PCT WELTORGANISATION FÜR GEISTIGES EIGENTUM
Internationales Büro

INTERNATIONALE ANMELDUNG VERÖFFENTLICHT NACH DEM VERTRAG ÜBER DIE INTERNATIONALE ZUSAMMENARBEIT AUF DEM GEBIET DES PATENTWESENS (PCT)

(51) Internationale Patentklassifikation 6 :		(11) Internationale Veröffentlichungsnummer: WO 95/02863
G06F 11/00	A1	(43) Internationales Veröffentlichungsdatum: 26. Januar 1995 (26.01.95)

(21) Internationales Aktenzeichen: PCT/DE94/00781

(22) Internationales Anmeldedatum: 7. Juli 1994 (07.07.94)

(30) Prioritätsdaten:
P 43 23 787.8 15. Juli 1993 (15.07.93) DE

(71) Anmelder *(für alle Bestimmungsstaaten ausser US)*: SIEMENS AKTIENGESELLSCHAFT [DE/DE]; Wittelsbacherplatz 2, D-80333 München (DE).

(72) Erfinder; und
(75) Erfinder/Anmelder *(nur für US)*: JÜTTNER, Peter [DE/DE]; Häherweg 3, D-81827 München (DE). KOLB, Sebald [DE/DE]; Clemens-Schöps-Strasse 2a, D-85521 Ottobrunn (DE). SIEBER, Stefan [DE/DE]; Waldheimplatz 64, D-81739 München (DE). ZIMMERER, Peter [DE/DE]; Oskar-Maria-Graf-Ring 37, D-81737 München (DE).

(81) Bestimmungsstaaten: US, europäisches Patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Veröffentlicht
Mit internationalem Recherchenbericht.

(54) Title: PROCESS FOR TESTING AN OBJECT-ORIENTED PROGRAMME

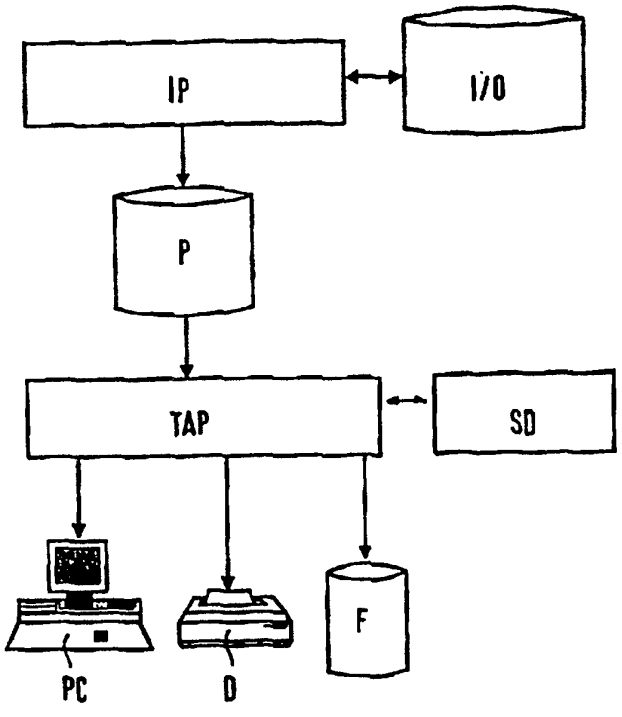
(54) Bezeichnung: VERFAHREN ZUM TEST EINES OBJEKTORIENTIERTEN PROGRAMMS

(57) Abstract

In order to test object-oriented programmes, a process instruments the programme locations in which communications between objects are involved. When the programme is run, all communication processes between objects are recorded with their corresponding parameters in a protocol file. This protocol file is then processed in an appropriate manner, so that the results of the object-oriented design may be directly compared with the programme operation, for example by means of a graphic representation.

(57) Zusammenfassung

Mit der Erfindung wird ein Verfahren vorgestellt, das zum Test objektorientierter Programme diejenigen Programmstellen, welche von der Kommunikation der Objekte untereinander betroffen sind, instrumentiert. Beim Ablauf des Programmes werden sämtliche Kommunikationsvorgänge zwischen den Objekten mit zugehörigen Parametern in einer Protokolldatei aufgezeichnet. Diese Protokolldatei wird anschließend geeignet aufbereitet, sodaß z.B. durch eine grafische Darstellung direkt die Ergebnisse des objektorientierten Designs mit dem Ablauf des Programmes verglichen werden können.



LEDIGLICH ZUR INFORMATION

Codes zur Identifizierung von PCT-Vertragsstaaten auf den Kopfbögen der Schriften, die internationale Anmeldungen gemäss dem PCT veröffentlichen.

AT	Österreich	**GA**	Gabon	**MR**	Mauretanien
AU	Australien	**GB**	Vereinigtes Königreich	**MW**	Malawi
BB	Barbados	**GE**	Georgien	**NE**	Niger
BE	Belgien	**GN**	Guinea	**NL**	Niederlande
BF	Burkina Faso	**GR**	Griechenland	**NO**	Norwegen
BG	Bulgarien	**HU**	Ungarn	**NZ**	Neuseeland
BJ	Benin	**IE**	Irland	**PL**	Polen
BR	Brasilien	**IT**	Italien	**PT**	Portugal
BY	Belarus	**JP**	Japan	**RO**	Rumänien
CA	Kanada	**KE**	Kenya	**RU**	Russische Föderation
CF	Zentrale Afrikanische Republik	**KG**	Kirgisistan	**SD**	Sudan
CG	Kongo	**KP**	Demokratische Volksrepublik Korea	**SE**	Schweden
CH	Schweiz	**KR**	Republik Korea	**SI**	Slowenien
CI	Côte d'Ivoire	**KZ**	Kasachstan	**SK**	Slowakei
CM	Kamerun	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Tschad
CS	Tschechoslowakei	**LU**	Luxemburg	**TG**	Togo
CZ	Tschechische Republik	**LV**	Lettland	**TJ**	Tadschikistan
DE	Deutschland	**MC**	Monaco	**TT**	Trinidad und Tobago
DK	Dänemark	**MD**	Republik Moldau	**UA**	Ukraine
ES	Spanien	**MG**	Madagaskar	**US**	Vereinigte Staaten von Amerika
FI	Finnland	**ML**	Mali	**UZ**	Usbekistan
FR	Frankreich	**MN**	Mongolei	**VN**	Vietnam

Verfahren zum Test eines objektorientierten Programms

In vermehrtem Umfang hat sich bei der Erstellung umfangreicher Programme die Technik des objektorientierten Programmierens etabliert. Diese Technik bietet sowohl bei der Erstellung der Programme als auch beim Ablauf verschiedenste Vorteile.

Dem Programmierer wird beispielsweise das strukturierte Programmieren erleichtert. Weiterhin ist es einfach, Klassen aus bereits vorhandenen Programmen für neue Programme zu übernehmen. Da Objekte zur Laufzeit erzeugt und auch wieder gelöscht werden können, wird zu jedem Zeitpunkt des Ablaufs nur der aktuell benötigte Speicher beansprucht. Im Zusammenhang mit der Erstellung der objektorientierten Programme hat sich die Technik des OOD (object oriented design) [1] durchgesetzt. Diese Technik macht sich grafische Eingabemöglichkeiten bei der Erstellung von Programmen zu Nutze und ist in [2]-[5] näher erläutert.

Schwierigkeiten bestehen zur Zeit beim Test objektorientierter Programme. Dies liegt insbesondere an der dynamischen Struktur beim Ablauf dieser Programme. Da während des Programmablaufs Objekte erzeugt und wieder gelöscht werden, sowie Methoden aufgerufen werden, ist es schwierig, bei einem Test alle Objekte und Methodenaufrufe bezüglich ihrer Richtigkeit und ihrer Parameter zu überprüfen.

Verfahren und Werkzeuge, die einen umfangreichen Test von objektorientierten Programmen zulassen, sind derzeit nicht bekannt.

Der im folgenden erläuterten Erfindung liegt die Aufgabe zugrunde, ein Verfahren zum Test eines objektorientierten Programmes anzugeben.

Diese Aufgabe wird gemäß den Merkmalen des Patentanspruchs 1 gelöst.

Weiterbildungen der Erfindung ergeben sich aus den Unteransprüchen.

Da die Funktion objektorientierter Programme auf der Kommunikation der Objekte untereinander basiert, wird bei der Erfindung in vorteilhafter Weise die Kommunikation der Objekte untereinander analysiert. Dies geschieht erfindungsgemäß dadurch, daß alle kommunikationsrelevanten Stellen in den Programmen, d.h. in den Klassen und in den Methodenaufrufen dahingehend instrumentiert werden, daß die testrelevante Information (z.B. Methodenname, Parameterwerte, Aufrufer und gerufenes Objekt) beim beispielsweise Methodenaufruf oder beim Instantiieren bzw. Löschen eines Objekts in einem separaten Protokoll festgehalten werden können. Ein weiterer Bestandteil des erfindungsgemäßen Verfahrens besteht in der vorteilhaften Auswertung einer erstellten Protokolldatei. Hierdurch wird einem Testenden der Programmablauf übersichtlich dargestellt.

In vorteilhafter Weise geschieht die Veranschaulichung des Programmablaufs in grafischer Art. Wenn man diese Darstellung so wählt, daß sie der grafischen Repräsentation beim object oriented design entspricht, so kann man einfach manuell oder automatisch die Spezifikation des Programmes mit dem tatsächlichen Ablauf vergleichen.

Die erfindungsgemäß vorgesehene Instrumentierung für Methoden bietet den Vorteil, daß man beim Test leicht überprüfen kann,

mit welchen Parametern welche Methode aufgerufen wurde. Weiterhin kann man statistisch auswerten, wie oft eine Methode aufgerufen wurde. Durch die Instrumentierung der Methodenaufrufe gewinnt man Informationen über die übergebenen Parameter und darüber, mit welchen Methodenaufrufen welche Methoden in welchen Objekten aufgerufen werden. Durch die Instrumentierung von beispielsweise den Methodenaufrufstellen in den verschiedenen Objekten gewinnt man Informationen, die man vor allen Dingen für die Analyse verketteter Methodenaufrufe benötigt. Verkettete Methodenaufrufe sind solche Methodenaufrufe, die beispielsweise von einem Objekt ausgehen und in einem weiteren Objekt wiederum einen Methodenaufruf bewirken und welcher dann in einem nächsten Objekt wieder eine Methode aufrufen kann.

Ein Vorteil des erfindungsgemäßen Verfahrens besteht vor allen Dingen in der Erstellung von Objektbilanzen. Damit läßt sich statistisch leicht feststellen, welche und wie viele Objekte von welchen Klassen während des Programmablaufs instantiiert wurden. Weiterhin kann man daraus ersehen, ob alle aufgerufenen Objekte auch wirklich benutzt wurden und ob mit den Methoden die richtigen Objekte angesprochen wurden. Man erkennt auch leicht, ob alle instantiierten Objekte auch wieder gelöscht worden sind.
Weiterhin gewinnt man so die Möglichkeit, die Tätigkeit des Programmierers überwachen zu können. Häufig gibt es in größeren Unternehmen an Programmierer die Anweisung, bereits erstellte Klassen oder Programmteile in neuen Programmen wieder zu verwenden. Mit dem erfindungsgemäßen Verfahren kann man nun leicht überprüfen, ob die Klassen, die in ein neues Programm eingebunden wurden, auch verwendet werden, oder ob sie nur einen quasi unnötigen Ballast im Programmcode darstellen.

Im folgenden wird die Erfindung anhand von Figuren und Beispielen weiter erläutert.

Figur 1 zeigt als Beispiel ein Blockschaltbild des erfindungsgemäßen Verfahrens.
Figur 2 zeigt die Spezifikation eines objektorientierten Programmes.
Figur 3 zeigt die Realisierung des Programmes von Figur 2.

Figur 1 zeigt als Beispiel ein Blockschaltbild des erfindungsgemäßen Verfahrens. Mit I/O ist die Aus- und Eingabe des spezifizierten und nicht instrumentierten objektorientierten Programmes bezeichnet. Mit IP ist das instrumentierte objektorientierte Programm dargestellt. P bezeichnet die Protokolldatei, die beim Test erstellt wird. TAP stellt das Test-Auswerteprogramm dar, mit dem die beim Test erstellte Protokolldatei aufbereitet wird. SD heißen die Spezifikationsdaten, mit denen das objektorientierte Programm spezifiziert wurde. Über das Test-Auswerteprogramm TAP können die aufbereiteten Protokolldaten beispielsweise auf einem PC auf einem Drucker und in einer Datei F ausgegeben werden. In diesem Beispiel des erfindungsgemäßen Verfahrens geschieht beim Testen eines objektorientierten Programmes folgendes. Die Original- Eingaben, die I/O, werden dem instrumentierten objektorientierten Programm IP zugeführt. Beim Abarbeiten des instrumentierten Programmes IP werden die Eingabedaten I/O verarbeitet und falls im Programm IP instrumentierte Programmteile angesprochen werden, ergeben sich Testprotokoll-Daten, welche in einer Protokolldatei P oder einem Protokoll P abgelegt werden. Die Ausgabe-Daten von IP entsprechen dabei den Ausgabe-Daten des nicht instrumentierten Programmes. Beispielsweise sind in dem objektorientierten Programm Methoden und Aufrufstellen von Methoden in Objekten instrumentiert. Insbesondere können bei in C++ erstellten Programmen beispielsweise alle Memberfunktionen, statische Memberfunktionen, Konstruktoren, Destruktoren, überladene Operatoren, Friends und Casts instrumentiert sein.

Anschließend wird das Protokoll P beispielsweise durch das Test-Auswerteprogramm TAP ausgewertet. Im Zuge der Auswertung kann die Protokolldatei beispielsweise grafisch dargestellt werden. Eine besonders geeignete Form der grafischen Darstellung bietet jenes grafische Format, welches beim Spezifizieren des Programmes im Zuge des objektorientierten Designs verwendet wurde. Man kann so leicht im Vergleich von Protokoll-Daten mit den Spezifikationsdaten SD Unterschiede zwischen Programm und zugehöriger Spezifikation feststellen. Es besteht die Möglichkeit, diese Unterschiede am Bildschirm oder am Drukker auszugeben oder auch in einer Datei abzuspeichern, um sie weiter verarbeiten zu können. Bei der Analyse des Protokolls P durch das Test-Auswerteprogramm TAP sind auch noch andere Auswertemethoden denkbar. Beispielsweise kann die Objektlandschaft angezeigt werden. Das bedeutet zu einem aktuellen Zeitpunkt, oder für einen Zeitabschnitt können alle existierenden Objekte ausgegeben werden. Weiterhin ist es denkbar, einen Methodenaufrufbaum darzustellen. Dieser Methodenaufrufbaum zeigt alle weiteren Methodenaufrufe an, die ein selektierter Methodenaufruf zur Folge hat. Weiterhin kann man sich denken, die Objektlandschaft mit Aufrufbeziehung anzeigen zu lassen, wobei alle jemals existenten Objekte einschließlich ihrer gegenseitigen Methodenaufrufe angezeigt werden können. Zusätzlich ist es denkbar, die Klassenlandschaft mit ihren Aufrufbeziehungen darzustellen. Dies hat zur Folge, daß alle existierenden Klassen einschließlich ihrer Methodenaufruf-Beziehungen untereinander gezeigt werden. Alle diese vorab aufgezählten Darstellungen werden in den verschiedenen OOD-Methoden zur Spezifikation eines objektorientierten Programmes verwendet. In analoger Weise kann man die Protokolldatei auf statistische Merkmale hin analysieren. Beispielsweise können alle Objekte gezählt werden. Es können sämtliche Methodenaufrufe gezählt werden, und man kann über eine Analyse der maximal und minimal existierenden

Objekte bzw. Methodenaufrufe Aussagen über die Programm-Komplexität machen.

Figur 2 zeigt ein objektorientiertes Programm in grafischer Darstellung nach Art des objektorientierten Designs (OOD). Es sind drei Objekte o1 bis o3 aus zugehörigen Klassen K1 bis K3 dargestellt. o1 ruft beim Objekt o2 eine Methode ma mit den Parametern x und y auf. o1 ruft auch beim Objekt o3 eine Methode mb mit den Parametern z, x und c auf. Figur 2 zeigt hier beispielsweise die Spezifikationsform des objektorientierten Programmes. Für diese Spezifikation folgt nun der Source-Code des objektorientierten Programmes.

1. C++-Source-Code

Im folgenden wird das Beispiel in C++-Syntax beschrieben.

```
class K1 {
        public:

                void m() {

                        K2 o2;                          // standard constructor call
                        K3 o3;                          // standard constructor call
                        int     x = 1,
                                y = 2;
                        float   z = 1.5;
                        char    c = '*';

                        ...
                        // other statements of the method
                        o2.ma(x, y);
                        ...
                        // other statements of the method
                        o3.mb(z, x, c);
                        ...
                        // other statements of the method

                }
```

```
}


class K2 {
        public:

                void ma(int p, int q) {
                        ...
                        // statements of the method
                }
}


class K3 {
        public:

                void mb(float l, int m, char n) {
                        ...
                        // statements of the method
                }
}


void main() {
        K1 o1;          // standard constructor call

        ...
                        // other statements
        o1.m();
        ...
                        // other statements
}
```

Für das gezeigte Beispiel folgt nun der nach dem erfindungsgemäßen Verfahren instrumentierte Source-Code. Darin sind die instrumentierten Stellen durch Fettdruck hervorgehoben.

2. Instrumentierter C++-Source-Code

```
class K1 {
        public:

                K1()
                {                       // standard constructor was added
                        protfile << "call K1::K1()\n"
                                 << "in file ... at line ... happened\n";
                        protfile << "parameters:\n";
                        protfile << "no\n";
                }

                void m() {
                        protfile << "call K1::m()\n"
                                 << "in file ... at line ... happened\n";
                        protfile << "parameters:\n";
                        protfile << "no\n";
                        K2 o2;                  // standard constructor call
                        protfile << "dcl K2 o2 in K1::m()\n"
                                 << "in file ... at line ...\n";
                        K3 o3;          // standard constructor call
                        protfile << "dcl K3 o3 in K1::m()\n"
                                << "in file ... at line ...\n";
                        int x = 1,
                            y = 2;
                        float z = 1.5;
                        char c = '*';

                        ...
                        // other statements of the method
                        protfile << "call o2.ma(x, y) from class K2\n"
                                 << "in K1::m() in file ... at line ... started\n";
                        protfile << "parameters:\n";
                        protfile << " int x = " << x << ", int y = " << y << "\n";
                        o2.ma(x, y);
                        protfile << "call o2.ma(x, y) from class K2\n"
                                 << "in K1::m() in file ... at line ... ended\n";
                        protfile << "parameters:\n";
                        protfile << " int x = " << x << ", int y = " << y << "\n";
                        ...
```

```
                    // other statements of the method
                    protfile << "call o3.mb(z, x, c) from class K3\n"
                            << "in K1::m() in file ... at line ... started\n";
                    protfile << "parameters:\n";
                    protfile << " float z = " << z << ", int x = " << x
                            << ", char c = " << c << "\n";
                    o3.mb(z, x, c);
                    protfile << "call o3.mb(z, x, c) from class K3\n" }
                            << "in K1::m() in file ... at line ... ended\n";
                    protfile << "parameters:\n";
                    protfile << " float z = " << z << ", int x = " << x
                            << ", char c = " << c << "\n";
                    ...
                    // other statements of the method

            }

    }

    class K2 {
            public:

                    K2()
                    {                                   // standard constructor was added
                            protfile << "call K2::K2()\n"
                                    << "in file ... at line ... happened\n";
                            protfile << "parameters:\n";
                            protfile << "no\n";
                    }

                    void ma(int p, int q) {
                            protfile << "call K2::ma(p,q)\n"
                                    << "in file ... at line ... happened\n";
                            protfile << "parameters at the begin of K2::ma(p,q):\n";
                            protfile << " int p = " << p << ", int q = " << q << "\n";
                            ...
                            // statements of the method
                            protfile << "parameters at the end of K2::ma(p,q):\n";
                            protfile << " int p = " << p << ", int q = " << q << "\n";
                    }
    }
```

```
class K3 {
        public:

                K3()
                {                       // standard constructor was added
                        protfile << "call K3::K3()\n"
                                << "in file ... at line ... happened\n";
                        protfile << "parameters:\n";
                        protfile << "no\n";
                }

                void mb(float l, int m, char n) {
                        protfile << "call K3::mb(l, m, n)\n"
                                << "in file ... at line ... happened\n";
                        protfile << "parameters at the begin of K3::mb(l, m, n):\n";
                        protfile << " float l = " << l << ", int m = " << m
                                << ", char n = " << c << "\n";
                        ...
                        // statements of the method
                        protfile << "parameters at the end of K3::mb(l, m, n):\n";
                        protfile << " float l = " << l << ", int m = " << m
                                << ", char n = " << c << "\n";
                }
}


void main() {

        {ofstream protfile("protocol");                 // protocol file}
        protfile << "call main()\n"
                << "in file ... at line ... happened\n";
        protfile << "parameters:\n";
        protfile << "no\n";
        K1 o1;                                          // standard constructor call
        protfile << "dcl K1 o1 in main()\n"
                << "in file ... at line ...\n";

        ...
                        // other statements
        protfile << "call o1.m() from class K1\n"
                << "in main() in file ... at line ... started\n";
        protfile << "parameters:\n";
```

```
        protfile << "no\n";
        o1.m();
        protfile << "call o1.m() from class K1\n"
                 << "in main() in file ... at line ... ended\n";
        protfile << "parameters:\n";
        protfile << "no\n";
        ...
                    // other statements
}
```

Es ist hier zwar der Source-Code instrumentiert, doch ist es auch möglich, wenn man genügend Kenntnis über die eingesetzte Hardware hat und den entsprechenden Compiler in Verbindung mit dem Objektcode, der für das objektorientierte Programm erzeugt wird, gut kennt, daß die entsprechende Instrumentierung auch auf der Objektcode-Ebene erfolgt.
Hier ist nur die Instrumentierung des Source-Codes gewählt worden, um ein anschauliches Beispiel darstellen zu können. Dieser instrumentierte Source-Code führt zu einer Ausgabeprotokolldatei, die im Anschluß folgt.

3. Ausgabe der Protokolldatei

Die beim Ablauf des Programmes (main function) erstellte Protokolldatei ist im folgenden aufgelistet. Die eingefügten Leerzeilen dienen ausschließlich der besseren Lesbarkeit.

Die angegebenen Parameterwerte sind natürlich nur exemplarisch, sie hängen logischerweise von den hier nicht im einzelnen aufgeführten Statements in den Methodenrümpfen ab. Ebenso sind bei der Ausgabe von "in file ... at line ... "
die angegebenen drei Punkte ("...") jeweils durch den aktuellen Dateinamen und durch die aktuelle Zeilennummer im (nicht instrumentierten) Source-Code ersetzt.

```
call main()
in file ... at line ... happened
parameters:
no

call K1::K1()
in file ... at line ... happened
parameters:
no

dcl K1 o1 in main()
in file ... at line ...

call o1.m() from class K1
in main() in file ... at line ... started
parameters:
no

call K1::m()
in file ... at line ... happened
parameters:
no

call K2::K2()
in file ... at line ... happened
parameters:
no

dcl K2 o2 in K1::m()
in file ... at line ...

call K3::K3()
in file ... at line ... happened
parameters:
no

dcl K3 o3 in K1::m()
in file ... at line ...

call o2.ma(x, y) from class K2
in K1::m() in file ... at line ... started
parameters:
```

```
int x = 1, int y = 2

call K2::ma(p, q)
in file ... at line ... happened
parameters at the begin of K2::ma(p, q):
int p = 1, int q = 2

parameters at the end of K2::ma(p, q):
int p = 1, int q = 2

call o2.ma(x, y) from class K2
in K1::m() in file ... at line ... ended
parameters:
int x = 1, int y = 2

call o3.mb(z, x, c) from class K3
in K1::m() in file ... at line ... started
parameters:
float z = 1.5, int x = 1, char c = *

call K3::mb(l, m, n)
in file ... at line ... happened
parameters at the begin of K3::mb(l, m, n):
float l = 1.5, int m = 1, char n = *

parameters at the end of K3::mb(l, m, n):
float l = 1.5, int m = 1, char n = *

call o3.mb(z, x, c) from class K3
in K1::m() in file ... at line ... ended
parameters:
float z = 1.5, int x = 1, char c = *

call o1.m() from class K1
in main() in file ... at line ... ended
parameters:
no
```

Mit dem erfindungsgemäßen Verfahren wird nun über die Aufbereitung durch das Test-Auswerteprogramm TAP die grafische Darstellung von Figur 3 erzeugt. Sie zeigt drei Objekte o1, o2 und o4, wobei o1 am Objekt o2 die Methode ma mit den Parametern

x und y aufruft und am Objekt o4 die Methode mb mit z, x und c aufruft. Hier ist deutlich zu erkennen, daß Figur 3 im Vergleich zur Spezifikation, welche in Figur 2 dargestellt ist, eine Abweichung aufweist.

In Figur 3 wird die Methode mb nämlich am Objekt o4, anstatt am Objekt o3 aufgerufen, was nicht mit dem Design übereinstimmt. Analog dazu können beispielsweise weitere Abweichungen vom Design, wie z.B. fehlende oder zusätzliche (und damit eventuelle überflüssige) Methodenaufrufe ermittelt werden.

Wichtig ist es, beim erfindungsgemäßen Verfahren zu beachten, daß die durch die Instrumentierung des Source-Codes beispielsweise ermittelte Protokolldatei nur als Mittel zum Zweck dient. Wesentlich ist es, daß die Protokolldatei weiterverarbeitet wird und geeignet beispielsweise grafisch aufbereitet wird. Geeignet bedeutet in diesem Zusammenhang, daß die Aufbereitung in Affinität zur Darstellung in der Phase des objektorientierten Designs erfolgt. Ein Ziel ist es hierbei beispielsweise, den eigentlichen Programmablauf anschließend direkt mit dem Design, d.h. mit den Ergebnissen aus der OOD-Phase vergleichen zu können.
Insbesondere kann die Instrumentierung des zu testenden Programmes auch vorteilhaft automatisiert durch ein hier nicht näher beschriebenes Instrumentierungs-Programm erfolgen.

Patentansprüche

1. Verfahren zum Test eines objektorientierten Programmes (o1, o2, o3, o4),
a) bei dem mindestens ein Kommunikations-Vorgang (ma(x,y)) zwischen Objekten des Programmes in einem Protokoll (P) protokolliert wird, indem mindestens ein Programmteil des Programmes, welcher vom Kommunikationsvorgang betroffen ist, instrumentiert wird,
b) bei dem das Programm auf einem Rechner mindestens einmal abläuft,
c) und bei dem der Test (TAP) dadurch erfolgt, daß das Protokoll (P) ausgewertet wird.

2. Verfahren nach Anspruch 1,
bei dem das Protokoll (P) automatisch in jenem grafischen Format aufbereitet wird, in welchem die Spezifikation des objektorientierten Programms erfolgte, so daß das Programm gegen die Spezifikation getestet werden kann.

3. Verfahren nach einem der Ansprüche 1 oder 2,
bei dem Methoden (ma(x,y),mb(z,x,c)) instrumentiert werden.

4. Verfahren nach einem der vorhergehenden Ansprüche,
bei dem Methodenaufrufe instrumentiert werden.

5. Verfahren nach einem der obigen Ansprüche,
bei dem als Bilanz aufgerufene und/oder gelöschte Programmelemente erfaßt werden.

6. Verfahren nach Anspruch 5,

bei dem Bilanzen über Klassen, oder Objekte, oder Methoden erstellt werden, indem die Zahl der jeweiligen Aufruf und Löschvorgänge miteinander verglichen wird.

7. Verfahren nach einem der obigen Ansprüche,
bei dem über eine statistische Auswertung die Wiederverwendung mindestens einer in dem objektorientierten Programm eingefügten Klasse überprüft wird.

FIG 1

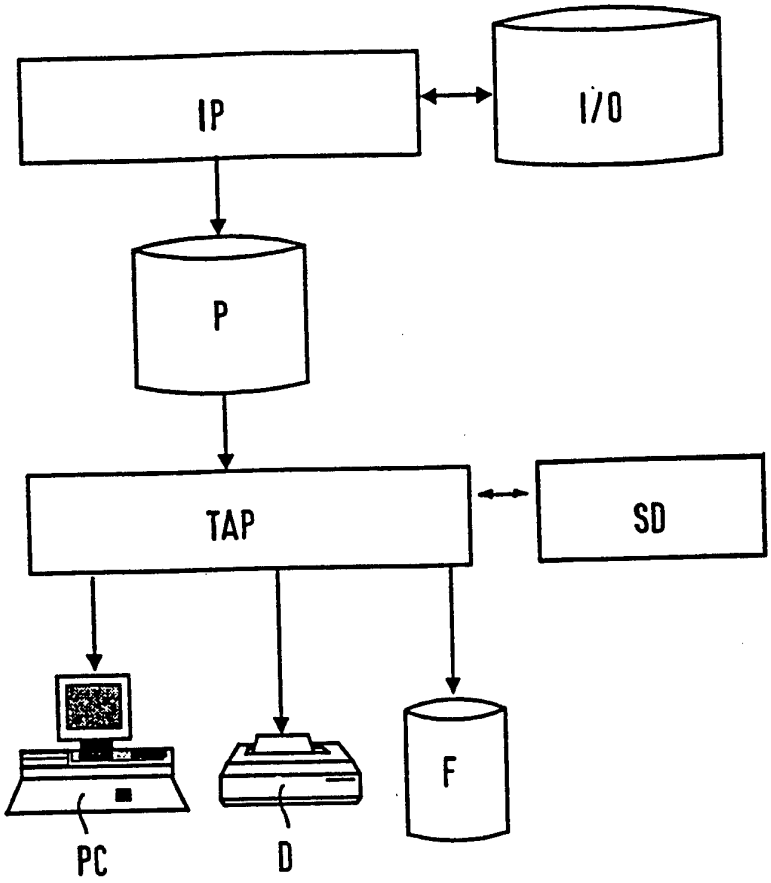


FIG 2

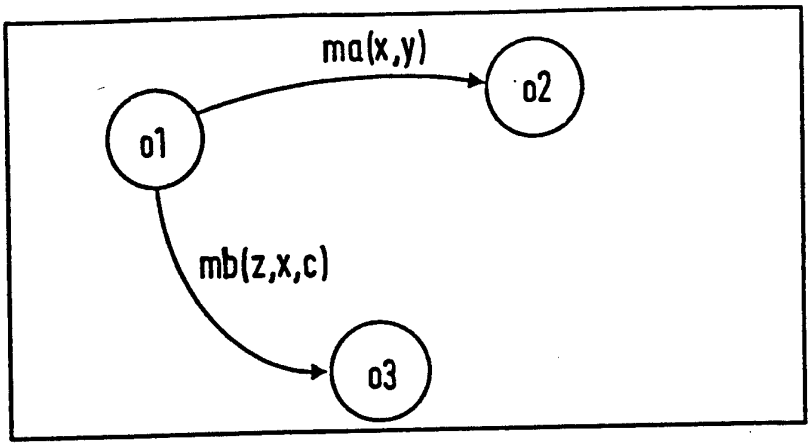
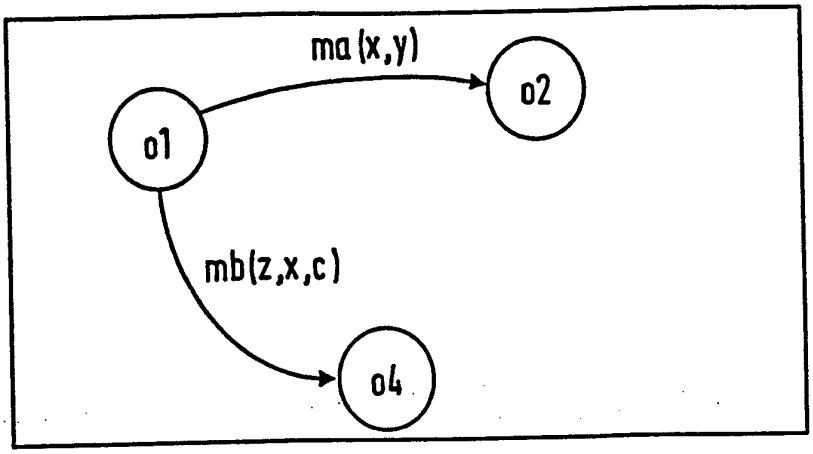


FIG 3



INTERNATIONAL SEARCH REPORT

International Application No
PCT/DE 94/00781

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	COMPUTER, vol.26, no.5, May 1993, LONG BEACH US pages 52 - 61 JAMES R. LARUS 'Efficient Program Tracing' see page 58, right column, line 21 - line 25	1,3-7
Y	---	2
Y	THE SECOND CONFERENCE ON ARTIFICIAL INTELLIGENCE APPLICATIONS, 13 December 1985, MIAMI BEACH, FLORIDA pages 422 - 427 MASAYUKI NUMAO ET AL. 'Visual debugger for Prolog' see page 423, left column, line 16 - line 32; figure 2	2

	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 September 1994	13. 10. 94

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Corremans, G

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	PROCEEDINGS OF THE 15TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 21 May 1993, BALTIMORE, MARYLAND pages 456 - 465 SOUGATA MUKHERJEA ET AL. 'Applying Algorithm Animation Techniques for Program Tracing, Debugging, and Understanding' see page 459, right column, line 38 - line 45 ---	2
A	EP,A,0 213 486 (HITACHI, LTD.) 11 March 1987 see column 7, line 11 - line 19; figure 11 -----	5-7

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/DE 94/00781

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0213486	11-03-87	JP-A- 62057033	12-03-87
		US-A- 4794529	27-12-88

INTERNATIONALER RECHERCHENBERICHT

Internationales Aktenzeichen
PCT/DE 94/00781

A. KLASSIFIZIERUNG DES ANMELDUNGSGEGENSTANDES
IPK 6 G06F11/00

Nach der Internationalen Patentklassifikation (IPK) oder nach der nationalen Klassifikation und der IPK

B. RECHERCHIERTE GEBIETE

Recherchierter Mindestprüfstoff (Klassifikationssystem und Klassifikationssymbole)
IPK 6 G06F

Recherchierte aber nicht zum Mindestprüfstoff gehörende Veröffentlichungen, soweit diese unter die recherchierten Gebiete fallen

Während der internationalen Recherche konsultierte elektronische Datenbank (Name der Datenbank und evtl. verwendete Suchbegriffe)

C. ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie*	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
X	COMPUTER, Bd.26, Nr.5, Mai 1993, LONG BEACH US Seiten 52 - 61 JAMES R. LARUS 'Efficient Program Tracing' siehe Seite 58, rechte Spalte, Zeile 21 - Zeile 25	1,3-7
Y	---	2
Y	THE SECOND CONFERENCE ON ARTIFICIAL INTELLIGENCE APPLICATIONS, 13. Dezember 1985, MIAMI BEACH, FLORIDA Seiten 422 - 427 MASAYUKI NUMAO ET AL. 'Visual debugger for Prolog' siehe Seite 423, linke Spalte, Zeile 16 - Zeile 32; Abbildung 2	2

	-/--	

[X] Weitere Veröffentlichungen sind der Fortsetzung von Feld C zu entnehmen

[X] Siehe Anhang Patentfamilie

* Besondere Kategorien von angegebenen Veröffentlichungen :
'A' Veröffentlichung, die den allgemeinen Stand der Technik definiert, aber nicht als besonders bedeutsam anzusehen ist
'E' älteres Dokument, das jedoch erst am oder nach dem internationalen Anmeldedatum veröffentlicht worden ist
'L' Veröffentlichung, die geeignet ist, einen Prioritätsanspruch zweifelhaft erscheinen zu lassen, oder durch die das Veröffentlichungsdatum einer anderen im Recherchenbericht genannten Veröffentlichung belegt werden soll oder die aus einem anderen besonderen Grund angegeben ist (wie ausgeführt)
'O' Veröffentlichung, die sich auf eine mündliche Offenbarung, eine Benutzung, eine Ausstellung oder andere Maßnahmen bezieht
'P' Veröffentlichung, die vor dem internationalen Anmeldedatum, aber nach dem beanspruchten Prioritätsdatum veröffentlicht worden ist
'T' Spätere Veröffentlichung, die nach dem internationalen Anmeldedatum oder dem Prioritätsdatum veröffentlicht worden ist und mit der Anmeldung nicht kollidiert, sondern nur zum Verständnis des der Erfindung zugrundeliegenden Prinzips oder der ihr zugrundeliegenden Theorie angegeben ist
'X' Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann allein aufgrund dieser Veröffentlichung nicht als neu oder auf erfinderischer Tätigkeit beruhend betrachtet werden
'Y' Veröffentlichung von besonderer Bedeutung; die beanspruchte Erfindung kann nicht als auf erfinderischer Tätigkeit beruhend betrachtet werden, wenn die Veröffentlichung mit einer oder mehreren anderen Veröffentlichungen dieser Kategorie in Verbindung gebracht wird und diese Verbindung für einen Fachmann naheliegend ist
'&' Veröffentlichung, die Mitglied derselben Patentfamilie ist

1

Datum des Abschlusses der internationalen Recherche	Absendedatum des internationalen Recherchenberichts
23. September 1994	13. 10. 94
Name und Postanschrift der Internationale Recherchenbehörde Europäisches Patentamt, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	**Bevollmächtigter Bediensteter** Corremans, G

Formblatt PCT/ISA/210 (Blatt 2) (Juli 1992)

C.(Fortsetzung) ALS WESENTLICH ANGESEHENE UNTERLAGEN

Kategorie°	Bezeichnung der Veröffentlichung, soweit erforderlich unter Angabe der in Betracht kommenden Teile	Betr. Anspruch Nr.
A	PROCEEDINGS OF THE 15TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 21. Mai 1993, BALTIMORE, MARYLAND Seiten 456 - 465 SOUGATA MUKHERJEA ET AL. 'Applying Algorithm Animation Techniques for Program Tracing, Debugging, and Understanding' siehe Seite 459, rechte Spalte, Zeile 38 - Zeile 45 ---	2
A	EP,A,0 213 486 (HITACHI, LTD.) 11. März 1987 siehe Spalte 7, Zeile 11 - Zeile 19; Abbildung 11 -----	5-7

INTERNATIONALER RECHERCHENBERICHT

Angaben zu Veröffentlichungen, die zur selben Patentfamilie gehören

Internationales Aktenzeichen
PCT/DE 94/00781

Im Recherchenbericht angeführtes Patentdokument	Datum der Veröffentlichung	Mitglied(er) der Patentfamilie	Datum der Veröffentlichung
EP-A-0213486	11-03-87	JP-A- 62057033	12-03-87
		US-A- 4794529	27-12-88