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

(10) **DE 60 2006 000 728 T2** 2009.04.23

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 708 083 B1**
(21) Deutsches Aktenzeichen: **60 2006 000 728.0**
(96) Europäisches Aktenzeichen: **06 251 778.4**
(96) Europäischer Anmeldetag: **30.03.2006**
(97) Erstveröffentlichung durch das EPA: **04.10.2006**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **19.03.2008**
(47) Veröffentlichungstag im Patentblatt: **23.04.2009**

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

(30) Unionspriorität:
96933 **31.03.2005** **US**

(84) Benannte Vertragsstaaten:
DE, GB

(73) Patentinhaber:
Sun Microsystems, Inc., Santa Clara, Calif., US

(72) Erfinder:
Bracha, Gilda, Los Altos, CA 94024, US

(74) Vertreter:
WSL Patentanwälte, 65183 Wiesbaden

(54) Bezeichnung: **Unterstützung dynamisch typisierter Sprachen in typisierten Assemblersprachen**

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich auf Computer und andere Systeme zum Ausführen von Programmbefehlen.

Hintergrund der Erfindung

[0002] Die virtuelle Java-Maschine (JVM) läßt ein Java-Programm ablaufen, indem sie "Bytecodes" ausführt. Jeder Bytecode ist ein Befehl, üblicherweise in der Größe von einem Byte, welcher der JVM sagt, daß sie eine bestimmte Operation durchführen soll. Die Sprache des Java-Bytecodes ist eine Form einer Assemblersprache.

[0003] Unter den von der JVM erkannten Bytecodes befindet sich ein bestimmter Bytecode, der "invokevirtual" heißt. Der Bytecode "invokevirtual" repräsentiert den Aufruf einer virtuellen Methode in einem entsprechenden Java-Quellcode (eine virtuelle Methode ist eine Methode, die in einer Unterklasse überschrieben werden kann). Der Bytecode "invokevirtual" wird begleitet von einem Operanden, der sowohl eine Klasse und eine Methode dieser Klasse identifiziert bzw. angibt. Der Operand identifiziert die Methode, die aufgerufen werden soll. in der Praxis ist der Operand ein 16-bit-Index in einer Tabelle, die der "constant pool" ("Konstantenvorrat") genannt wird, wobei diese Tabelle Information über die Methode enthält.

[0004] Bevor der Bytecode "invokevirtual" für eine bestimmte Methode aufgerufen wird, müssen eine oder mehrere Operanden auf den Operandenstapel geschoben werden. Beispielsweise muß ein Hinweis auf das Objekt, bezüglich dessen die betreffende Methode aufgerufen werden soll (der "receiver" bzw. "Empfänger"), auf den Operandenstapel verschoben werden. Zusätzlich müssen, falls die spezielle Methode irgendwelche Parameter erfordert, dann die aktuellen Parameter, die zu dem Aufruf der betreffenden Methode weitergeleitet werden, ebenfalls auf den Operandenstapel geschoben werden. Diese Operanden werden über den Bytecode "push" auf den Operandenstapel verschoben.

[0005] Anders als einige andere Assemblersprachen ist die Java-Bytecode-Sprache typenorientiert. Ein Mechanismus der JVM, welcher der "verifier" ("Verifizierer") genannt wird, untersucht die Bytecodes vor ihrer Ausführung und macht die Operandentypen ausfindig, die auf den Operandenstapel verschoben werden sollen. Bevor das Programm, welches den Bytecode enthält, ausgeführt wird, bestimmt der Verifizierer, ob die Operandentypen, die sich zu dem Zeitpunkt auf dem Stapel befinden werden, zu welchem der Bytecode "invokevirtual" ausgeführt wird, zu den Typen formaler Parameter paßt, welche diesen Operanden entsprechen (die Typen der formalen Parameter sind in der Erklärung der Methode angegeben, die aufgerufen werden soll). Wenn der Verifizierer irgendeine Fehlanpassung erfaßt, so kann der Verifizierer anzeigen, daß ein Fehler vorliegt.

[0006] Wenn beispielsweise die Erklärung einer Methode "C.foo()" zwei formale Parameter "bar" und "baz" definiert, und wenn "bar" als der Typ "integer" (ganze Zahl) erklärt wird, während "baz" als der Typ "List" (Liste) erklärt wird, so stellt der Verifizierer fest, ob zu dem Zeitpunkt, zu welchem der Bytecode "invokevirtual" für die Methode "C.foo()" ausgeführt wird, die Operanden auf dem Operandenstapel die Typen "Integer" bzw. "List" haben.

[0007] Die Typenüberprüfung, die durch den Verifizierer durchgeführt wird, hilft dabei, sicherzustellen, daß die Java-Laufzeitumgebung sicher und robust ist. Neben anderen Sicherungen hilft auch das Verhindern von Typ-Fehlanpassungen dabei, sicherzustellen, daß Bytecodes nicht für Zwecke verwendet werden, für welche sie nicht vorgesehen sind, so daß beispielsweise ein Java-Programm keine Operationen durchführen kann, welche Speicherbereiche beschädigen, die für dieses Programm nicht reserviert wurden. Wenn Bytecode aus dem Internet heruntergeladen wird, kann der Bytecode vor der Ausführung durch den Verifizierer geleitet werden, um sicherzustellen, daß der Bytecode keine verbotenen Operationen durchführt, wenn er ausgeführt wird. Nachdem der Bytecode einmal verifiziert worden ist, muß die Typenprüfung des Bytecodes nicht mehr jedes Mal durchgeführt werden, wenn der Bytecode ausgeführt wird. Aufgrund dieser Tatsache kann der Bytecode schneller ausgeführt werden.

[0008] Allgemein gesprochen ist Java eine statische, typenorientierte Sprache. Im Java-Quellcode muß ein Typ üblicherweise ausdrücklich für jedes Objekt erklärt werden. Nicht jede Programmiersprache ist jedoch in dieser Hinsicht statisch typenorientiert. Beispielsweise erfordern die Programmiersprache Perl und Python keine Typenerklärungen; anstatt daß sie statisch typenorientiert sind, sind diese Sprachen dynamisch typisiert bzw. typenorientiert. Trotz der Tatsache, daß die letzteren Sprachen dynamisch typisiert sind, wäre es günstig, wenn Bytecode, der auf der Basis von Programmen erzeugt wurde, welche in diesen Sprachen geschrieben wurden, auch durch die JVM ausgeführt werden könnte.

[0009] Da jedoch der Verifizierer laufend erfordert, daß die aktuellen Argumente, die sich auf dem Operandenstapel befinden, mit den erklärten Typen der formalen Parameter einer aufgerufenen Methode zusammenpassen, wenn der Bytecode "invokevirtual" ausgeführt wird, verursachen Bytecodedarstellungen von Programmen, die in dynamisch typisierten Spra-

chen geschrieben sind, immer, daß der Verifizierer einen Fehler ausgibt und die Ausführung verhindert. Es gibt typischerweise keine erklärten Typen für formale Parameter von Methoden in derartigen Programmen, so daß der Verifizierer keine Möglichkeit hat, die Typen der Argumente zu überprüfen, die sich auf dem Operandenstapel befinden.

Zusammenfassung der Erfindung

[0010] Die Erfindung wird in den anhängenden Ansprüchen definiert. Gemäß einer Ausführungsform der vorliegenden Erfindung wird eine Technik zum Unterstützen von dynamisch typenorientierten Sprachen in typenorientierten Assemblersprachen bereitgestellt. Gemäß einer Ausführungsform ergänzt ein neuer Bytecode-Befehl "invokedynamic" den existierenden Bytecode-Befehl "invokevirtual". Wie "invokevirtual" teilt auch "invokedynamic" der JVM mit, eine angegebene bzw. spezifizierte Methode aufzurufen. "Invokedynamic" veranlaßt jedoch nicht, daß der Verifizierer die gesamte strenge Typüberprüfung vor der Ausführung durchführt, zu der "invokevirtual" einen Verifizierer veranlaßt.

[0011] In einer Ausführungsform wird vor der Ausführung eines typenorientierten Assemblersprachenprogramms (beispielsweise eines Programms, das Java-Bytecodes aufweist) festgestellt, ob ein bestimmter Befehl, der eine programmatische Methode aufruft, wenn er ausgeführt wird, eine bestimmte Art von Befehl ist. Beispielsweise kann der Verifizierer feststellen, ob ein bestimmter Bytecode-Befehl innerhalb des Java-Bytecode-Programms ein Befehl "invokedynamic" ist.

[0012] Wenn der betreffende Befehl von dieser besonderen Art ist (wenn beispielsweise der betreffende Bytecode-Befehl "invokedynamic" anstatt "invokevirtual" ist), so sieht der Verifizierer davon ab, die übliche Strenge der Typüberprüfung der Argumente, die sich auf dem Operandenstapel befinden, vor der Ausführung anzuwenden, wenn der betreffende Befehl ausgeführt wird. In einer Ausführungsform werden unter solchen Umständen zumindest einige Aspekte der Typenüberprüfung bis zur Laufzeit ausgesetzt.

[0013] Wenn alternativ der betreffende Befehl nicht von der speziellen Art ist (wenn beispielsweise der betreffende Bytecode-Befehl "invokevirtual" anstatt "invokedynamic" ist), so führt der Verifizierer die übliche strenge Typüberprüfung der Argumente, die sich auf dem Operandenstapel befinden, wenn der betreffende Befehl ausgeführt wird, vor der Ausführung durch. Wenn die Typen der Argumente nicht zu den Typen der entsprechenden formalen Parameter passen, die für die programmatische Methode erklärt wurden, so verhindert der Verifizierer, daß das Assemblerspracheprogramm (in Assemblersprache geschriebenes Programm) ausgeführt wird.

[0014] Im Ergebnis kann die spezielle Art von Befehl (z. B. "invokedynamic") verwendet werden, um im Bytecode den Aufruf einer programmatischen Methode darzustellen, die nicht notwendigerweise die Typen der formalen Parameter der Methoden spezifiziert. Da der Verifizierer in Reaktion auf einen solchen Befehl eine weniger strenge Typüberprüfung vornimmt, kann die JVM Assemblerspracheprogramme, die auf der Basis von Sourcecode erzeugt wurden, welcher in einer dynamisch typenorientierten Sprache geschrieben wurde, ausführen.

Kurzbeschreibung der Figuren

[0015] Verschiedene Ausführungsformen der Erfindung werden nun im einzelnen lediglich beispielhaft unter Bezug auf die folgenden Zeichnungen beschrieben:

[0016] Fig. 1 ist ein Flußdiagramm, welches eine Übersicht des Betriebs bzw. der Arbeitsweise der Ausführungsform der vorliegenden Erfindung veranschaulicht.

[0017] Fig. 2 ist ein Flußdiagramm, das eine beispielhafte Technik zum Unterstützen einer dynamisch typenorientierten Sprache in einer typenorientierten Assemblersprache veranschaulicht, welche eine erleichterte Typenüberprüfung vor der Laufzeit gemäß einer Ausführungsform der vorliegenden Erfindung verwendet.

[0018] Fig. 3 ist ein Hardwareblockdiagramm einer beispielhaften Computereinheit, auf welcher gewisse Ausführungsformen der Erfindung implementiert sein können.

Genaue Beschreibung

Übersicht

[0019] Gemäß einer Ausführungsform der vorliegenden Erfindung wird für dynamisch typenorientierte bzw. typisierte Sprachen, einschließlich objektorientierter Sprachen, eine Unterstützung in typenorientierten Assemblersprachen bereitgestellt. Ein Arbeitsflußdiagramm, welches eine Übersicht von höherer Ebene über die Arbeitsweise einer Ausführungsform der vorliegenden Erfindung veranschaulicht, ist in Fig. 1 wiedergegeben.

[0020] Gemäß Fig. 1 wird vor der Ausführung eines Assemblerspracheprogramms, welches einen bestimmten Befehl enthält, der eine programmatische Methode aufruft, wenn er ausgeführt wird, festgestellt, ob der betreffende Befehl "invokedynamic" ist. Beispielsweise kann der Verifizierer einer JVM eine solche Feststellung vor der Ausführung eines Bytecode-Programms treffen. Auch wenn das Befehlsetikett "invokedynamic" hier nur für Zwecke der Veran-

schaulichung verwendet wird, könnten alternative Ausführungsformen auch andere Etiketten für einen solchen Befehl verwenden. Wenn der betreffende, eine Methode aufrufende Befehl "invokedynamic" ist, so geht die Steuerung weiter zu Block **104**. Ansonsten wäre der betreffende Befehl irgendein anderer Befehl, der eine programmatische Methode aufruft, wenn er ausgeführt wird (z. B. "invokevirtual"), und die Steuerung geht dann weiter an Block **106**.

[0021] In Block **104** wird, wenn der betreffende Befehl "invokedynamic" ist, die übliche Überprüfung der Typen der Argumente, die sich auf dem Stapel befinden, wenn der betreffende Befehl ausgeführt wird, wobei die Überprüfung üblicherweise in Reaktion auf einen "invokevirtual"-Befehl durchgeführt werden würde, vor der Laufzeit nicht vollständig durchgeführt. Statt dessen kann vor der Laufzeit eine erleichterte Form der Überprüfung durchgeführt werden. Im Gegensatz zu der üblichen Typenüberprüfung vor der Ausführung werden in der erleichterten Überprüfung die Typen der Argumente, die sich auf dem Stapel befinden werden oder das Fehlen derartiger Typen für sich gesehen nicht bewirken, daß der Verifizierer verhindert, daß das Assemblerspracheprogramm abläuft bzw. ausgeführt wird.

[0022] Es kann jedoch eine andere Überprüfung bezüglich der Argumente durchgeführt werden, die sich auf dem Stapel befinden werden und das Ergebnis dieser anderen Überprüfung kann bewirken, daß der Verifizierer das Programm an der Ausführung hindert. Einige Beispiele dieser weiteren Überprüfung werden unten noch genauer beschrieben. Weiterhin können die Typen der Argumente, die sich zu dem Zeitpunkt, zu welchem der Befehl für die Ausführung bereit ist, auf dem Operandenstapel befinden, später zur Laufzeit überprüft werden, um zu verhindern, daß sich ein Programm fehlerhaft verhält.

[0023] Alternativ wird in Block **106**, wenn der betreffende Befehl ein eine Methode aufrufender Befehl ist, der nicht "invokedynamic" ist, wie z. B. "invokevirtual", festgestellt, ob die Typen von Argumenten, die sich zu dem Zeitpunkt auf dem Stapel befinden, zu welchem der betreffende Befehl ausgeführt wird, zu den entsprechenden Typen formaler Parameter der aufzurufenden Methode passen. Wenn beispielsweise der betreffende Befehl, der hier eher symbolisch anstatt wörtlich wiedergegeben ist, "invokevirtual C.foo()" ist, und wenn die Erklärung der Methode "C.foo()" anzeigt, daß "C.foo()" zwei formale Parameter der Typen "Integer" bzw. "List" hat, so kann der Verifizierer feststellen, ob die Typen der beiden Argumente, die sich zu dem Zeitpunkt, zu welchem der "invokevirtual"-Befehl ausgeführt wird, auf dem Operandenstapel befinden, tatsächlich "Integer" und "List" sind. Wenn die Typen der Argumente zu den Typen der formalen Parameter passen, so geht die Steuerung weiter an Block **108**. Ansonsten geht die Steuerung weiter zu Block **110**.

[0024] In Block **108** kann eine weitere Überprüfung vor der Ausführung des Assemblerspracheprogramms durchgeführt werden. Beispielsweise kann der Verifizierer eine zusätzliche Überprüfung bezüglich des speziellen Befehls anderer Befehle in dem Assemblerspracheprogramm durchführen. Wenn all diese weiteren Überprüfungen zu einem zufriedenstellenden Ergebnis führen, so kann die JVM das Assemblerspracheprogramm ausführen.

[0025] Alternativ wird in Block **110** die Ausführung des Assemblerspracheprogramms verhindert. Beispielsweise könnte der Verifizierer eine Fehlermeldung ausgeben, die eine Fehlanpassung der Typen anzeigt, und verhindern, daß die JVM irgendeinen der Bytecodes in dem Assemblerspracheprogramm ausführt.

[0026] In gewissen Ausführungsformen der Erfindung kann eine Computervorrichtung, ein Rechnersystem und/oder ein computerlesbares Medium so ausgestaltet sein, daß es die vorstehende Technik gemäß Fig. 1 ausführt.

Erleichterte Typenüberprüfung vor der Laufzeit

[0027] Fig. 2 ist ein Flußdiagramm, das eine beispielhafte Technik zum Unterstützen einer dynamisch typisierten bzw. typenorientierten Sprache in einer typenorientierten Assemblersprache veranschaulicht, welches eine erleichterte Typenüberprüfung vor der Laufzeit gemäß einer Ausführungsform der vorliegenden Erfindung verwendet. Beispielsweise kann der Verifiziermechanismus der JVM eine solche Technik ausführen. Auch wenn die Schritte der Technik für die Zwecke der Veranschaulichung in einer bestimmten Reihenfolge dargestellt sind, können die Schritte der Technik auch in anderen Reihenfolgen durchgeführt wenden, als der speziell in diesem Beispiel dargestellten Reihenfolge.

[0028] Gemäß Fig. 2 wird in Block **202** vor der Ausführung eines Assemblerspracheprogramms, das einen bestimmten Befehl enthält, der bei seiner Ausführung eine programmatische Methode aufhob, festgestellt, ob der betreffende Befehl "invokedynamic" ist. Beispielsweise kann der Verifizierer eine solche Feststellung vor der Ausführung eines Java-Bytecode-Programms treffen. Auch wenn hier für Zwecke der Veranschaulichung das Befehlsetikett "invokedynamic" verwendet wird, so können doch alternative Ausführungsformen auch andere Etiketten für einen solchen Befehl verwenden. Wenn der betreffende, die Methode aufrufende Befehl "invokedynamic" ist, so geht die Steuerung weiter an Block **204**. Ansonsten ist der betreffende Befehl irgendein anderer Befehl, der bei seiner Ausführung eine programmatische Methode aufruft (z. B. "invokevirtual"), und die

Steuerung geht weiter zu Block **220**.

[0029] In Block **204** wird, wenn der betreffende Befehl "invokedynamic" ist, vor der Ausführung des Assemblerspracheprogramms festgestellt, ob eine Anzahl von Argumenten, die sich zu dem Zeitpunkt, zu welchem der betreffende Befehl ausgeführt wird, auf dem Operandenstapel befinden, zu einer Anzahl formaler Parameter paßt, die zu der programmatischen Methode gehören. Wenn die Methode beispielsweise "C.foo()" ist, und die Erklärung von "C.foo()" zwei formale Parameter "bar" und "baz" definiert, so bestimmt der Verifizierer, ob zu dem Zeitpunkt, zu welchem der Bytecode "invokedynamic" für die Methode "C.foo()" ausgeführt wird, es zumindest zwei Argumente auf dem Operandenstapel neben dem Aufruf des Objektes gibt, welcher die Methode selbst enthält. Wenn die Anzahl von Argumenten zu der Anzahl der formalen Parameter paßt, so geht die Steuerung weiter an Block **206**. Ansonsten geht die Steuerung weiter zu Block **224**.

[0030] Gemäß einer Ausführungsform wird in Block **206** vor der Ausführung des Assemblerspracheprogramms festgestellt, ob irgendwelche der sich auf Methoden beziehenden Argumente, die sich zu dem Zeitpunkt auf dem Operandenstapel befinden werden, zu welchem der betreffende Befehl ausgeführt wird, einen primitiven Typus (z. B. "Integer" ("ganze Zahl")) haben. Beispielsweise kann der Verifizierer vor der Ausführung des Assemblerspracheprogramms feststellen, ob irgendeines der Argumente, die sich auf die Methode "C.foo()" beziehen und die sich auf dem Operandenstapel befinden wenden, wenn der "invokedynamic", welcher ausgeführt wird, von einem primitiven Typ und damit kein Objekt ist. Wenn irgend eines der Argumente von einem primitiven Typ ist, so geht die Steuerung weiter zu Block **224**. Ansonsten geht die Steuerung weiter zu Block **208**.

[0031] In der oben beschriebenen Ausführungsform geht die Steuerung von Block **206** weiter zu Block **224**, wenn irgendwelche der Argumente von einem primitiven Typ sind. In einer alternativen Ausführungsform jedoch wird die Ausführung des Programms nicht wie in Block **224** verhindert, selbst wenn eines oder mehrere der Argumente von einem primitiven Typ ist. Statt dessen kann in einer solchen Ausführungsform die übliche strenge Typenüberprüfung bezüglich der Argumente vom primitiven Typ vor der Ausführung durchgeführt werden, während eine mehr erleichterte Form der Überprüfung bezüglich der Argumente vom nicht-primitiven Typ vor der Ausführung durchgeführt wird; eine strengere Typüberprüfung kann bezüglich dieser Argumente vom nicht-primitiven Typ zur Laufzeit vorgenommen werden, und zwar in der Art und Weise, die unter Hinweis auf die Blöcke **216** und **218** unten beschrieben wird.

[0032] In Block **208** wird vor der Ausführung des Assemblerspracheprogramms festgestellt, ob ein Objekt, bezüglich dessen die programmatische Methode aufgerufen werden soll (der Empfänger), sich zu dem Zeitpunkt auf dem Operandenstapel befindet, zu welchem der betreffende Befehl ausgeführt wird. Wenn beispielsweise die aufzurufende Methode "C.foo()" ist, so kann der Verifizierer vor der Ausführung des Assemblerspracheprogramms feststellen, ob das Objekt "C" sich tatsächlich zu dem Zeitpunkt auf dem Operandenstapel befindet, zu welchem der Bytecode "invokedynamic" ausgeführt wird. In einer Ausführungsform kommt es nicht darauf an, ob der Empfänger von irgendeinem bestimmten Typ ist, es kommt lediglich darauf an, ob der Empfänger sich auf dem Stapel befindet. Wenn das Objekt auf dem Operandenstapel ist, so geht die Steuerung weiter zu Block **210**. Ansonsten geht die Steuerung zu Block **224**.

[0033] In Block **210** beginnt die Ausführung des Assemblerspracheprogramms. Beispielsweise kann die JVM damit beginnen, Java-Bytecode-Befehle auszuführen, welche der Verifizierer entsprechend der obigen Erläuterung überprüft hat. Wie nachstehend beschrieben wird, kann eine Typenüberprüfung, die bezüglich des Befehls "invokedynamic" nicht vor der Laufzeit durchgeführt wurde, nunmehr zur Laufzeit durchgeführt werden, um sicherzustellen, daß sich das Programm nicht fehlerhaft verhält.

[0034] In Block **212** wird während der Ausführung des Assemblerspracheprogramms festgestellt, ob der nächste Befehl "invokedynamic" ist. Beispielsweise kann die JVM eine solche Feststellung treffen, während sie das Java-Bytecode-Programm ausführt. Wenn der nächste aufzurufende Befehl "invokedynamic" ist, so geht die Steuerung weiter zu Block **216**. Ansonsten geht die Steuerung weiter zu Block **214**.

[0035] In Block **214** wird der nächste Befehl ausgeführt. Vorausgesetzt, daß das Assemblerspracheprogramm die Ausführung noch nicht abgeschlossen hat, geht die Steuerung zurück zu Block **212**.

[0036] Alternativ wird in Block **216**, wenn der nächste Befehl "invokedynamic" ist, für jedes methodenbezogene Argument auf dem Operandenstapel, welches einem formalen Parameter entspricht, der nicht von dem Typ "Objekt" ist, ein Versuch vorgenommen, das Argument so auszubilden, daß es von dem Typ des formalen Parameters ist. Wenn beispielsweise der nächste Befehl, der nur symbolisch anstatt wörtlich wiedergegeben ist, "invokevirtual C.foo()" ist, und wenn die Erklärung der Methode "C.foo()" anzeigt, daß "C.foo()" zwei formale Parameter "bar" und "baz" der Typen "Integer" bzw. "Objekt" hat, so kann die JVM versuchen, das Argument, welches "bar" entspricht, zu dem Typ "Integer" zu machen, das Argument, welches "baz" entspricht, jedoch so zu lassen

wie es ist.

[0037] In einer Ausführungsform wird diese Grundform implementiert durch Bereitstellen doppelter Eintrittspunkte für jede Methode – einer, welcher mit einem Vorlauf beginnt, der Umformungen durchführt, und einem weiteren, der die üblichen Operationen der Methode ausführt.

[0038] In Block **218** wird festgestellt, ob der Umformversuch, der in Block **216** durchgeführt wurde, fehlgeschlagen ist. Der Umformversuch schlägt fehl, wenn die Typen der Argumente, die sich auf dem Stapel befinden, nicht zu den entsprechenden Typen der formalen Parameter der Methode passen, die aufgerufen werden soll. Wenn beispielsweise der nächste Befehl, der nur symbolisch und nicht wörtlich wiedergegeben ist, "invokevirtual C.foo()" ist, und wenn die Erklärung der Methode "C.foo()" anzeigt, daß "C.foo()" zwei formale Parameter der Typen "Integer" bzw. "List" hat, so schlägt der Umformversuch fehl, wenn nicht die Typen der beiden Argumente, die sich auf dem Operandenstapel befinden, tatsächlich "Integer" und "List" sind. Wenn der Umformversuch nicht fehlgeschlagen ist, so geht die Steuerung weiter zu Block **214**, ansonsten geht die Steuerung weiter zu Block **226**.

[0039] Alternativ wird in Block **220**, wenn der betreffende Befehl ein eine Methode aufrufender Befehl ist, der nicht "invokedynamic" ist, wie z. B. "invokevirtual", vor der Ausführung des Assemblerspracheprogramms festgestellt, ob die Typen der Argumente, die sich zu dem Zeitpunkt auf dem Stapel befinden, zu welchem der betreffende Befehl ausgeführt wird, mit den entsprechenden Typen der formalen Parameter der aufzurufenden Methode zusammenpassen. Wenn beispielsweise der betreffende Befehl, der nur symbolisch anstatt wörtlich wiedergegeben ist, "invokevirtual C.foo()" ist, und wenn die Erklärung der Methode "C.foo()" anzeigt, daß "C.foo()" zwei formale Parameter der Typen "Integer" bzw. "List" hat, so kann der Verifizierer feststellen, ob die Typen der beiden Argumente, die sich zu dem Zeitpunkt auf dem Operandenstapel befinden, zu welchem der Befehl "invokevirtual" ausgeführt wird, tatsächlich "Integer" und "List" sind. Wenn die Typen der Argumente zu den Typen der formalen Parameter passen, so geht die Steuerung weiter zu Block **222**. Ansonsten geht die Steuerung zu Block **224**.

[0040] In Block **222** kann eine weitere Überprüfung vor der Ausführung des Assemblerspracheprogramms durchgeführt werden. Beispielsweise kann der Verifizierer eine zusätzliche Überprüfung bezüglich des betreffenden Befehls und anderer Befehle in dem Assemblerspracheprogramm vornehmen. Wenn all diese weiteren Überprüfungen zu einem zufriedenstellenden Ergebnis führen, so kann die JVM beginnen, in Block **210** das Assemblerspracheprogramm auszuführen.

[0041] Alternativ wird in Block **224** die Ausführung des Assemblerspracheprogramms verhindert. Beispielsweise kann der Verifizierer eine Fehlermeldung ausgeben und verhindern, daß die JVM irgendwelche der Bytecodes in dem Assemblerspracheprogramm ausführt.

[0042] Alternativ wird in Block **226** eine Ausnahme eingeführt. Beispielsweise kann die JVM eine Ausnahme einführen. In Reaktion auf das Einführen einer solchen Ausnahme kann das Ausnahmehandhabungsprogramm die Ausführung des Assemblerspracheprogramms stoppen bzw. unterbrechen. Beispielsweise kann in Reaktion auf das Einführen einer solchen Ausnahme die JVM eine Fehlermeldung ausgeben und die Ausführung jeglicher weiterer Bytecodes in dem Assemblerspracheprogramm beenden.

Übersicht über die Hardware

[0043] Fig. 3 ist ein Blockdiagramm, welches ein Computersystem **300** veranschaulicht, auf welchem eine Ausführungsform der Erfindung implementiert werden kann. Das Computersystem **300** weist einen Bus **302** zum Ermöglichen des Informationsaustausches und einen oder mehrere Prozessoren **304** für die Verarbeitung von Informationen auf, die mit dem Bus **302** verbunden sind. Das Computersystem **300** umfaßt auch einen Hauptspeicher **306**, wie z. B. einen Speicher mit wahlfreiem Zugriff (RAM) oder irgendeine andere dynamische Speichereinrichtung für das Speichern von Informationen und Befehlen, die in dem Prozessor **304** ausgeführt werden sollen, welcher mit dem Bus **302** verbunden ist. Der Hauptspeicher **306** kann ebenfalls verwendet werden, um temporäre Variable oder andere zwischenzeitliche Informationen während der Ausführung von Befehlen durch den Prozessor **304** zu speichern. Das Computersystem **300** kann weiterhin einen nur lesbaren Speicher (ROM) **308** oder eine andere statische Speichereinrichtung aufweisen, die mit dem Bus verbunden ist, und die für das Speichern von Zustandsinformationen und Befehlen für den Prozessor **304** dient. Eine Speichereinrichtung **310**, wie z. B. eine Magnetplatte oder eine optische Platte, ist für das Speichern von Informationen und Befehlen vorgesehen und mit dem Bus **302** verbunden.

[0044] Das Computersystem **300** kann über den Bus **302** mit einer Anzeige bzw. einem Display **312** verbunden sein, wie z. B. mit einer Kathodenstrahlröhre (CRT), um für einen Benutzer Information anzuzeigen. Eine Eingabeeinrichtung **314**, welche alphanumerische Tasten und sonstige Tasten aufweist, ist mit dem Bus **302** verbunden, um Information und Befehlsauswahlen an dem Prozessor **304** zu kommunizieren. Eine andere Art von Benutzereingabe-

einrichtung ist die Cursorsteuerung **316**, wie z. B. eine Mouse, ein Trackball oder Cursorleittasten für das Kommunizieren von Richtungs- oder Leitungsinformation und Befehlsauswahlen für den Prozessor **304** und zum Steuern der Cursorbewegung auf der Anzeige **312**. Diese Eingabeeinrichtung hat typischerweise zwei Freiheitsgrade in zwei Achsen, nämlich einer ersten Achse (z. B. x) und einer zweiten Achse (z. B. y), die es dem Gerät erlaubt, Positionen in einer Ebene anzugeben.

[0045] In dem Computersystem **300** kann der Bus **302** irgendeinen Mechanismus und/oder Medium sein, welches ermöglicht, daß Information, Signale, Daten etc. zwischen den verschiedenen Komponenten ausgetauscht wenden. Beispielsweise kann der Bus **302** ein Satz von elektrischen Leitern sein, der elektrische Signale trägt. Der Bus **302** kann auch ein drahtloses Medium (z. B. Luft) sein, die drahtlose Signale zwischen einem oder mehreren der Komponenten trägt. Der Bus **302** kann auch ein Medium (z. B. Luft) sein, das es ermöglicht, daß Signale kapazitiv zwischen einem oder mehreren der Komponenten ausgetauscht werden. Der Bus **302** kann weiterhin ein Netzwerkanschluß sein, der eine oder mehrere der Komponenten verbindet bzw. anschließt. Insgesamt gesehen kann irgendein beliebiger Mechanismus und/oder Medium, die es ermöglichen, daß Information, Signale, Daten etc. zwischen den verschiedenen Komponenten ausgetauscht werden, als Bus **302** verwendet werden.

[0046] Der Bus **302** kann auch eine Kombination dieser Mechanismen/Medien sein. Beispielsweise kann der Prozessor **304** drahtlos mit der Speichereinrichtung **310** kommunizieren. In einem solchen Fall wäre der Bus **302** vom Standpunkt des Prozessors **304** und der Speichereinrichtung **310** aus ein drahtloses Medium, wie z. B. Luft. Weiterhin kann der Prozessor **304** mit dem ROM **308** kapazitiv kommunizieren. In diesem Fall wäre der Bus **302** das Medium (wie z. B. Luft), welche es ermöglicht, daß diese kapazitive Kommunikation stattfindet. Weiterhin kann der Prozessor **304** über eine Netzwerkverbindung mit dem Hauptspeicher **306** kommunizieren. In diesem Fall wäre der Bus **302** die Netzwerkverbindung bzw. der Netzwerkanschluß. Weiterhin kann der Prozessor **304** über einen Satz von elektrischen Leitern mit der Anzeige **312** kommunizieren. In diesem Fall wäre der Bus **302** der Satz von elektrischen Leitern. Je nachdem wie die verschiedenen Komponenten miteinander kommunizieren, kann also der Bus **302** unterschiedliche Formen annehmen. Der Bus **302**, wie er in Fig. 3 dargestellt ist, repräsentiert funktionell alle diese Mechanismen und/oder Medien, die es ermöglichen, daß Information, Signale, Daten, etc. zwischen den verschiedenen Komponenten ausgetauscht werden.

[0047] Gemäß einer Ausführungsform der Erfindung können die hier beschriebenen Methoden bzw. Verfahren durch ein Computersystem **300** in Reaktion darauf ausgeführt werden, daß der Prozessor **304** eine oder mehrere Sequenzen aus einem oder mehreren Befehlen ausführt, die in dem Hauptspeicher **306** enthalten sind. Derartige Befehle können von einem anderen maschinenlesbaren Medium, wie z. B. einer Speichereinrichtung **310**, in den Hauptspeicher **306** gelesen werden. Die Ausführung der Sequenzen von Befehlen, die in dem Hauptspeicher **306** enthalten sind, veranlaßt den Prozessor **304**, die darin beschriebenen Prozeßschritte durchzuführen. In alternativen Ausführungsformen kann eine hartverdrahtete Schaltung anstelle von oder in Kombination mit Softwarebefehlen verwendet werden, um die Methoden zu implementieren, wobei derartige Implementierungen nicht auf irgendeine spezielle Kombination von Hardwareschaltung und Software beschränkt sind.

[0048] Der Begriff "maschinenlesbares Medium", wie er hier verwendet wird, bezieht sich auf irgendein Medium, das an der Bereitstellung von Daten beteiligt ist, die bewirken, daß eine Maschine in einer bestimmten Art und Weise arbeitet. In einer Ausführungsform, die unter Verwendung des Computersystems **300** implementiert ist, sind verschiedene maschinenlesbare Medien involviert, beispielsweise beim Bereitstellen von Befehlen für den Prozessor **304** für die Ausführung. Ein solches Medium kann viele Formen annehmen, was nichtflüchtige Medien, flüchtige Medien und Übertragungsmedien einschließt, ohne jedoch hierauf beschränkt zu sein. Nichtflüchtige Medien umfassen beispielsweise optische oder magnetische Platten, wie z. B. die Speichereinrichtung **310**. Flüchtige Medien umfassen dynamischen Speicher, wie z. B: den Hauptspeicher **306**. Übertragungsmedien umfassen Koaxialkabel, Kupferdraht und Faseroptiken, einschließlich der Kabel, welche den Bus **302** aufweisen. Übertragungsmedien können auch die Form von akustischen Wellen oder Lichtwellen annehmen, wie z. B. diejenigen, die während Radiowellen- und Infrarot-Datenkommunikationen erzeugt werden.

[0049] Übliche Formen maschinenlesbarer Medien umfassen beispielsweise eine Floppydisk, eine flexible Diskette, eine Festplatte, Magnetbänder oder irgendein anderes magnetisches Medium, eine CD-ROM, irgendein anderes optisches Medium, Lochkarten, Papierband oder irgendein anderes physikalisches Medium mit Mustern aus Löchern, einen RAM, ein PROM und ein EPROM, ein FLASH-EPROM, irgendeinen anderen Speicherchip oder -kassette, eine Trägerwelle, wie sie nachstehend beschrieben wird, oder irgendein anderes Medium, von welchem ein Computer lesen kann.

[0050] Verschiedene Formen maschinenlesbarer Medien könnten beim Übertragen einer oder mehre-

rer Sequenzen eines oder mehrerer Befehle an den Prozessor **304** für die Ausführung involviert sein. Beispielsweise könnten die Befehle ursprünglich auf einer Magnetplatte eines entfernten Computers getragen werden bzw. vorhanden sein. Der entfernte Computer kann die Befehle in seinen dynamischen Speicher laden und die Befehle über eine Telefonleitung unter Verwendung eines Modems senden. Ein lokales Modem des Computersystems **300** kann die Daten über die Telefonleitung empfangen und einen Infrarotsender verwenden, um die Daten in ein Infrarotsignal umzuwandeln. Ein Infrarotdetektor kann die Daten, die durch das Infrarotsignal getragen werden, empfangen und eine geeignete Schaltung kann die Daten auf den Bus **302** bringen. Der Bus **302** überträgt die Daten an den Hauptspeicher **306**, von welchem der Prozessor **304** die Befehle holt und ausführt. Die durch den Hauptspeicher **306** empfangenen Befehle können wahlweise entweder vor oder nach der Ausführung durch den Prozessor **304** auf der Speichereinrichtung **310** gespeichert werden.

[0051] Das Computersystem **300** umfaßt auch eine Kommunikationsschnittstelle **318**, die mit dem Bus **302** verbunden ist. Die Kommunikationsschnittstelle **318** stellt eine Datenkommunikationsverbindung mit zwei Wegen zu einer Netzwerkverbindung **320** bereit, die mit einem lokalen Netzwerk **322** verbunden ist. Beispielsweise kann die Kommunikationsschnittstelle **318** eine ISDN-Karte (Integrated Services Digital Network-Karte) oder ein Modem sein, um eine Datenkommunikationsverbindung mit einem entsprechenden Typ einer Telefonleitung bereitzustellen. Als ein weiteres Beispiel kann die Kommunikationsschnittstelle **318** eine Karte für ein Nahbereichsnetz (Local Area Network – LAN) sein, um eine Datenkommunikationsverbindung mit einem kompatiblen LAN bereitzustellen. Drahtlose Verbindungen können ebenfalls implementiert sein. In irgendeiner solchen Implementierung sendet die Kommunikationsschnittstelle **318** elektrische, elektromagnetische oder optische Signale, welche digitale Datenströme tragen, die verschiedene Typen von Information repräsentieren, und empfängt solche Signale.

[0052] Die Netzwerkverbindung **320** stellt typischerweise eine Datenkommunikation durch eines oder mehrere Netzwerke mit anderen Dateneinrichtungen bereit. Beispielsweise kann die Netzwerkverbindung **320** eine Verbindung durch ein lokales Netzwerk **322** mit einem Host-Computer **324** oder einer Datenausrüstung bereitstellen, die durch einen Internet Service Provider (ISP) **326** betrieben wird. Der ISP **326** stellt seinerseits Datenkommunikationsdienste über das weltweite Kommunikationsnetz für Paketdaten bereit, welches inzwischen üblicherweise als das "Internet" **328** bezeichnet wird. Das lokale Netzwerk **322** und das Internet **328** verwenden beide elektrische, elektromagnetische oder optische Signale, welche digitale Datenströme tragen. Die Signale durch die verschiedenen Netzwerke und die Signale auf der Netzwerkverbindung **320** und durch die Kommunikationsschnittstelle **318**, welche die digitalen Daten zu und von dem Computersystem **300** tragen, sind beispielhafte Formen von Tragerwellen, welche die Information transportieren.

[0053] Das Computersystem **300** kann durch das Netzwerk (die Netzwerke), die Netzwerkverbindung **320** und die Kommunikationsschnittstelle **318** Nachrichten senden und Daten empfangen, einschließlich Programmcode. Im Beispiel des Internets könnte ein Server **330** einen angeforderten Code für ein Anwendungsprogramm über das Internet **328**, dem ISP **326**, das lokale Netzwerk **322** und die Kommunikationsschnittstelle **318** übermitteln.

[0054] Der empfange Code kann durch einen Prozessor **304** beim Empfang ausgeführt werden und/oder kann in der Speichereinrichtung **310** oder einem anderen nichtflüchtigen Speicher für eine spätere Ausführung gespeichert werden. Auf diese Weise kann das Computersystem **300** Anwendungscode in Form einer Trägerwelle erhalten.

[0055] Die vorstehenden Beschreibungen von Ausführungsformen der vorliegenden Erfindung sind für Zwecke der Veranschaulichung dargeboten worden, um Fachleute auf diesem Gebiet in die Lage zu versetzen, die Erfindung zu verstehen und zu implementieren. Es sind im Kontext bestimmte Anwendungen und deren Erfordernisse bereitgestellt worden, sollen jedoch nicht erschöpfend sein und die Erfindung auch nicht auf die dargestellten Formen beschränken. Dementsprechend liegen viele Modifikationen und Variationen für Fachleute auf der Hand und der Schutzumfang der vorliegenden Erfindung wird durch die anhängenden Ansprüche und deren Äquivalente definiert.

Patentansprüche

1. Computer-implementiertes Verfahren, welches aufweist:
Vor dem Ausführen eines typenorientierten, in einer Maschinensprache bzw. Assemblersprache geschriebenen Programms, welches einen bestimmten Befehl aufweist, der eine Programm-Methode aufruft, die, wenn sie ausgeführt wird, die Schritte ausführt, welche aufweisen:
Feststellen, ob der betreffende Befehl eine bestimmte Art eines Befehles ist,
wenn der betreffende Befehl von der bestimmten Art ist, davon Absehen, vor der Ausführung des Programms in Assemblersprache Typen von Argumenten zu überprüfen, die sich in einem Stapel befinden, wenn der betreffende Befehl ausgeführt wird, und
falls der betreffende Befehl nicht von der bestimmten Art ist, dann vor der Ausführung des Programms in der Maschinensprache Überprüfen der Typen der Ar-

gumente, die sich im Stapel befinden, wenn der betreffende Befehl ausgeführt wird, und, falls die Typen der Argumente nicht zu Typen formaler Parameter passen, die zu der Programm-Methode gehören, Verhindern, dass das Programm in der Assemblersprache ausgeführt wird.

2. Verfahren nach Anspruch 1, wobei die Schritte weiterhin aufweisen:
Falls der betreffende Befehl von der bestimmten Art ist, dann vor der Ausführung des Programms in Assemblersprache Feststellen, ob irgendeines der Argumente von einem Grundtyp ist (primitive) und, falls irgendeines der Argumente von einem Grundtyp ist, dann Verhindern, dass das Programm in Assemblersprache ausgeführt wird.

3. Verfahren nach Anspruch 1 oder 2, wobei die Schritte weiterhin aufweisen:
Wenn der betreffende Befehl von der bestimmten Art ist, dann vor Ausführung des Programms in Assemblersprache Feststellen, ob ein Objekt, bezüglich dessen die Programm-Methode aufgerufen wird, sich auf dem Stapel befindet, wenn der betreffende Befehl ausgeführt wird, und, falls das Objekt sich nicht auf dem Stapel befindet, wenn der betreffende Befehl ausgeführt wird, Verhindern, dass das Programm in Assemblersprache ausgeführt wird.

4. Verfahren nach einem der vorstehenden Ansprüche, wobei die Schritte weiterhin aufweisen:
Falls der betreffende Befehl von der bestimmten Art ist, dann vor Ausführung des Programms in Assemblersprache Feststellen, ob eine Anzahl von Argumenten, die sich auf einem Stapel befinden, wenn der betreffende Befehl ausgeführt wird, eine Anzahl von formaler Parameter erfüllt, die zu der Programm-Methode gehören, und, falls die Anzahl von Argumenten die Anzahl formaler Parameter nicht erfüllt, die zu der Programm-Methode gehören, Verhindern, dass das Programm in Assemblersprache ausgeführt wird.

5. Verfahren nach Anspruch 1, wobei die Schritte weiterhin aufweisen:
Falls der betreffende Befehl von der bestimmten Art ist, Zulassen, dass das Programm in Assemblersprache ausgeführt wird, unabhängig davon, ob der Typ der Argumente mit dem Typ der formalen Parameter übereinstimmt.

6. Verfahren nach einem der vorstehenden Ansprüche, welches weiterhin aufweist:
Während der Ausführung des Programms in Assemblersprache und falls der betreffende Befehl von der bestimmten Art ist, Überprüfen der Typen von Argumenten, die sich auf dem Stapel befinden, wenn der betreffende Befehl ausgeführt werden soll, und, falls die Typen von Argumenten nicht zu den Typen der formalen Parameter passen, die zu der Programm-Methode gehören, Verhindern, dass der betreffende Befehl ausgeführt wird.

7. Verfahren nach einem der vorstehenden Ansprüche, welches aufweist:
Während der Ausführung des Programms in Assemblersprache, falls der betreffende Befehl von der bestimmten Art ist, Umformen jedes einzelnen Argumentes auf dem Stapel, welches einem formalen Parameter entspricht, der nicht von dem Typ "Objekt" ist, so dass es dem Typ des formalen Parameters entspricht.

8. Verfahren nach einem der vorstehenden Ansprüche, wobei das Feststellen, ob der betreffende Befehl eine bestimmte Art von Befehl ist, das Feststellen aufweist, ob der betreffende Befehl ein Java-Bytecode-Befehl "invokevirtual" ist, und wobei der betreffende Befehl, nicht die bestimmte Art von Befehl ist, falls der betreffende Befehl ein Java-Bytecode-Befehl "invokevirtual" ist.

9. Verfahren nach einem der vorstehenden Ansprüche, wobei das Programm in Assemblersprache Java-Bytecodes aufweist.

10. Computerlesbares Medium, welches ein oder mehrere Sequenzen von Befehlen trägt, wobei diese Befehle, wenn sie durch einen oder mehrere Prozessoren ausgeführt werden, bewirken, dass der eine oder die mehreren Prozessoren das Verfahren nach einem der vorstehenden Ansprüche ausführen.

11. Vorrichtung, welche aufweist:
Einen Mechanismus, um vor der Ausführung eines in Maschinensprache bzw. Assemblersprache geschriebenen Programms, welches einen bestimmten Befehl enthält, der bei seiner Ausführung eine Programm-Methode aufruft, festzustellen, ob der betreffende Befehl eine bestimmte Art von Befehl ist,
einen Mechanismus, um davon abzusehen, dass vor der Ausführung des Programms in Assemblersprache Typen von Argumenten überprüft werden, die sich in einem Stapel befinden, wenn der betreffende Befehl ausgeführt wird, ob der betreffende Befehl von einer bestimmten Art ist, und
einen Mechanismus, um vor der Ausführung des Maschinensprache-Programms die Arten von Argumenten zu überprüfen, die sich auf dem Stapel befinden, wenn der betreffende Befehl ausgeführt wird, und, für den Fall, dass die Arten von Argumenten nicht mit formalen Parametern übereinstimmen, welche zu der Programm-Methode gehören, verhindern, dass das Programm in Assemblersprache ausgeführt wird, wenn der betreffende Befehl nicht von der bestimmten Art ist.

12. Vorrichtung nach Anspruch 11, welche weiterhin aufweist:
Einen Mechanismus, um vor der Ausführung des Programms in Assemblersprache festzustellen, ob ir-

gendeines der Argumente von einem Grundtyp (primitive) ist, und, falls eines der Argumente von einem Grundtyp ist, Verhindern der Ausführung des Programms in Assemblersprache.

13. Vorrichtung nach Anspruch 11 oder 12, welche weiterhin aufweist:
Einen Mechanismus, um vor der Ausführung des Programms in Assemblersprache festzustellen, ob ein Objekt, bezüglich dessen die Programmiermethode aufgerufen werden soll, sich auf dem Stapel befindet, wenn der bestimmte Befehl ausgeführt wird, und dann, wenn das Objekt sich nicht auf dem Stapel befindet, wenn der betreffende Befehl ausgeführt wird, Verhindern, dass das Programm in Assemblersprache ausgeführt wird, wenn der betreffende Befehl von der bestimmten Art ist.

14. Vorrichtung nach einem der Ansprüche 11 bis 13, welche weiterhin aufweist:
Einen Mechanismus, um vor der Ausführung des Programms in Assemblersprache zu bestimmen, ob eine Anzahl von Argumenten, die sich auf einem Stapel befinden, wenn der bestimmte Befehl ausgeführt wird, mit einer Anzahl formaler Parameter übereinstimmt, die zu der Programm-Methode gehören, und, falls die Anzahl der Argumente nicht mit der Anzahl formaler Parameter übereinstimmt, welche zu der Programm-Methode gehören, Verhindern, dass das Programm in Assemblersprache ausgeführt wird, wenn der bestimmte Befehl von der bestimmten Art ist.

15. Vorrichtung nach Anspruch 11, welche weiterhin aufweist:
Einen Mechanismus, um zuzulassen, dass das Programm in Assemblersprache ausgeführt wird, unabhängig davon, ob die Art der Argumente zu den formalen Parametern passt, wenn der betreffende Befehl von der bestimmten Art ist.

16. Vorrichtung nach einem der Ansprüche 11 bis 15, welche weiterhin aufweist:
Einen Mechanismus, um während der Ausführung des Programms in Assemblersprache die Typen von Argumenten zu überprüfen, die sich auf dem Stapel befinden, wenn der bestimmte Befehl ausgeführt werden soll, und dann, falls die Typen von Argumenten nicht mit den Typen formaler Parameter übereinstimmen, die zu der Programmiermethode gehören, Verhindern, dass der bestimmte Befehl ausgeführt wird, wenn der bestimmte Befehl von der besonderen Art ist.

17. Vorrichtung nach einem der Ansprüche 11 bis 16, welche weiterhin aufweist:
Einen Mechanismus, um während der Ausführung des Programms in Assemblersprache für jedes einzelne Argument auf dem Stapel, welches einem formalen Parameter entspricht, der nicht von dem Typ "Objekt" ist, Umformen eines betreffenden Argumentes, so dass es den formalen Parametertyp hat, wenn der betreffende Befehl von der besonderen Art ist.

18. Vorrichtung nach einem der Ansprüche 11 bis 17, wobei der Mechanismus zum Feststellen, ob der betreffende Befehl von einer bestimmten Art eines Befehles ist, einen Mechanismus aufweist, um festzustellen, ob der betreffende Befehl ein Java-Bytecode-Befehl "invokevirtual" ist, und wobei der betreffende Befehl nicht die bestimmte Art von Befehl ist, wenn der betreffende Befehl ein Java-Bytecode-Befehl "invokevirtual" ist.

19. Vorrichtung nach einem der Ansprüche 11 bis 18, wobei das Programm in Assemblersprache Java-Bytecodes aufweist.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen

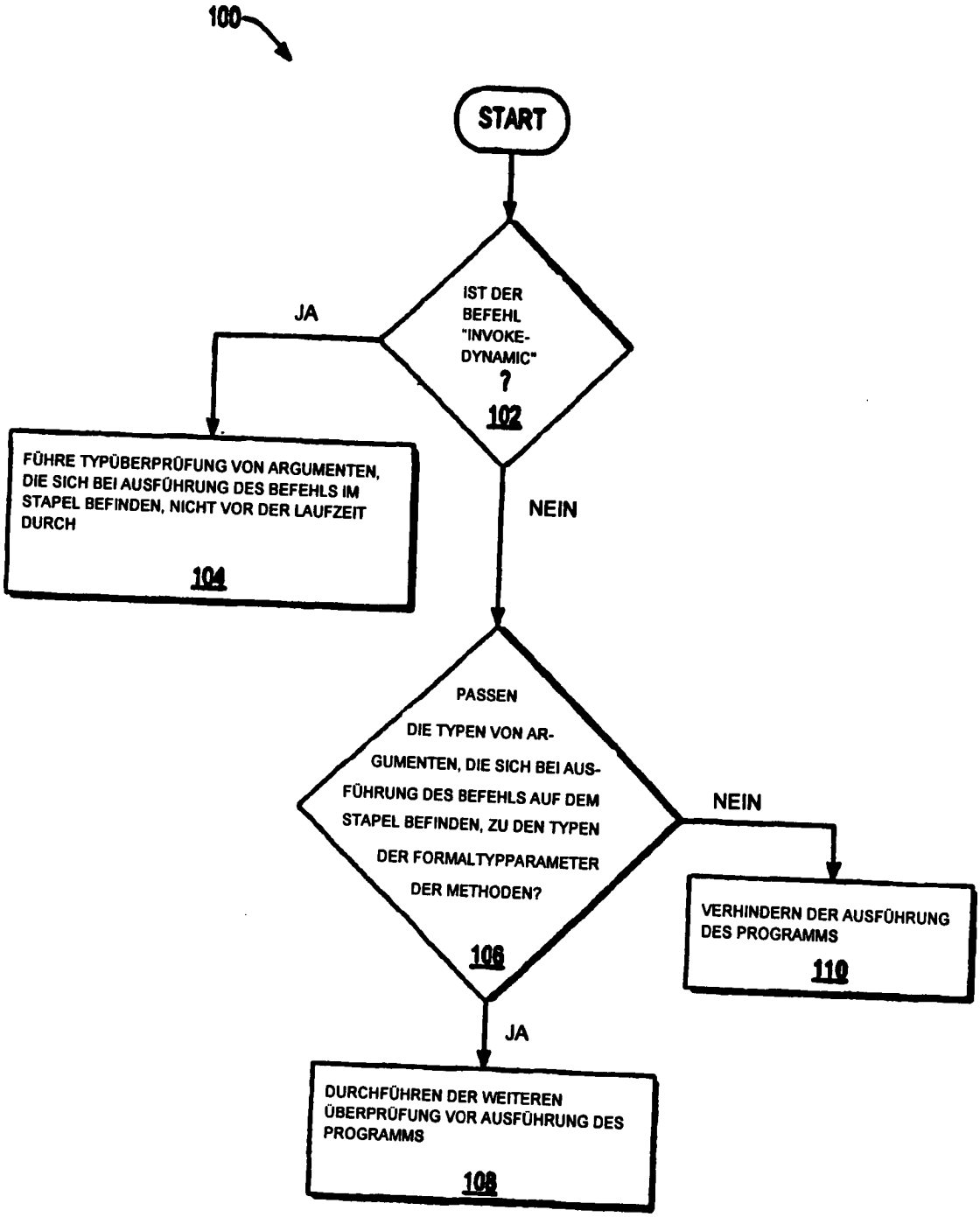

Fig. 1

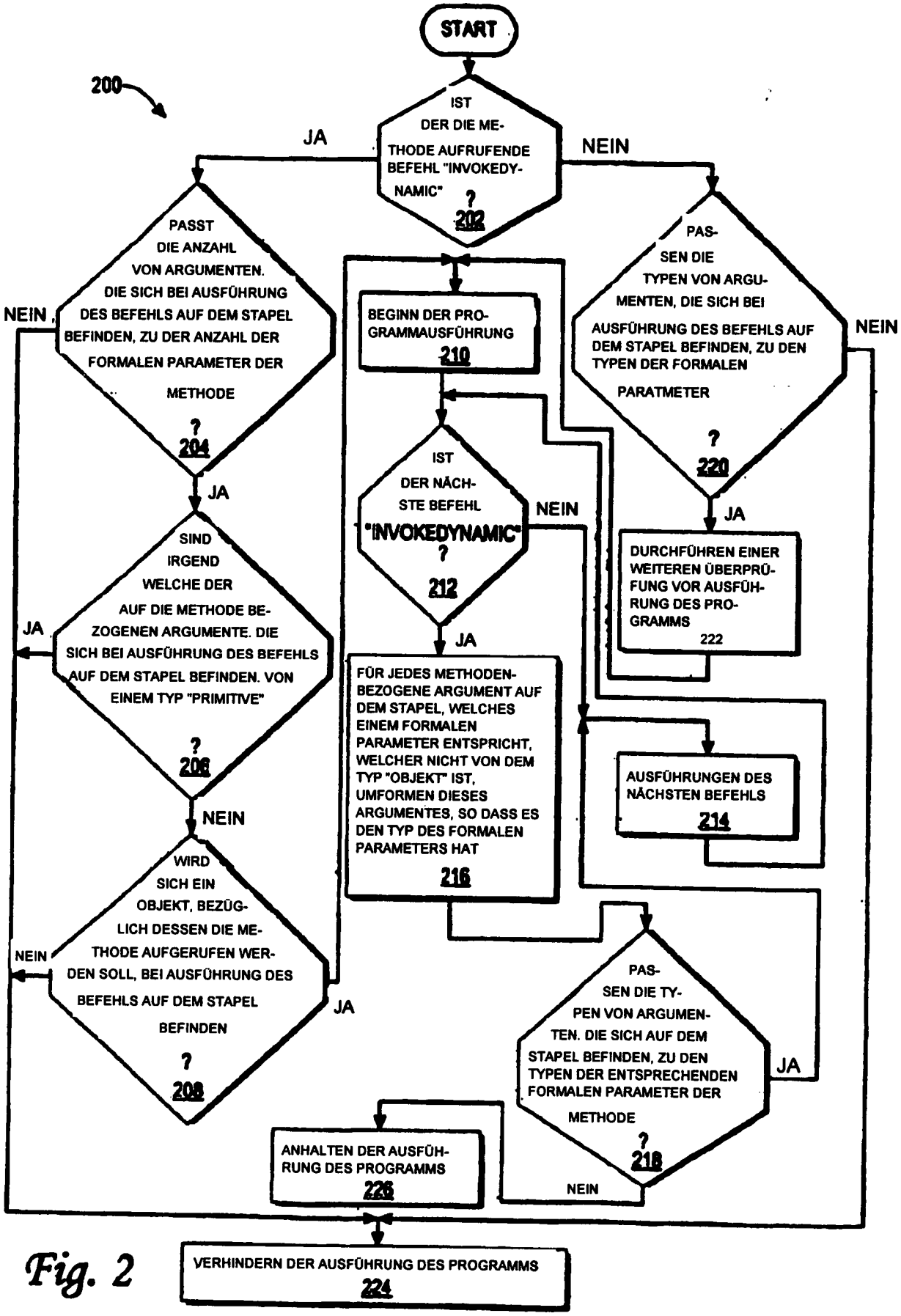
START
200
IST DER DIE METHODE AUFRUFENDE BEFEHL "INVOKEDYNAMIC" ? 202
JA
NEIN
PASST DIE ANZAHL VON ARGUMENTEN, DIE SICH BEI AUSFÜHRUNG DES BEFEHLS AUF DEM STAPEL BEFINDEN, ZU DER ANZAHL DER FORMALEN PARAMETER DER METHODE ? 204
NEIN
JA
SIND IRGEND WELCHE DER AUF DIE METHODE BEZOGENEN ARGUMENTE, DIE SICH BEI AUSFÜHRUNG DES BEFEHLS AUF DEM STAPEL BEFINDEN, VON EINEM TYP "PRIMITIVE" ? 206
JA
NEIN
WIRD SICH EIN OBJEKT, BEZÜGLICH DESSEN DIE METHODE AUFGERUFEN WERDEN SOLL, BEI AUSFÜHRUNG DES BEFEHLS AUF DEM STAPEL BEFINDEN ? 208
NEIN
JA
BEGINN DER PROGRAMMAUSFÜHRUNG 210
IST DER NÄCHSTE BEFEHL "INVOKEDYNAMIC" ? 212
NEIN
JA
FÜR JEDES METHODENBEZOGENE ARGUMENT AUF DEM STAPEL, WELCHES EINEM FORMALEN PARAMETER ENTSPRICHT, WELCHER NICHT VON DEM TYP "OBJEKT" IST, UMFORMEN DIESES ARGUMENTES, SO DASS ES DEN TYP DES FORMALEN PARAMETERS HAT 216
PASSEN DIE TYPEN VON ARGUMENTEN, DIE SICH BEI AUSFÜHRUNG DES BEFEHLS AUF DEM STAPEL BEFINDEN, ZU DEN TYPEN DER FORMALEN PARATMETER ? 220
NEIN
JA
DURCHFÜHREN EINER WEITEREN ÜBERPRÜFUNG VOR AUSFÜHRUNG DES PROGRAMMS 222
AUSFÜHRUNGEN DES NÄCHSTEN BEFEHLS 214
PASSEN DIE TYPEN VON ARGUMENTEN, DIE SICH AUF DEM STAPEL BEFINDEN, ZU DEN TYPEN DER ENTSPRECHENDEN FORMALEN PARAMETER DER METHODE ? 218
JA
NEIN
ANHALTEN DER AUSFÜHRUNG DES PROGRAMMS 226
VERHINDERN DER AUSFÜHRUNG DES PROGRAMMS 224

Fig. 2

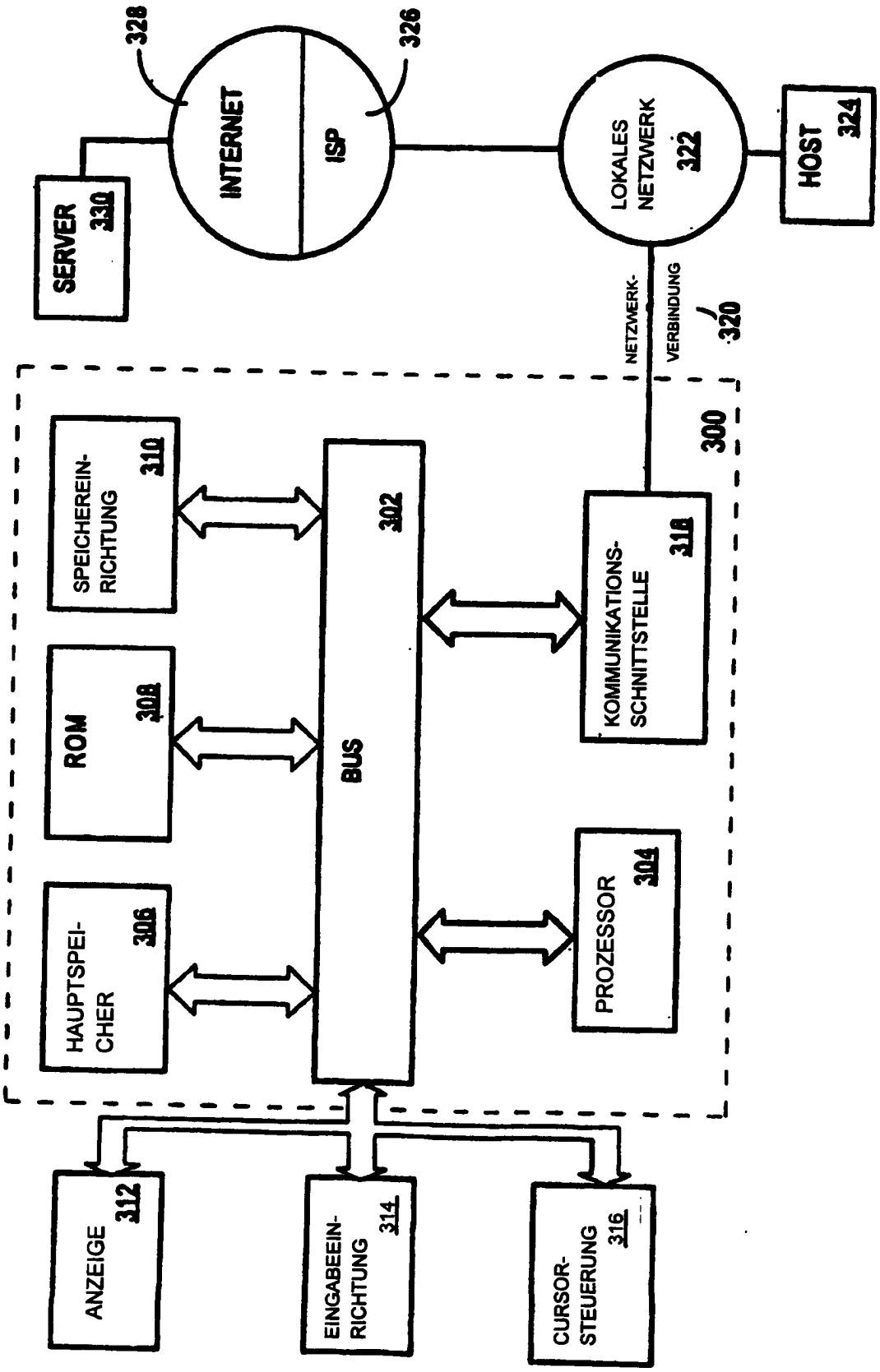
328
INTERNET
ISP
326
SERVER
330
LOKALES NETZWERK
322
HOST
324
NETZWERK-VERBINDUNG
320
300
SPEICHEREIN-RICHTUNG
310
ROM
308
HAUPTSPEI-CHER
306
BUS
302
KOMMUNIKATIONS-SCHNITTSTELLE
318
PROZESSOR
304
ANZEIGE
312
EINGABEEIN-RICHTUNG
314
CURSOR-STEUERUNG
316

FIG. 3