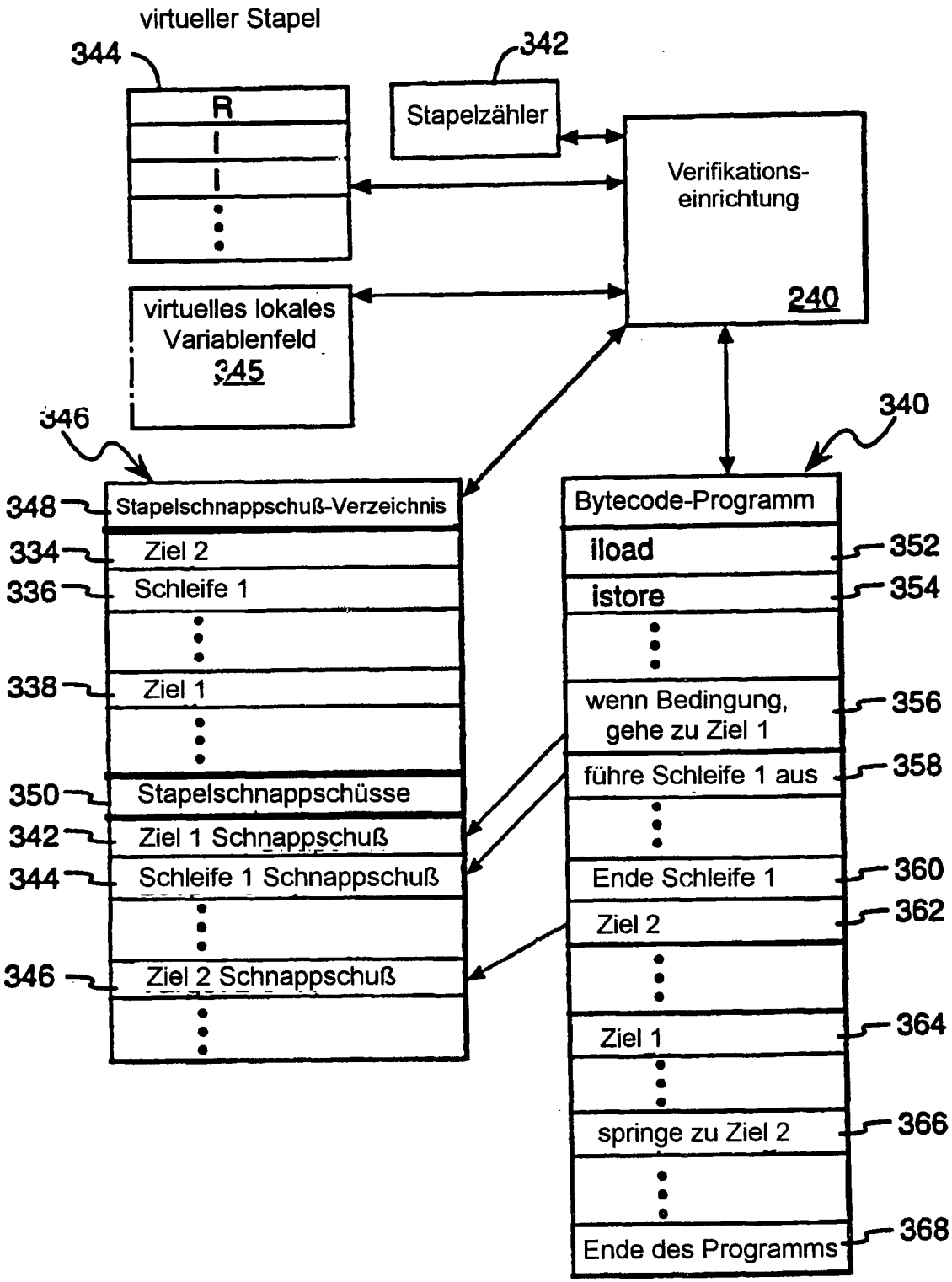

virtueller Stapel
344
342
R
Stapelzähler
Verifikations-
einrichtung
240
virtuelles lokales
Variablenfeld
345
346
340
348
Stapelschnappschuß-Verzeichnis
Bytecode-Programm
334
Ziel 2
iload
352
336
Schleife 1
istore
354
338
Ziel 1
wenn Bedingung,
gehe zu Ziel 1
356
350
Stapelschnappschüsse
führe Schleife 1 aus
358
342
Ziel 1 Schnappschuß
344
Schleife 1 Schnappschuß
Ende Schleife 1
360
Ziel 2
362
346
Ziel 2 Schnappschuß
Ziel 1
364
springe zu Ziel 2
366
Ende des Programms
368

Fig. 5

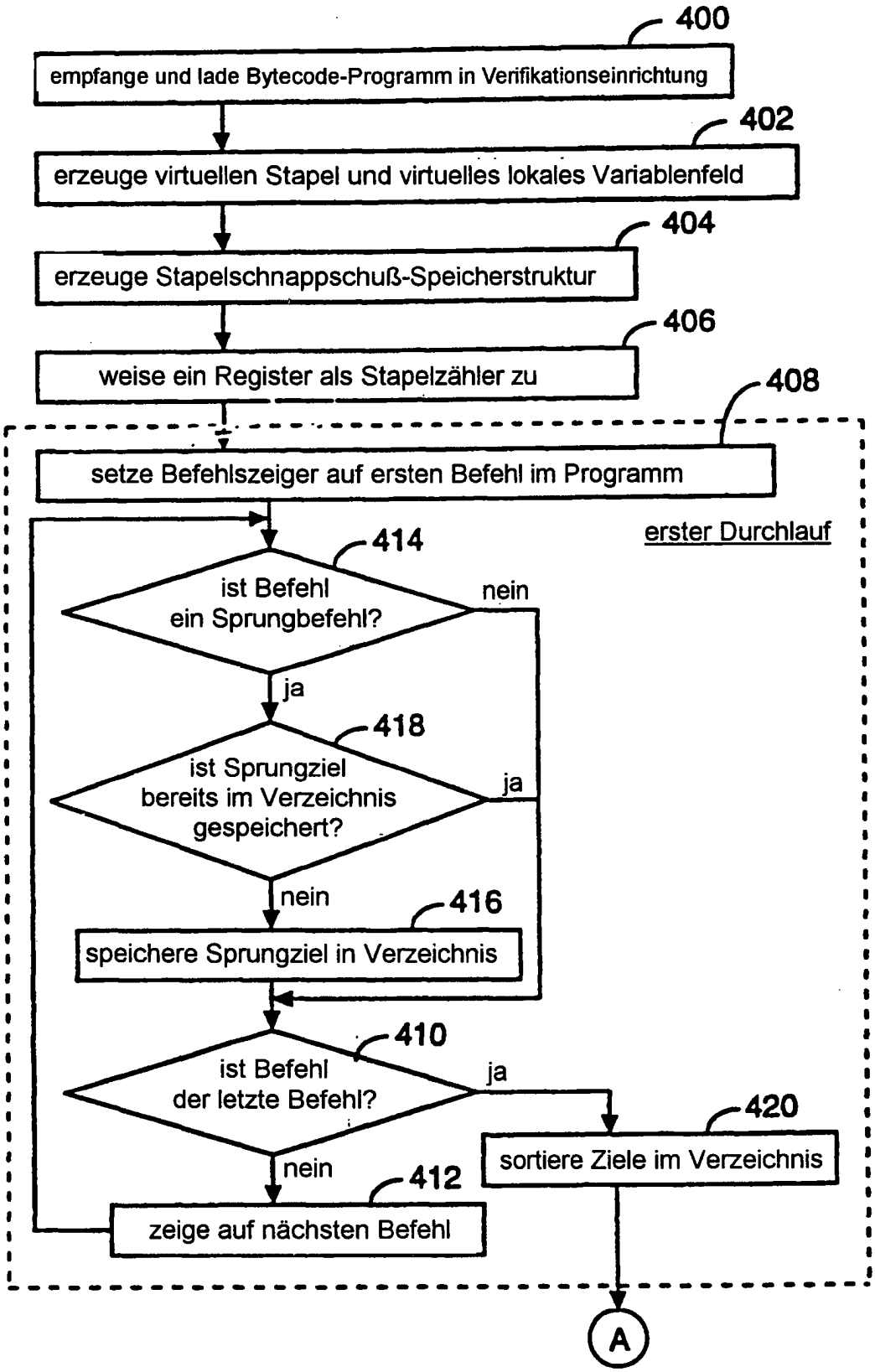

400
empfange und lade Bytecode-Programm in Verifikationseinrichtung
402
erzeuge virtuellen Stapel und virtuelles lokales Variablenfeld
404
erzeuge Stapelschnappschuß-Speicherstruktur
406
weise ein Register als Stapelzähler zu
408
setze Befehlszeiger auf ersten Befehl im Programm
erster Durchlauf
414
ist Befehl ein Sprungbefehl?
nein
ja
418
ist Sprungziel bereits im Verzeichnis gespeichert?
ja
nein
416
speichere Sprungziel in Verzeichnis
410
ist Befehl der letzte Befehl?
ja
420
sortiere Ziele im Verzeichnis
nein
412
zeige auf nächsten Befehl
A

Fig. 6A

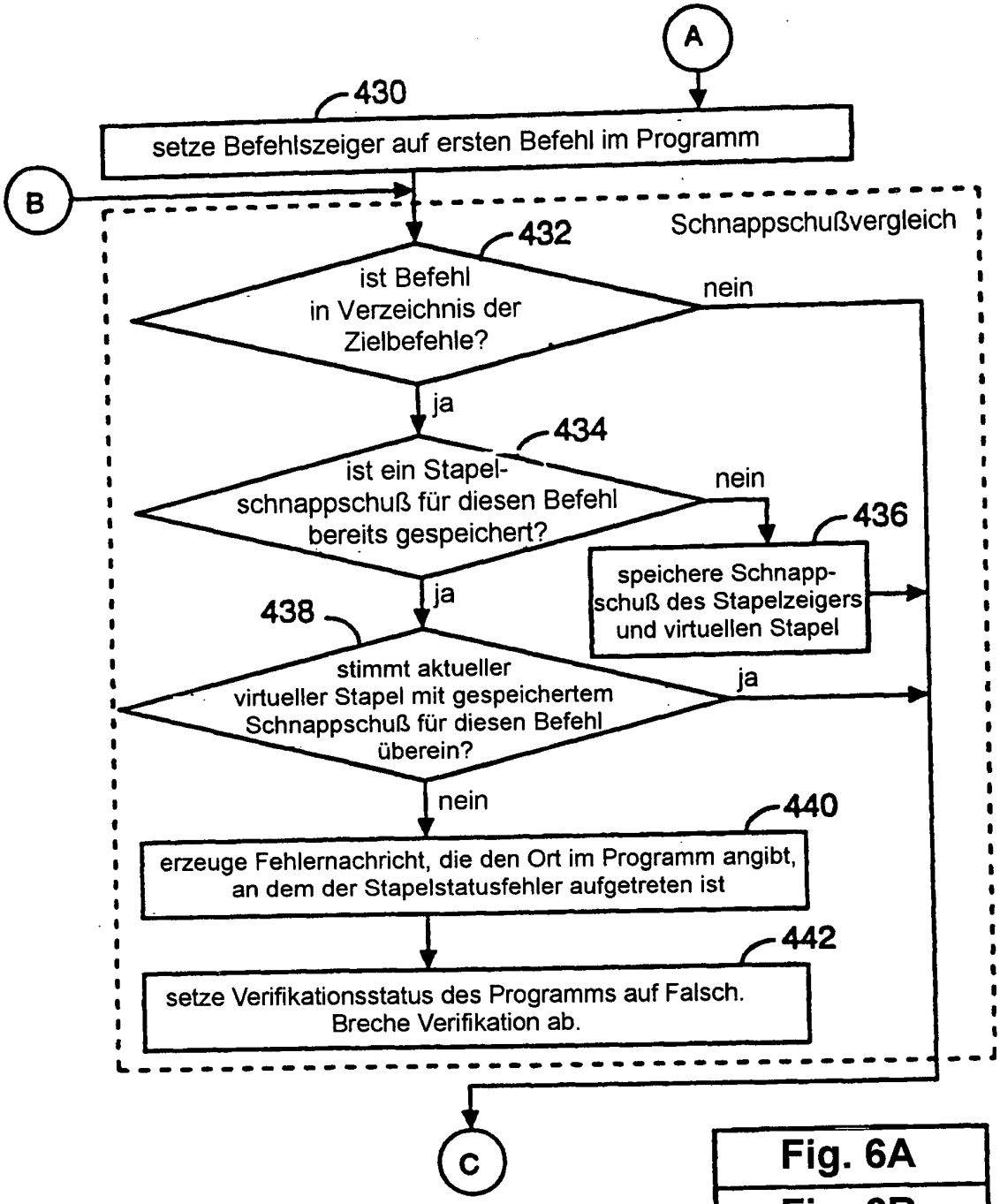


Fig. 6B

Fig. 6A
Fig. 6B
Fig. 6C
Fig. 6D
Fig. 6E
Fig. 6F
Fig. 6G

Fig. 6

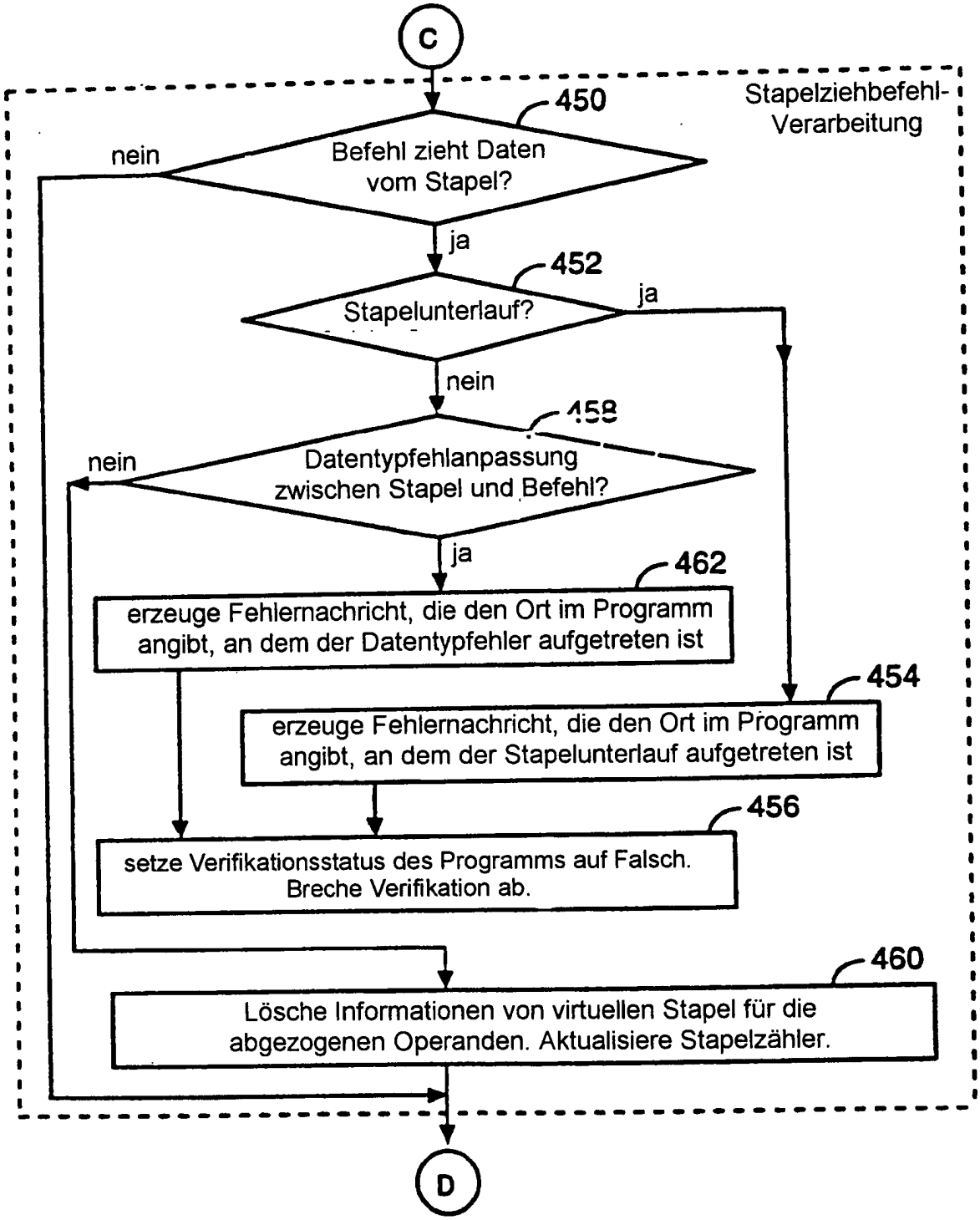

C
Stapelziehbefehl-Verarbeitung
450
Befehl zieht Daten vom Stapel?
nein
ja
452
Stapelunterlauf?
ja
nein
458
Datentypfehlanpassung zwischen Stapel und Befehl?
nein
ja
462
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Datentypfehler aufgetreten ist
454
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Stapelunterlauf aufgetreten ist
456
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
460
Lösche Informationen von virtuellen Stapel für die abgezogenen Operanden. Aktualisiere Stapelzähler.
D

Fig. 6C

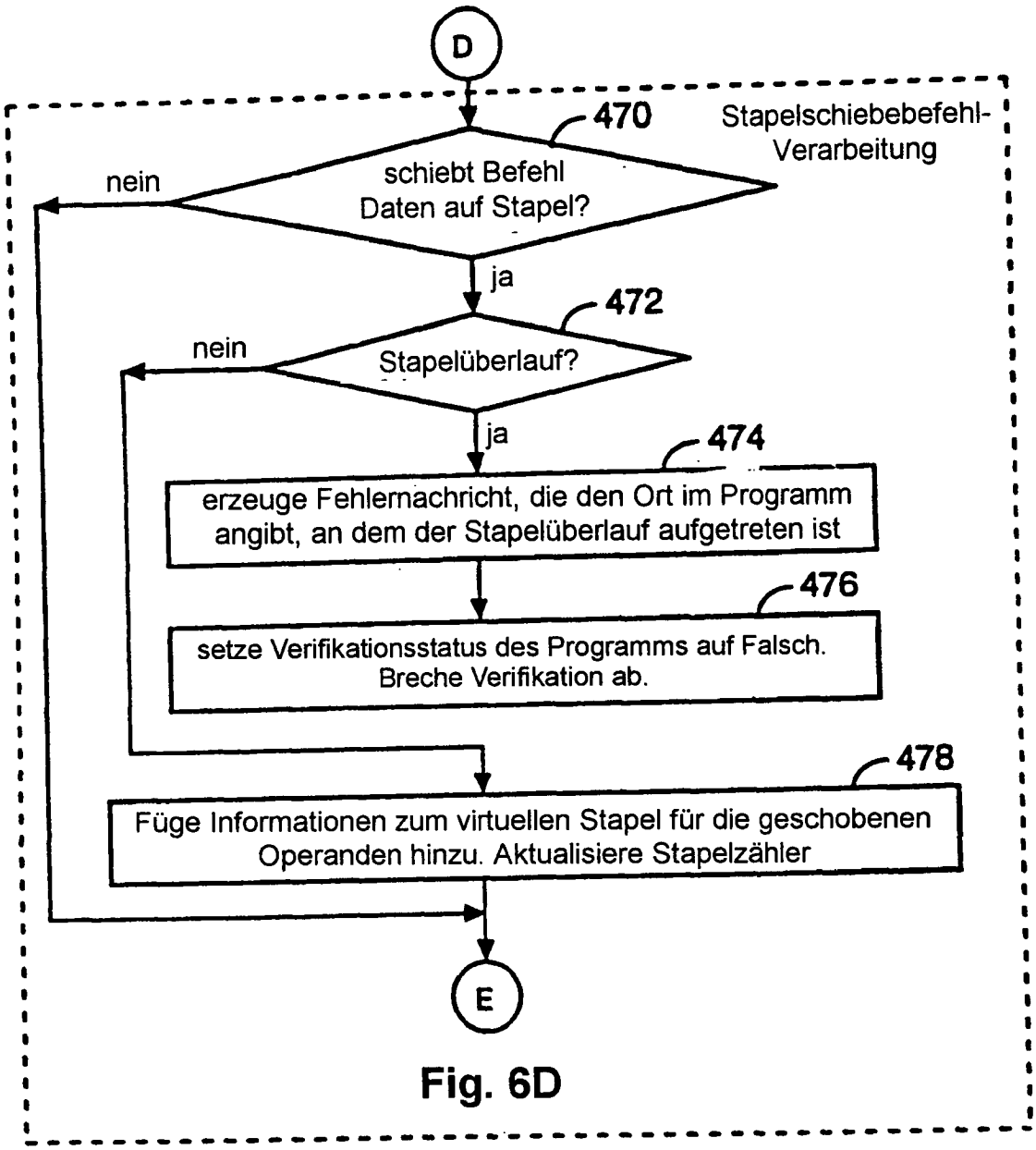

D
470
Stapelschiebebefehl-Verarbeitung
schiebt Befehl Daten auf Stapel?
nein
ja
472
Stapelüberlauf?
nein
ja
474
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Stapelüberlauf aufgetreten ist
476
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
478
Füge Informationen zum virtuellen Stapel für die geschobenen Operanden hinzu. Aktualisiere Stapelzähler
E

Fig. 6D

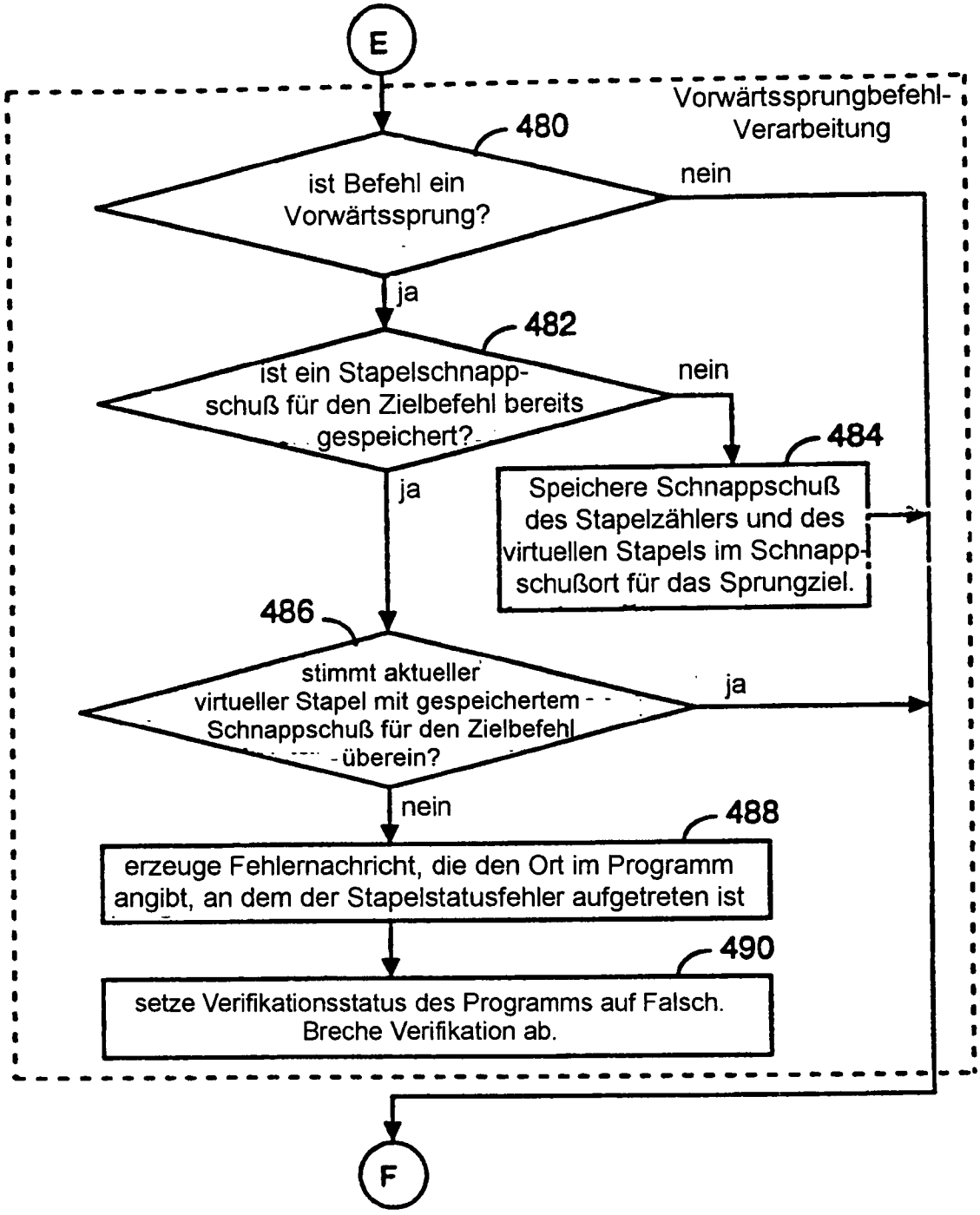

E
Vorwärtssprungbefehl-Verarbeitung
480
ist Befehl ein Vorwärtssprung?
nein
ja
482
ist ein Stapelschnapp-schuß für den Zielbefehl bereits gespeichert?
nein
ja
484
Speichere Schnappschuß des Stapelzählers und des virtuellen Stapels im Schnapp-schußort für das Sprungziel.
486
stimmt aktueller virtueller Stapel mit gespeichertem Schnappschuß für den Zielbefehl überein?
ja
nein
488
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Stapelstatusfehler aufgetreten ist
490
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
F

Fig. 6E

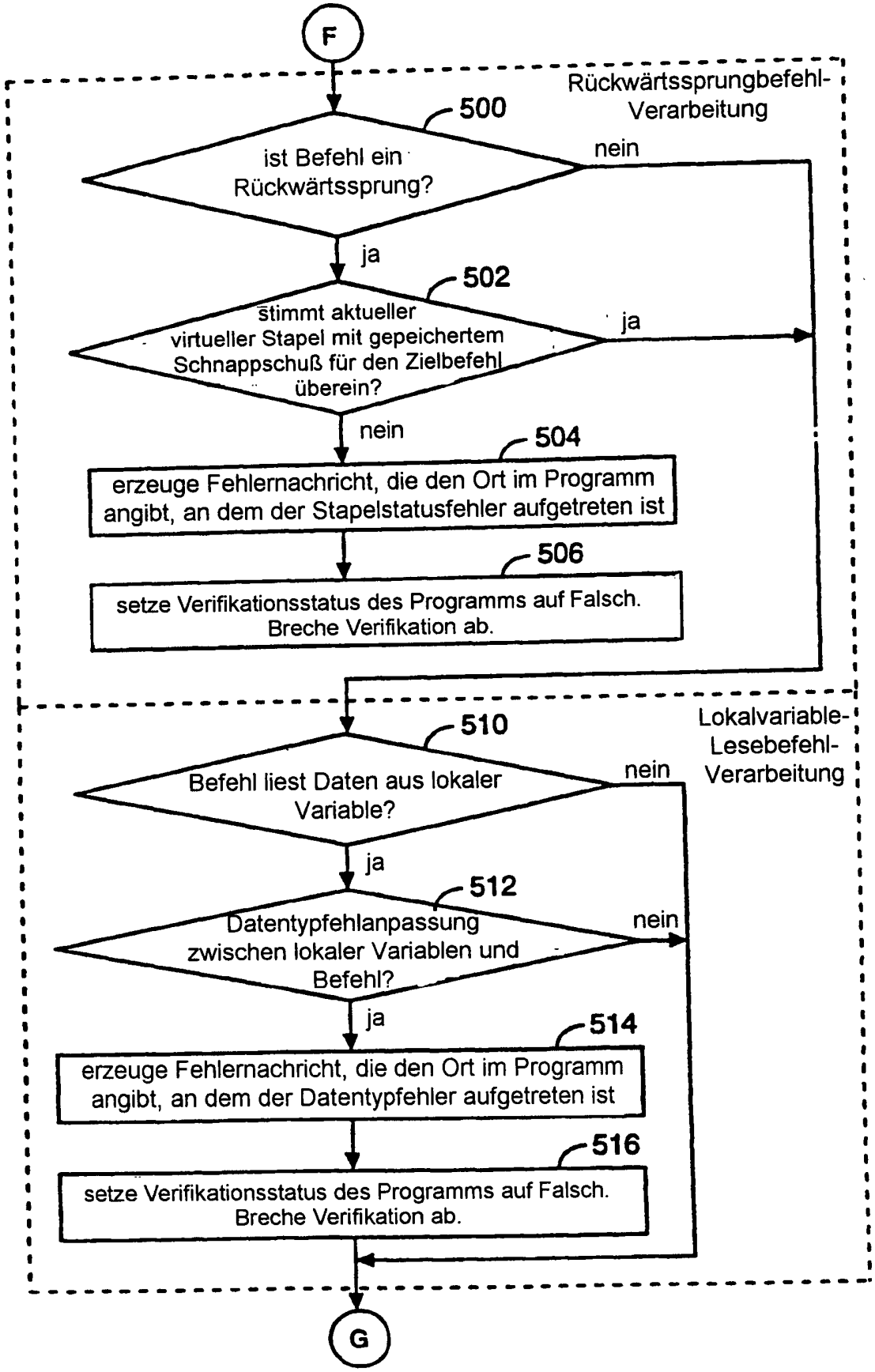

F
Rückwärtssprungbefehl-Verarbeitung
500
ist Befehl ein Rückwärtssprung?
nein
ja
502
stimmt aktueller virtueller Stapel mit gepeichertem Schnappschuß für den Zielbefehl überein?
ja
nein
504
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Stapelstatusfehler aufgetreten ist
506
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
Lokalvariable-Lesebefehl-Verarbeitung
510
Befehl liest Daten aus lokaler Variable?
nein
ja
512
Datentypfehlanpassung zwischen lokaler Variablen und Befehl?
nein
ja
514
erzeuge Fehlernachricht, die den Ort im Programm angibt, an dem der Datentypfehler aufgetreten ist
516
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
G

Fig. 6F

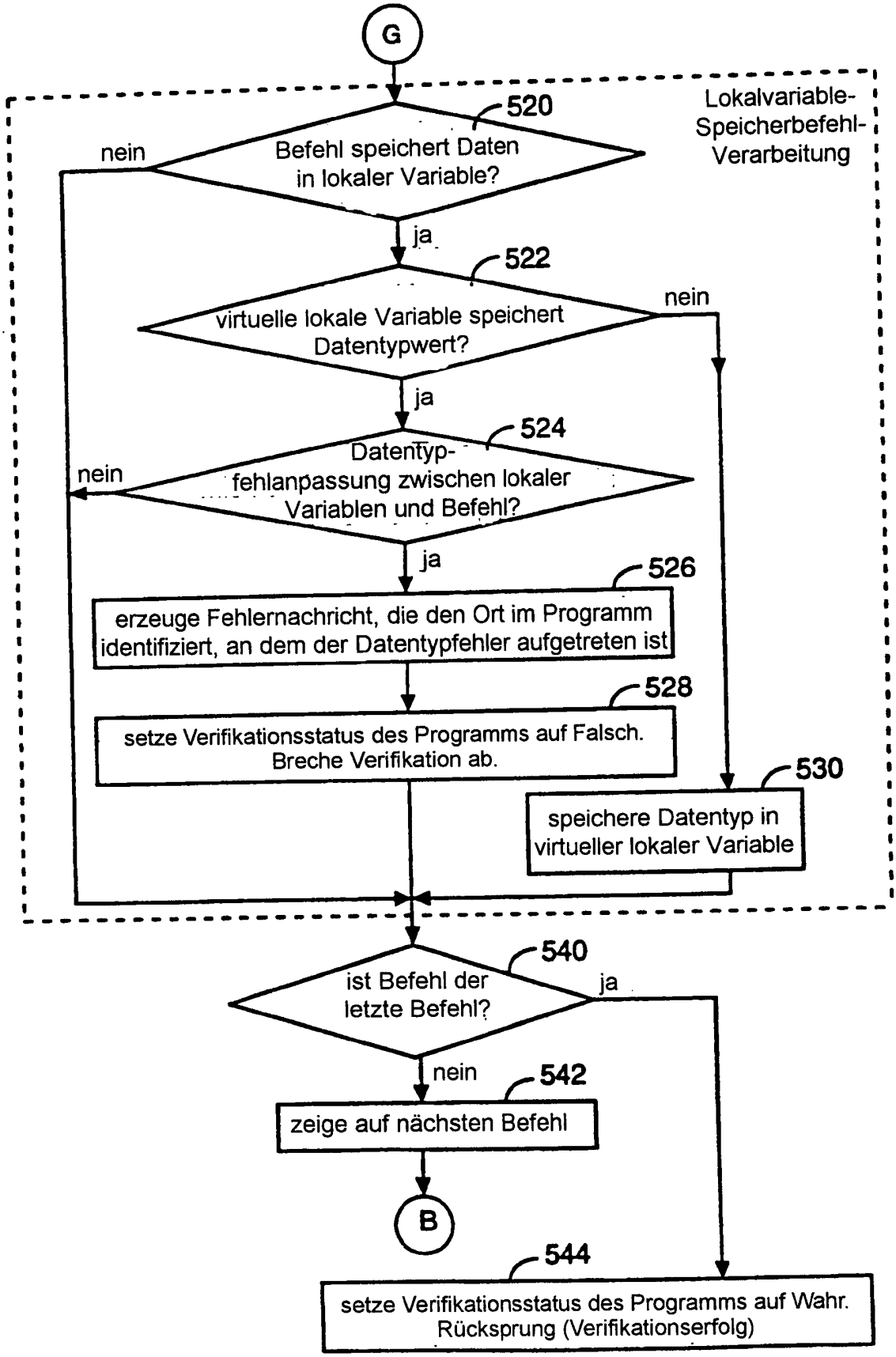

G
Lokalvariable-Speicherbefehl-Verarbeitung
520
Befehl speichert Daten in lokaler Variable?
nein
ja
522
virtuelle lokale Variable speichert Datentypwert?
nein
ja
524
Datentyp-fehlanpassung zwischen lokaler Variablen und Befehl?
nein
ja
526
erzeuge Fehlernachricht, die den Ort im Programm identifiziert, an dem der Datentypfehler aufgetreten ist
528
setze Verifikationsstatus des Programms auf Falsch. Breche Verifikation ab.
530
speichere Datentyp in virtueller lokaler Variable
540
ist Befehl der letzte Befehl?
ja
nein
542
zeige auf nächsten Befehl
B
544
setze Verifikationsstatus des Programms auf Wahr. Rücksprung (Verifikationserfolg)

Fig. 6G

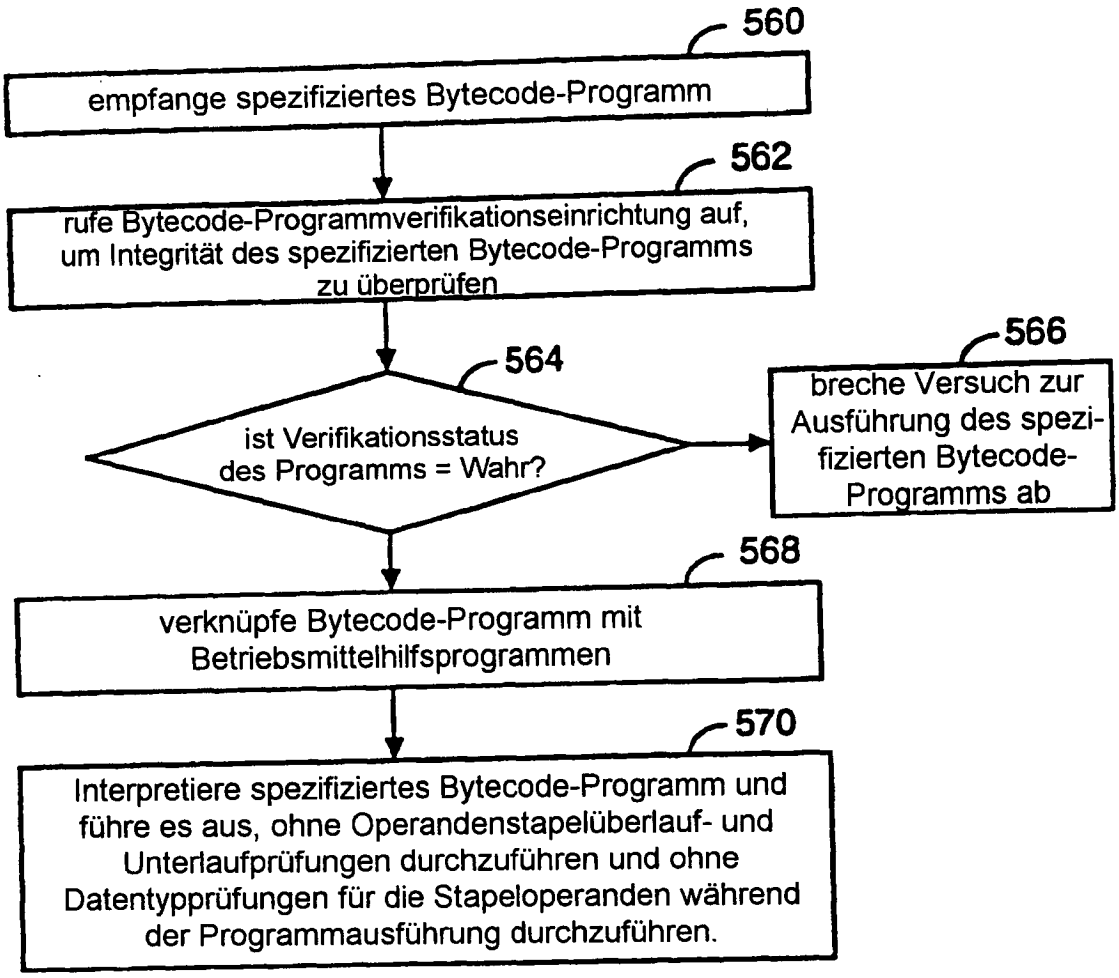

560
empfange spezifiziertes Bytecode-Programm
562
rufe Bytecode-Programmverifikationseinrichtung auf, um Integrität des spezifizierten Bytecode-Programms zu überprüfen
564
ist Verifikationsstatus des Programms = Wahr?
566
breche Versuch zur Ausführung des spezifizierten Bytecode-Programms ab
568
verknüpfe Bytecode-Programm mit Betriebsmittelhilfsprogrammen
570
Interpretiere spezifiziertes Bytecode-Programm und führe es aus, ohne Operandenstapelüberlauf- und Unterlaufprüfungen durchzuführen und ohne Datentypprüfungen für die Stapeloperanden während der Programmausführung durchzuführen.

Fig. 7