(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 September 2006 (14.09.2006)

PCT

(10) International Publication Number
WO 2006/095358 A1

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/IN2005/000080

(22) International Filing Date: 11 March 2005 (11.03.2005)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY L.P.** [—/US]; 20555 S.H. 249, Houston, Texas 77070 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **BHAT, Anand, Gajanan** [IN/IN]; 603, Pragati Nagar, Sirsi 5811402 (IN). **MAKKAR, Manish** [IN/IN]; 96 Harjinder Nagar, 11, Lal Bangla, Kanpur 208007 (IN).

(74) Agent: **NAMA, Prakash**; Global IP Services PLLC, c/o Intellevate Private Limited, Second Floor, A-20, Sector 2, Noida 201 301, Uttar Pradesh (IN).

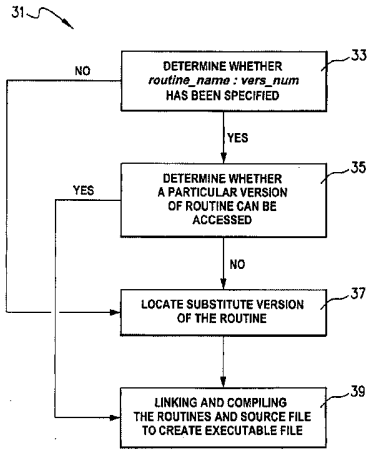
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SUBSTITUTING SOFTWARE ROUTINES FROM ALTERNATIVE LIBRARIES



(57) Abstract: A software development application determines whether routine name and version number parameters have been identified (33). If so, the application determines whether the specified version of the routine can be accessed (35). If the specified version cannot be accessed, the application locates a substitute version of the routine (37) and compiles and links the routines and a source file to produce an executable file (39).

WO 2006/095358 A1

METHODS, DEVICES AND SOFTWARE APPLICATIONS FOR FACILITATING A DEVELOPMENT OF A COMPUTER PROGRAM

Field of the Invention

[0001] The present invention relates generally to methods, devices and software applications for facilitating a development of a computer program, and has particular - but by no means exclusive - application to providing source level compatibility for the computer program.

Background of the Invention

[0002] Software compatibility is an important aspect of the software industry. Software compatibility essentially relates to the ability of software to be used in different environments such as those provided under dissimilar operating systems. Software compatibility can be broadly divided into two categories. The first category deals with source level compatibility, which relates to allowing source code to be used in different environments. The second category deals with binary compatibility, which relates to a compiled (binary) program's ability to be executed in different environments.

[0003] Whilst there exists various techniques for providing source level compatibility, those techniques have significant shortcomings. For example, object versioning used with the GNU C library (glibc) involves creating a new version of the glibc library whenever an object (routine) in the library is updated such that the updated object is incompatible with early versions of the object. A new version of the glibc library creating using object versioning is such that it contains sources (code) for all versions of objects contained in the library. As a result, the glibc library can become bulky because it contains all versions of the objects.

[0004] Another problem closely associated with object versioning in glibc is that dynamic linkers based on object versioning tend to fail the application linking process if the available glibc library does not contain the specific version of an object that the linker requires.

[0005] A further example of an existing technique used to provide source level compatibility is function level versioning in HP-UX libraries. Function level versioning is similar to objecting versioning in the glibc library in that it involves creating a new library that contains all versions of a particular object. As mentioned previously, creating a new library that contains all versions of an object has the potential to produce bulky libraries.

Summary of the Invention

[0006] In an embodiment of a method of facilitating a development of a computer program, the embodiment includes the step of determining whether there exists a first version of a routine that is to be incorporated into the computer program. Upon determining that the first version of the routine does not exist, the embodiment of the method proceeds to perform the step of locating a second version of the routine as a substitute for the first version to thereby facilitate the development.

[0007] The present invention will be more fully understood from the following description of a specific embodiment. The description is provided with reference to the accompanying figs.

Brief Description of the Drawings

[0008] Fig. 1 illustrates a personal computer embodying the present invention;

[0009] Fig. 2 is a flow chart of various steps performed by the personal computer shown in fig. 1;

[0010] Fig. 3 is a flow chart of various steps performed by the personal computer shown in fig. 1; and

[0011] Fig. 4 illustrates a data structure used by the personal computer of fig. 1.

Detailed Description

[0012] Fig 1. shows an embodiment of the present invention in the form of a personal computer 11. The personal computer is made up of numerous components that cooperate with each other. The components include: a power supply; motherboard; random access memory; a video card; a monitor; keyboard; and a hard disk loaded with the Linux operating system and a software development application. In alternative embodiments of the present invention, the hard disk of the personal computer 11 is loaded with other operating systems such as, for example, Microsoft XP, SunOS and MacOS.

[0013] The software development application can be executed by a software developer to essentially perform two tasks that facilitate the development of a computer program. The first of the tasks relates to creating a software library that includes routines that can be used to develop a computer program. The second of the tasks relates to locating a library routine that is to be incorporated into a computer program.

[0014] In relation to the first task of creating the software library, the software development application can be invoked by typing at a command line prompt: **cc -c** *file_name*, where **cc** is the file name of the software

development application, **-c** is an option that informs the software development application to create a linkable object file (that is, to create the software library); and *file_name* is the file name of the source file that is to be used to create the linkable object file.

[0015] Once the software development application has been invoked as specified in the preceding paragraph, the personal computer 11 executes the various instructions included in the software development application. Execution of the instructions causes the personal computer 11 to carry out the steps shown in the flow chart 21 of fig. 2. With reference to fig. 2, the first step 23 that the software development application performs is to obtain the source file that is identified by the *file_name* parameter. The source file contains at least one routine that can be incorporated into a computer program. As persons skilled in the art will readily appreciate, a software developer can create the source file using an application such as a text editor or perhaps by using a more advanced application such a software development tool. The source file is typically written in a high level language such as C or C++ so that it can be easily interpreted by software developers.

[0016] Subsequent to carrying out the first step 23, the software development application proceeds to carry out the second step 25 of compiling the source file (that was obtained during the first step 23) to create the software library, which is in the form of linkable object code. The second step 25 of compiling the source file is such that the software library contains only a single version of any of the routines in the source file. The second step 25 of compiling the source code is such that it is capable of identifying the different versions of the routine in the source file, and placing the latest version into the library created during the compilation step 25. Earlier versions of the routine reside in previous versions of the

software library. A consequence of the second step 25 is that potentially there will be multiple versions of the software library, each of which contains a single version of the routine. An advantage of this is that unlike existing techniques for providing source level compatibility (for example, object versioning in glibc), the software library created by the second step 25 does not contain multiple version of a routine and therefore the size of the library is minimised in comparison to existing source level compatibility techniques. Whilst a software library created by the second step 25 contains a single version of a routine, this does not preclude the library from containing multiple routines, each of which performs a different function. For example, a particular library may have functions *abc*() and *def*(), but the library will only contain a single version of *abc*() and *def*().

[0017] To ensure that each version of a software library that the previous step 25 creates, the software development application carries out the third step 27 of assigning the software library a version number. In particular, the version number is appended to a file name of the library. For example, if the file name of the library is *xyz.lib* the third step 27 is such that the library would have the file name *xyz.lib.10* where the library is the tenth version. There would also be other versions of the library that have the file names *xyz.lib.1*, *xyz.lib.2*, *xyz.lib.3*...*xyz.lib.9*.

[0018] In addition to the step 27 of assigning the software library the version number, the software development application also includes the fourth step 29 of assigning the version number to the routine in the software library. The version number assigned to the routine corresponds to the version number assigned to the library during the third step 27. For instance, the routines in version 10 of a library (for example, *xyz.lib.10*) would be assigned the version number 10. The forth step 29 is such

that it assigns the version number to the routine by assigning the version number to an entry in the software library's export table. The entry to which the version number is assigned is the entry that represents the routine. As persons skilled in the art will readily appreciate, the export table is essentially information contained in the library that enables a compiler to determine details of the various routines that are in the library.

[0019] Once the software library has been created it can then be used by a software developer to develop a computer program (application). As persons skilled in the art will readily appreciate the process of developing a computer program initially involves developing a source code file. The source code file can be developed by using a text editor or a more advanced software development tool. The source code file is usually written in a high level language such as C or C++. To create a binary (executable) version of the source file the software development application can be invoked to compile the source file, which includes linking the source code to the software library in the event that the source file uses one or more of the routines contained in the library.

[0020] To compile the source code file the software development application can be invoked by typing at a command line prompt: **cc**, *routine_name1:vers_num1 routine_name2:vers_num2 file_name* **-o** *output_name* **-l***lib_name*, where *routine_name1:vers_num1* is an optional parameter that identifies the name (*routine_name1*) of a first routine in the software library and the version number (*vers_num2*) of the first routine, *routine_name2:vers_num2* is a further optional parameter that identifies the name *(routine_name2)* of a second routine in the software library and the version number *(vers_num2)* of the second routine, *file_name* is the file

name of the source code file to be compiled, **-o** is the name of the binary (compiled) version of the source code file, and **-l***lib_name* is the file name of the software library. It is noted that whilst the previous example of the command line contains two optional parameters that are used to identify routines in a library (*routine_name1:vers_*num1 and *routine _name2:vers_num2*), it will be readily appreciated that the command line is not restricted two parameters. For example, the command line might contain the names and version numbers of five routines in the software library. A specific example of how the software development application might be invoked is as follows: **cc**, abc:4 main.c -o main -**l**xyz, which effectively causes the software development application to compile the *main.c* source file into the binary file *main* and to resolve routine *abc* from version 4 of library *xyz*.

[0021] Once the software development application has been invoked as specified in the previous paragraph, the personal computer 11 executes various instructions included in the software development application. Execution of these instructions causes the personal computer 11 to carry out the steps shown in the flow chart 31 of fig. 3. With reference to fig. 3, the first step 33 that the software development application performs is to determine whether the *routine_name:vers_num* parameters have been specified. If the *routine_name:vers_num* parameters have not been specified, the software development application proceeds to the third step 37 (which is discussed in detail in subsequent paragraphs of this specification) to obtain the latest copy of library routines required to compile the computer program into an executable file. If, on the other hand, the *routine_name:vers_num* parameters have been specified, the software development application proceeds to carry out the second step 35 of determining whether a particular version (*vers_num*) of the routine (*routine_name*) can be accessed.

[0022] To carry out the second step 35 the software development application attempts to locate a software library on the personal computer 11 that contains the particular version of the routine. More specifically, the second step 35 involves searching for a copy of the library identified in the **-l***lib_name* parameter that has been assigned the same version number as the routine, the latter of which is identified by the *vers_num* parameter. To determine whether the copy of the software library is accessible, the software development application checks whether the routine version number (*vers_num*) is contained in the file name of the software library. For example, if the *vers_num* parameter is "4" and the **-l** parameter is "xyz", then the second step 35 effectively involves determining whether library *xyz.lib.4* exists. If it is determined that the copy of the software library is accessible, the software development application precedes to carry out the final step 39, which is discussed in detail in subsequent paragraphs of this specification.

[0023] If as a result of carrying out the second step 35 the software development application is unable to locate a copy of the library (specified by the **-l** parameter) that contains the required version of the routine (*vers_num*), the software development application proceeds to carry out the third step 37 of locating a substitute version of the routine. An advantage of locating the substitute version is that unlike existing techniques for providing source level compatibility (such as that used with glibc), the software development application will not fail the linking process if it is unable to find the required version of a routine. The software development application will use the substitute version of the routine instead of the required version. It is noted, however, that use of the substitute version of the routine may cause the compiled computer program to perform in an unexpected manner.

[0024] To locate the substitute version of the routine, the third step 37 involves locating the latest version of the library identified by the **-l** parameter. This is achieved by looking for a copy of the library that has the highest version number in its file name. This assumes that version numbers are assigned in ascending order, which results in the library with the highest version number being the latest version of the library. Once the latest version of the library has been located, the third step 37 involves processing a data structure that is stored in memory and which is associated with the latest version of the library. The data structure is processed to obtain a memory address from which the routine can be retrieved. The data structure is in the form of a linked list, which provides an advantage of being able to retrieve a routine quicker than using an export table of the library. The linked list is made up of nodes, each of which points to a routine in the library. Each node also contains the name of the associated routine and the routine's version number. Fig. 4 provides a representation of the linked list.

[0025] On completing the third step 37, the software development application proceeds to carry out the final step 39 of linking the routines obtained in the preceding steps and compiling the routines with the source code file to create a binary version thereof that can be executed on a computer system.

What is claimed is:

1. A method of facilitating a development of a computer program, the method comprising the steps of:

determining whether there exists a first version of a routine that is to be incorporated into the computer program; and

upon determining that the first version of the routine does not exist, locating a second version of the routine as a substitute for the first version to thereby facilitate the development.

2. The method as claimed in claim 1, wherein the step of determining whether the first version of the routine exists comprises the step of attempting to locate a software library that comprises the first version.

3. The method as claimed in claim 2, wherein the step of attempting to locate the software library comprises the step of checking a version number that has been assigned to the software library.

4. The method as claimed in claim 3, wherein the step of checking the version number comprises the step of checking whether a file name of the software library comprises the version number.

5. The method as claimed in claim 2, wherein the software library does not comprises any other version of the routine.

6. The method as claimed in claim 1, wherein the step of locating the second version of the routine comprises the step of obtaining an address of the second version from a data structure that is stored in memory and which is associated with an alternative software library that comprises the second version.

7. The method as claimed in claim 6, wherein the data structure comprises a linked listed that has a node that identifies a name of the second version and a version number assigned thereto.

8. The method as claimed in claim 6, wherein the alternative software library does not contain any other version of the routine.

9. A method of facilitating a development of a computer program, the method comprising the step of creating a software library that has a single version of a routine that can be incorporated into the computer program to thereby facilitate the development.

10. The method as claimed in claim 9, further comprising the step of assigning the software library a version number.

11. The method as claimed in claim 10, wherein the step of assigning the software library the version number comprises the step of incorporating the version number into a file name of the software library.

12. The method as claim in claim 9, further comprising the step of assigning the version number to the routine.

13. The method as claimed in claim 12, wherein the step of assigning the version number to the routine comprises the step of incorporating the version number into an entry in an export table that is associated with the software library, the entry representing a name of the routine.

14. A computing device comprising a data storage means that has a software application comprising at least

one instruction for causing the computing device to carry out the method as claimed in claim 1.

15. A computing device comprising a data storage means that has a software application comprising at least one instruction for causing the computing device to carry out the method as claimed in claim 9.

16. A software application comprising at least one instruction for causing a computing device to carry out the method as claimed in claim 1.

17. A software application comprising at least one instruction for causing a computing device to carry out the method as claimed in claim 9.

18. A computer program that has been developed using the method as claimed in claim 1.

19. A software library that has been creating using the method as claimed in claim 9.

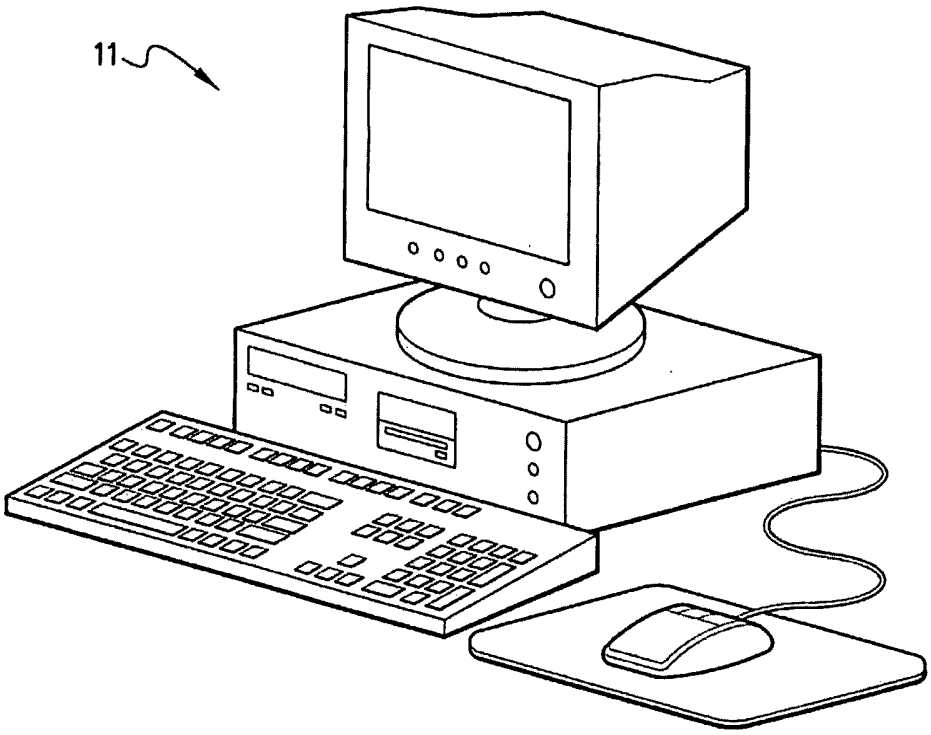


Fig. 1

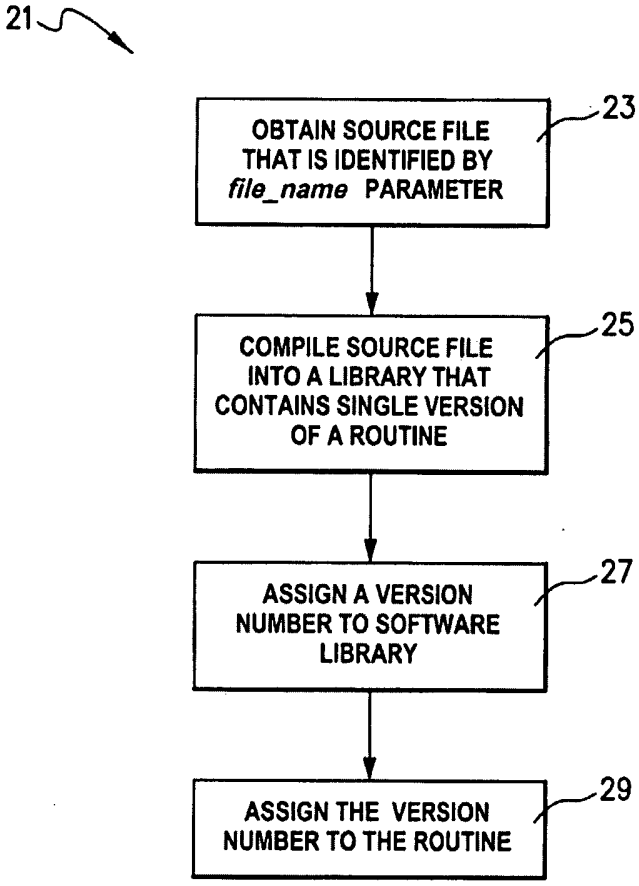

21
OBTAIN SOURCE FILE THAT IS IDENTIFIED BY *file_name* PARAMETER
23
COMPILE SOURCE FILE INTO A LIBRARY THAT CONTAINS SINGLE VERSION OF A ROUTINE
25
ASSIGN A VERSION NUMBER TO SOFTWARE LIBRARY
27
ASSIGN THE VERSION NUMBER TO THE ROUTINE
29

Fig. 2

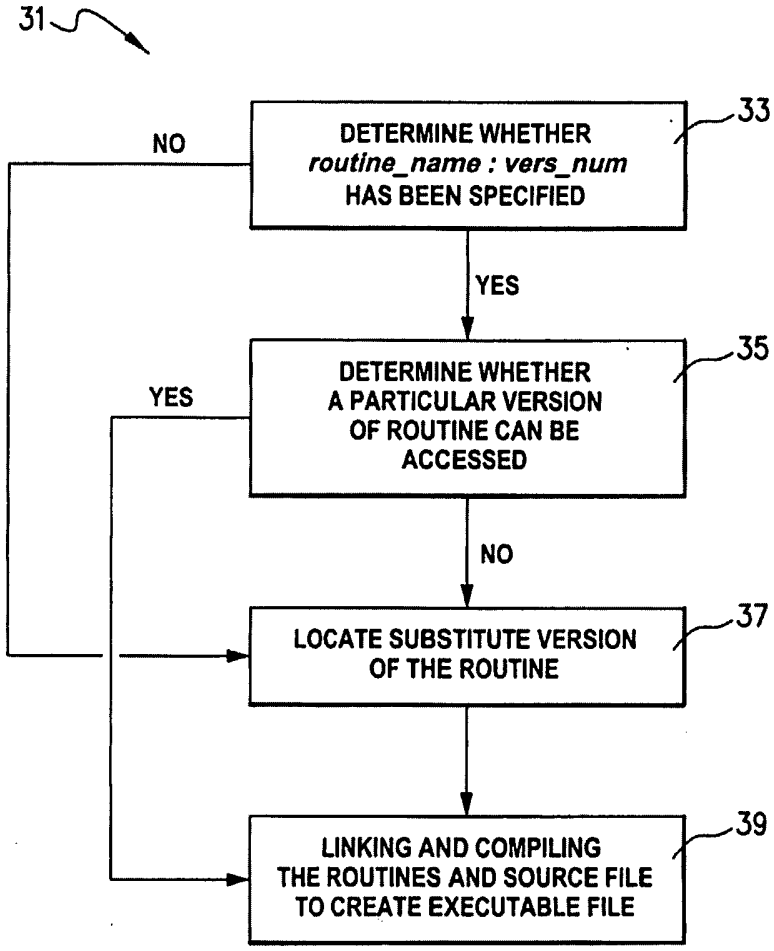

31
DETERMINE WHETHER
routine_name : vers_num
HAS BEEN SPECIFIED
33
NO
YES
DETERMINE WHETHER
A PARTICULAR VERSION
OF ROUTINE CAN BE
ACCESSED
35
YES
NO
LOCATE SUBSTITUTE VERSION
OF THE ROUTINE
37
LINKING AND COMPILING
THE ROUTINES AND SOURCE FILE
TO CREATE EXECUTABLE FILE
39

Fig. 3

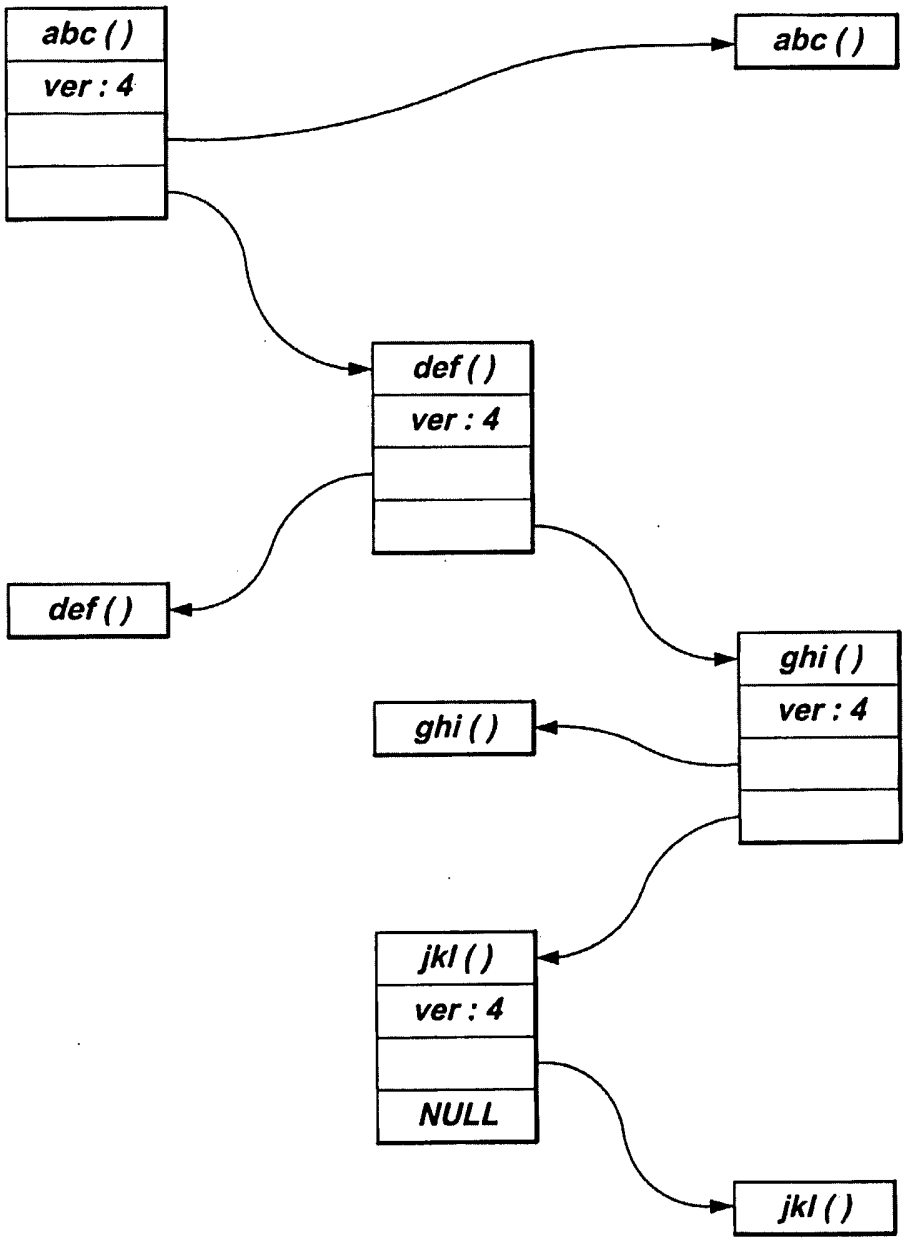


Fig. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IN05/00080

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 9/44
US CL : 717/163
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)
U.S. : 717/163

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 practicable, search terms used)
EAST(USPAT, US-PGPUB, EPO, JPO, DERWENT, IBM_TDB)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,805,899 (EVANS et al) 8 September 1998 (08.09.1998), columns 4-16.	1-19
A	US 6,658,659 B2 (HILLER et al) 2 December 2003 (02.12.2003), columns 1-16.	1-19
A	US 2001/0039650 A1 (BODO) 8 November 2001 (08.11.2001), pp. 1-3.	1-19
A	US 2002/0133804 A1 (SHEEDY) 19 September 2002 (19.09.2002), pp. 1-4.	1-19

☐ Further documents are listed in the continuation of Box C. ☐ See patent family 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 application or patent 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
14 September 2005 (14.09.2005)

Date of mailing of the international search report
30 SEP 2005

Name and mailing address of the ISA/US
Mail Stop PCT, Attn: ISA/US
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
Facsimile No. (571) 273-8300

Authorized officer
Tuan Q. Dam
Telephone No. 571-272-2100

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IN05/00080

Continuation of Item 4 of the first sheet:
The title is too long.
NEW TITLE:
"SUBSTITUTING SOFTWARE ROUTINES FROM ALTERNATIVE LIBRARIES"