PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/45, 11/34, 11/00 — A1

(11) International Publication Number: **WO 97/19403**

(43) International Publication Date: 29 May 1997 (29.05.97)

(21) International Application Number: PCT/US96/18201

(22) International Filing Date: 15 November 1996 (15.11.96)

(30) Priority Data:
08/560,426 17 November 1995 (17.11.95) US

(71) Applicant: MCI COMMUNICATIONS CORPORATION [US/US]; 1133 19th Street, N.W., Washington, DC 20036 (US).

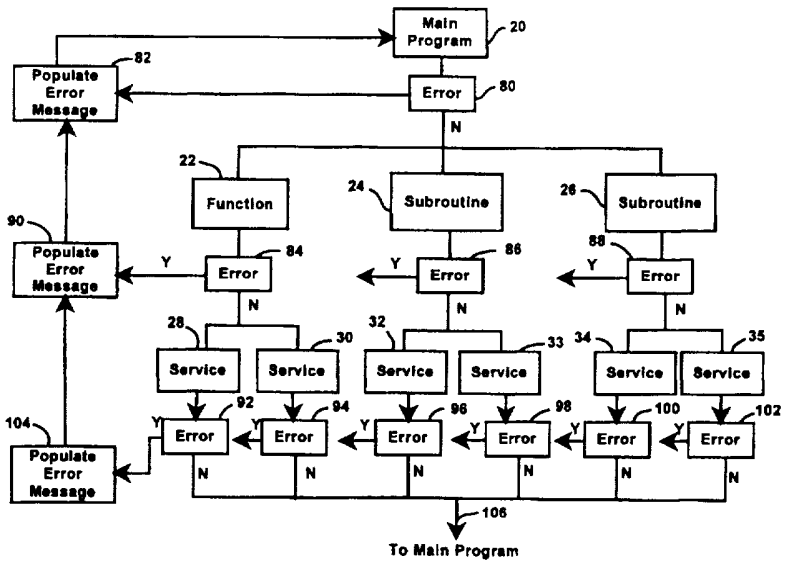
(72) Inventor: MCQUEEN, Stan; 2145 Sather Drive, Colorado Springs, CO 80915 (US).

(74) Agents: LISS, Morris et al.; Pollock, Vande Sande & Priddy, P.O. Box 19088, Washington, DC 20036 (US).

(81) Designated States: CA, JP, MX, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: HIERARCHICAL ERROR REPORTING SYSTEM



(57) Abstract

Error reporting may be enhanced by utilizing a programming language, such as C++ and its attendant enhanced error reporting facility. The invention generates an error message at the function level where the error occurs, as well as noting the line of source code during which time the error occurred. The resulting populated error message (82) is then rolled up toward the main program (20). At each preceding roll up level, an additional error message (90) is populated which notes the original error message (104) information as well as adding return path information. Thus, after completed roll up to the main program (20), stacked error messages (76) are made available to the user (10) that fully define the complete path involving the error (80), in addition to the identification of the source code line where the error occurred.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

HIERARCHICAL ERROR REPORTING SYSTEM

Field of the Invention

The present invention relates to computer error reporting systems, and more particularly to such a system that tracks and reports the software path traversed by an error generated during the running of a software application.

Background of the Invention

Error code generation is an important aspect of running computer software (hereinafter referred to as "applications"). Even operating systems generate error codes or phrases when an abnormality is detected during operation of the software. Typical error messages in applications may include a message indicating that a particular requested file has not been found. Similarly, in an operating system, error messages are presented to the user, such as in the instance where a command is entered by the user that is not recognized by the operating system.

A primary disadvantage of present error reporting systems, in applications and operating systems, is the lack of fully identifying where in the program (e.g., line of source code) the error was detected. This would enable programmers to review the source code and quickly concentrate on the cause of the error, based on its occurrence at a particular point in the program. Further, the same error code may be returned during various subroutines or functions, during an application, so that no unique correspondence between the error and the function/subroutine exists. This makes it more difficult to troubleshoot problems when running applications.

The problem becomes multiplied when a main program accomplishes its processing by relying upon modules, in the nature of functions and subroutines, that themselves rely upon a library of software services. By way of example, a software function may include an application command to retrieve a file. Services relate to underlying surfaces such as OLE which provides a software transport for importing data from one particular application, such as Excel™ into a second application, such as Word™. Since the same service may be employed by various functions and

subroutines, the generation of a simple service error message is not terribly useful since it does not uniquely identify which branch of the application was involved.

Accordingly, a need exists to track and report errors with greater detail than that currently available.

Brief Description of the Present Invention

The present invention anticipates the rewrite of existing applications, or the generation of new applications utilizing programming languages that permit the generation and relaying of detailed error messages (hereinafter referred to as "exceptions") that identifies a line of source code where the error is generated, as well as the path traversed by the exception on a return of the resulting error message to a user's display.

This is accomplished by generating a message, upon the occurrence of an exception, the message being populated with fields that define where the error occurred. This message is then relayed through subsequent application functions as part of a return path to the main program and subsequently to the user's display. Once an error message is generated at one particular function layer, each subsequent functional layer affected generates its own message identifying the location of that subsequent function layer. The end result will be a stacking of these error messages, relating to an exception detected along the return path to the main program.

In this manner, the return path is completely identified since path data has populated fields of respective error messages, created at each affected functional layer. Part of the information includes the line of source code affected when the error was detected.

This level of detail in error reporting is stored for troubleshooting. It is most helpful to detect whether the error occurred as a result of a "bug" in the program itself, or whether it was most probably introduced as a result of user error.

Certainly, the generation of hierarchical error reporting is extremely useful during development of a software application.

Brief Description of the Figures

The above-mentioned objects and advantages of the present invention will be more clearly understood when considered in conjunction with the accompanying drawings, in which:

Fig. 1 is an architectural block diagram of a generalized computer system platform.

Fig. 2 is a hierarchical flowchart of a program indicating the two principal layers of application and underlying software layer, namely, services.

Fig. 3 is a basic block diagram of a client–server environment involving database management.

Fig. 4 is a flowchart indicating a typical prior art exception (error) handling process between functions of an application.

Fig. 5 is a flowchart, similar to Fig. 4, but indicating a hierarchical exception handling, as employed in the present invention.

Fig. 6 is a layout of an exception message, indicating typical fields to be populated with detailed error message information, as employed in the present invention.

Fig. 7 is flowchart, similar to that of Fig. 2 but indicating additional hierarchical error reporting, as contributed by the present invention.

Fig. 8 is a flowchart of a sequence of normal operation of a program written in a language such as C++.

Fig. 9 is a flowchart similar to Fig. 8, with operation of error handling depicted.

Detailed Description Of The Invention

Fig. 1 is a generalized architecture of a personal computer (PC). A user 10 relies upon a software program or application 12 to process data (for example, in a word processing program such as Word™ by Microscoft). The PC system is controlled by an operating system 16, such as DOS™, Windows™, etc. The hardware of the computer 18 carries out the various processes required by the application and the operating system. Often times, an additional layer, indicated as services 14, interfaces with the application. For example, if data from the computer software Excel™, by Microscoft, is to be inserted into a document being prepared in Word™, an underlying service, such as the conventional service OLE may be required. Any of the boxes shown in Fig. 1 may generate an error. The purpose of the present invention is to report back, with detailed particularity to the user 10, an error report that identifies the path of the error as well as the line of source code where the error was encountered. Basically, in order to achieve this function, and as will be explained hereinafter in greater detail, this requires the generation of an error message in the layer where the error is introduced. Subsequently "touched" levels in the return path of the error, to the user, generate a separate error message identifying each respective layer in the return path. The end result is a stacking of error messages that completely define the error and the return path of the error. The error messages created avoid the ambiguity that results from present error reporting systems where a similar error can occur in parallel branches so that a returned error does not fully identify the particular branch affected.

For example, in Fig. 2, the hierarchy for a prior art architecture involving a software application and attendant services is depicted. More particularly, a main program 20 controls subroutines 24 and 26. The main program also controls a function 22 which may be a simple command from the user 10, such as a retrieve file function, or similarly a read function of a database record. As the hierarchy diagram of Fig. 2 indicates, the function 22 and subroutines 24, 26 may each cooperate with various services 28–35. As a reminder, "services" is used in the context of an

underlying software layer such as the case of OLE, as previously explained in connection with Fig. 1. Function 22 again refers to an application command, such as retrieve a file or read a database record. Typically, software applications include a library of functions, any of which may be the particular function indicated by block 22 in Fig. 2.

In order to understand the limitations of the prior art reporting systems, consider more than one of the services 28–35 as generating an error message. The error message is reported back through a respective preceding level (function 22, subroutines 24, 26) to the main program 20. The user is informed as to the error, which is typically a code or short phrase. However, since multiple branches from the main program invoke the same service, it would be impossible to determine through which path the error message was returned.

The present invention solves this problem by keeping track of the path through which an error message is "rolled up" to a top level, namely the main program. The error message, as will be developed hereinafter, not only develops information concerning the path through which the function error is returned, but also indicates which line of source code was being operated upon by the main program during the generation of the error. Such a reporting system is of great advantage during operation of the system, so that it may be determined whether a user is introducing the error, or whether a software problem exists. Usefulness in detecting the latter mentioned problem is of particular importance during the debugging of applications during development.

In order to accomplish the objectives of the present invention, an application must be written (or an existing application rewritten) in a programming language that provides a tool kit sufficiently extensive to allow the types of error message generation presently required. Thus, in a preferred embodiment of the present invention, the known programming language C++ is employed.

To further explore the usefulness of the present invention, reference is made to Fig. 3 which shows a typical client-server environment for database applications 40, 42 and 44. Typically, a user 36 monitors a display, which may be a graphical user interface (GUI) 38 (e.g., Windows™). This represents a data presentation layer for the user. The client is provided with a local database 46 which is constantly refreshed during system operation. Communication between the client and server occurs at a communication interface including a messaging layer 48 at the client and a corresponding messaging layer 50 at the server. The main database 52 at the server is the source of the complete database. In typical applications, messaging may occur by utilizing TCP/IP protocol.

In the system of Fig. 3, the user requests records from the main database 52. During such an operation, if an error message were created, the local database 46 would return a message indicating that the record could not be found. However, there would be no information as to which aspect of the communication chain actually caused the error. The programming language C++ includes exception handling capabilities (error messages) and by virtue of the present invention, the language's exception handling is augmented, in a manner heretofore unknown, so as to add error path tracking. Thus, in the example discussed in connection with Fig. 3, the presentation layer 38 would enable the user to specifically track where in the communication chain the error occurred, and which application was involved, including source code line identification.

Fig. 8 represents the normal flow of control from function to function in a program written in C++, when there are no errors. In the Figure, Function A calls Function B. This results in a transfer of control to the beginning of Function B. At some time in its processing, Function B calls Function C, resulting in a transfer of control to the beginning of Function C. Function C calls no additional function and executes from beginning to end. When Function C has completed execution, control is transferred back to Function B, which continues executing, starting at the

instruction following the call to Function C. Function B then completes executing and returns control back to Function A. Function A resumes execution starting at the instruction following the call to Function B. Function A then completes executing and returns control to its caller (which could possibly be the operating system if Function A is the "main" or topmost function in the application). Even though each function has "exception handling code", the code is never invoked since no error occurred.

Fig. 9 represents the same application as the previous Fig., except that, in this case, Function C detects an error during its processing. When Function C "throws" an exception, control transfers to Function B. In this case, however, instead of execution resuming in Function B at the instruction following the call to Function C, execution now transfers to the exception handling code that is part of Function B. If this code passes the exception back up the chain ("rethrows" the exception), then control transfers to the exception handling code that is part of Function A.

Thus it is seen that, typically, function execution does not occur sequentially in the sense that Function A completes processing followed by Function B, followed by Function C. Instead, Function A interrupts itself to call Function B, which in turn interrupts itself to call Function C. When the lowest called function is complete, the next lowest function is then re-entered and allowed to complete. And so it goes, up and down the chain of function calls.

A further exploration of normal exception handling by C++ is indicated in Fig. 4, which does not include the augmentation of the present invention. In Fig. 4, a portion of an application is indicated, employing C++, wherein four sequential Functions A-D (56-62) are shown. For example, if an exception 64 was "thrown" (reported) by Function D, that exception is "caught" (received) by the previous layer C. As indicated in Fig. 4, the exception is "rolled up" through prior Functions (A-C) (66, 68), thereby completing an error reporting process. However, although

the nature of the error may be identified, its particular path from Function D-A will not be derived.

Fig. 5 indicates the stacking of exceptions so that the entire return path of an exception may be recorded. In accordance with the present invention, if Function D throws Exception 1 (at reference numeral 70), a message will be generated indicating that Function D is found to be in error and what line of source code was effected. This message corresponds to Exception 1 (reference numeral 70). The thrown Exception 1 is caught by Function C which generates its own message acknowledging the previous message. Thus, Exception 2, thrown by Function C (72), repeats the information that Function D was found to be in error and that Function D was called from Function C. Further, the line of source code, where the error occurred, is repeated.

Reference numeral 74 in Fig. 5 indicates the continued rolling up of Exceptions 1 and 2 to the preceding Function B. Again, the error messages are stacked, with the Function B adding its own error message to repeat the previous information and adding the information that Function B called Function C.

Fig. 6 indicates the stacking of two exception (error) messages 76 and 78. Block 76 indicates the typical types of fields that would be included in an exception generated by C++. These would typically include a text message, corresponding to the error. The file name indicates the original source file containing the function generated error. The function or function module name is next included. The line number in the original source code is similarly listed. If an error code is generated from the application itself, for example error codes in a word processing or database program, the error code would be included, as well. A field is also available for explanatory information relative to the error (variable string information). Another field may include an alternate error code, which, for example, may be a programmer's own error code.

In operation of the present invention, Fig. 7 indicates a software flow diagram with enhanced error reporting, as compared with the prior art of Fig. 2.

Thus, as shown in Fig. 7, if an error is detected at step 80, between the main program 20 and the function/subroutine steps 22/24, 26 an exception is generated and an error message 82 is populated, with the information indicated in Fig. 6.

If no error occurs, the application continues with the appropriate function or subroutine. If an error is detected during function 22, the error step 84 will populate an error message 90, and this is rolled up toward the main program by stacking a second error message 82 for completing the roll up path information. Similarly, if errors occur during subroutines 24, 26, corresponding error steps 86 and 88 will populate an error message 90. In the example illustrated in Fig. 7, if no error messages occur at the conclusion of the function or subroutines, various services 28–35 will occur.

An error during the first executed service will cause completion of a corresponding error step 92–102, thereby causing the population of an error message 104 at this level. As indicated in Fig. 7, an error message 104 will be stacked with error messages 90 and 82 during roll up of the error message to the main program 20. If no error occurs, the application may be returned to the main program 106. The end result of error reporting is the stacking of error messages to complete the information concerning return path and source code line number.

Therefore, by virtue of the present invention, software applications may be written (or rewritten) in a programming language such as C++ with enhanced error reporting as compared with the prior art. Full information concerning error path and source line identification expedites error detection and correction in software that is being developed or thereafter.

It should be understood that the invention is not limited to the exact details of construction shown and described herein for obvious modifications will occur to persons skilled in the art.

Claims

We claim:

1. An error reporting system for computer applications, comprising:

means for detecting an error at a present step during the execution of a software application;

means for populating the fields of a first error message corresponding to the detected error, with comprehensive information identifying the error and the step where it occurred;

the message including a field identifying the line of source code being executed when the error occurred;

the error message being rolled up to a preceding step of the application;

means for populating a second error message with preselected path information regarding the preceding step and stacking the second error message to the first message;

means for rolling up the first and second error messages to earlier executed steps of the application and correspondingly populating respective error messages for stacking with the first and second messages; and

means for displaying information from resulting stacked messages, to a user, thereby defining the path of the error and the line of source code executed when the error occurred.

2. The system set forth in claim 1 further including client-server components, comprising:

a client local database accessed by at least one application;

first means for messaging data requests from an application, located at the client, to the server;

second messaging means, located at the server, for communicating with the first messaging means; and

a main database, located at the server, for downloading the data requested during an application via the first and second messaging means.

3. In a data processing system having a software application invoking at least one layer of software services, an error reporting apparatus comprising:

means for detecting an error at a present step during the execution of a service invoked by the software application;

means for populating the fields of a first error message corresponding to the detected error in execution of the service, with comprehensive information identifying the error and the step where it occurred;

the message including a field identifying the line of source code being executed when the error occurred;

the error message being rolled up to a preceding step of the application;

means for populating a second error message with preselected path information regarding the preceding step and stacking the second error message to the first message;

means for rolling up the first and second error messages to earlier executed steps of the application and correspondingly populating respective error messages for stacking with the first and second messages; and

means for displaying resulting stacked messages to a user thereby defining the path of the error occurring during execution of the service and the line of source code executed when the error occurred.

4. The system set forth in claim 3 further including client-server components, comprising:

a client local database accessed by at least one application;

first means for messaging data requests from an application, located at the client, to the server;

second messaging means, located at the server, for communicating with the first messaging means; and

a main database, located at the server, for downloading the data requested during an application via the first and second messaging means.

5. A method for reporting errors to a user during execution of a software application, the method including the steps of:

detecting an error at a present step during the execution of a software application;

populating the fields of a first error message corresponding to the detected error, with comprehensive information identifying the error and the step where it occurred;

the message including a field identifying the line of source code being executed when the error occurred;

the error message being rolled up to a preceding step of the application;

populating a second error message with preselected path information regarding the preceding step and stacking the second error message to the first message;

rolling up the first and second error messages to earlier executed steps of the application and correspondingly populating respective error messages for stacking with the first and second messages; and

displaying information from resulting stacked messages, to a user, thereby defining the path of the error and the line of source code executed when the error occurred.

6. The method set forth in claim 5 further including client-server communication steps comprising:

accessing a client local database, by at least one application;

messaging data requests, by the client to the server;

downloading the data requested during an application, from a server based main database, via return messaging, to the local database at the client.

7. In a data processing system having a software application invoking at least one layer of software services, an error reporting method comprising the steps:

detecting an error at a present step during the execution of a service invoked by the software application;

populating the fields of a first error message corresponding to the detected error, in execution of the service, with comprehensive information identifying the error and the step where it occurred;

the message including a field identifying the line of source code being executed when the error occurred;

the error message being rolled up to a preceding step of the application;

populating a second error message with preselected path information regarding the preceding step and stacking the second error message to the first message;

rolling up the first and second error messages to earlier executed steps of the application and correspondingly populating respective error messages for stacking with the first and second messages; and

displaying information from resulting stacked messages, to a user, thereby defining the path of the error occurring during the execution of the service and the line of source code executed when the error occurred.

8. The method set forth in claim 7 further including client-server communication steps comprising:

accessing a client local database, by at least one application;

messaging data requests, by the client to the server;

downloading the data requested during an application, from a server based main database, via return messaging, to the local database at the client.

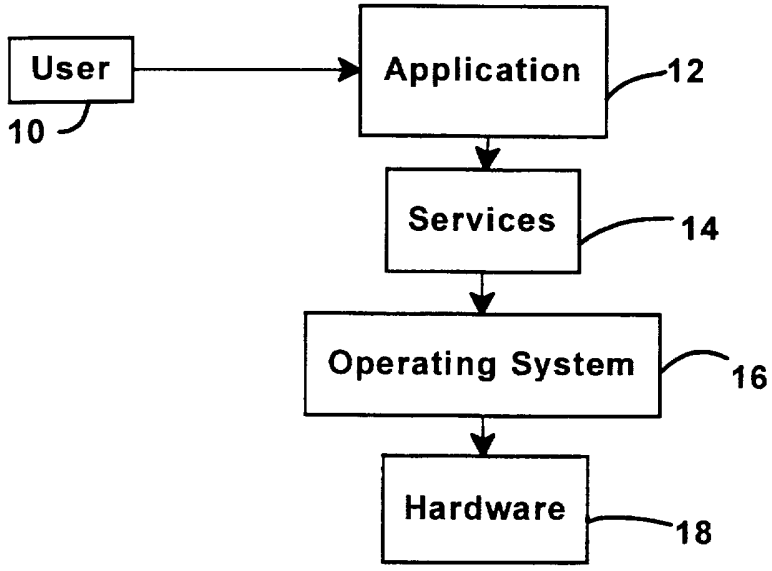


FIG. 1

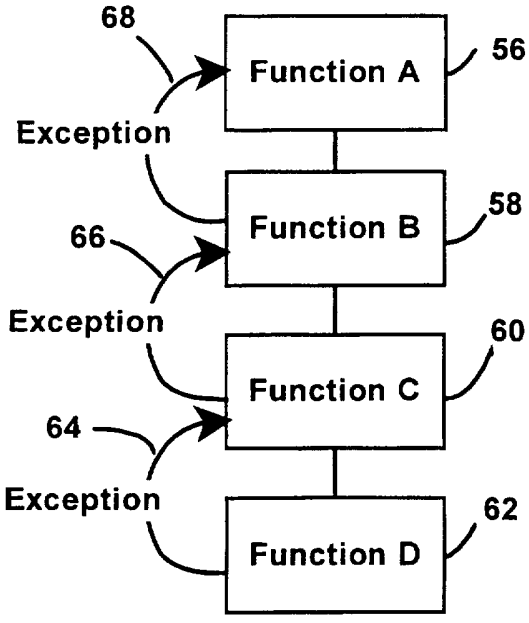


FIG. 4
PRIOR ART

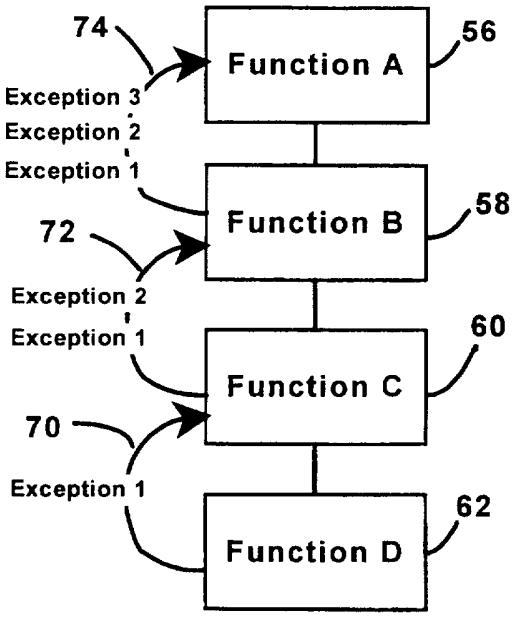


FIG. 5

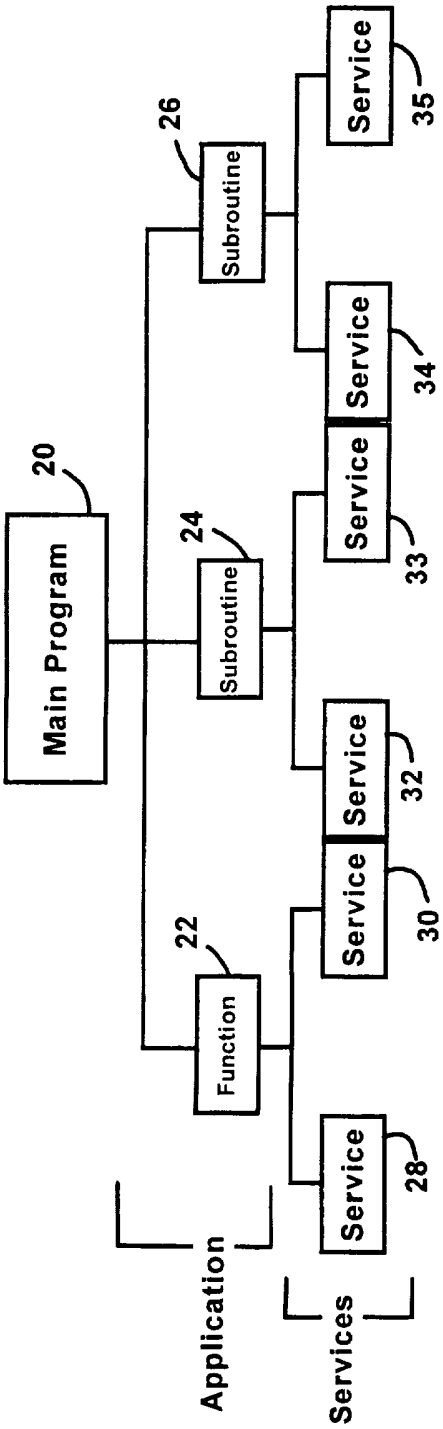


FIG. 2
PRIOR ART

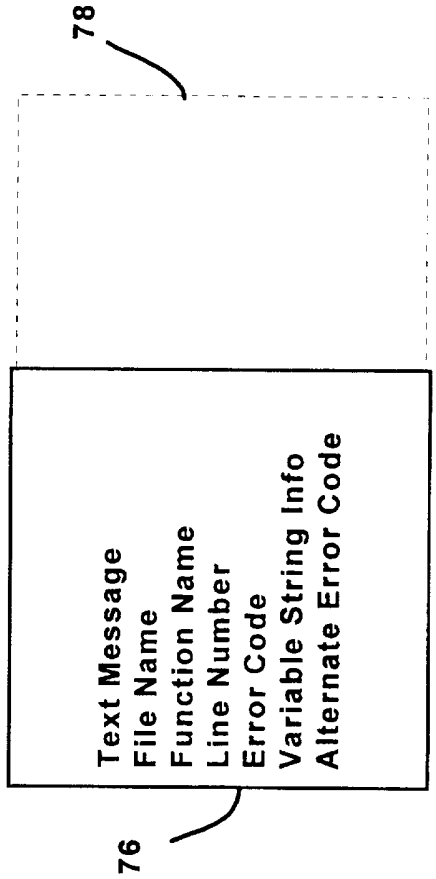


FIG. 6

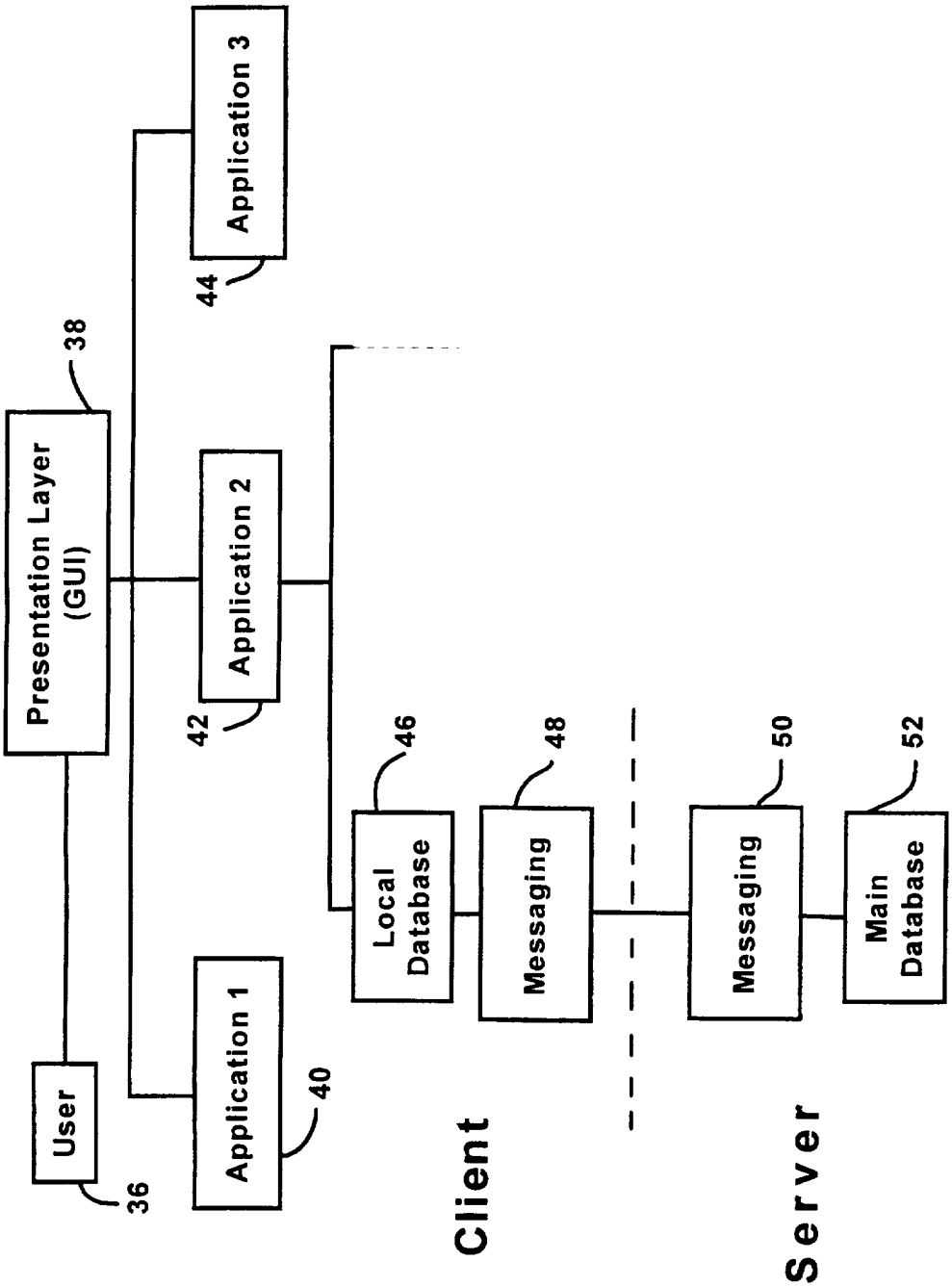

Application 3
44
38
Presentation Layer (GUI)
Application 2
42
46
48
50
52
Local Database
Messaging
Messaging
Main Database
User
36
Application 1
40
Client
Server

FIG. 3

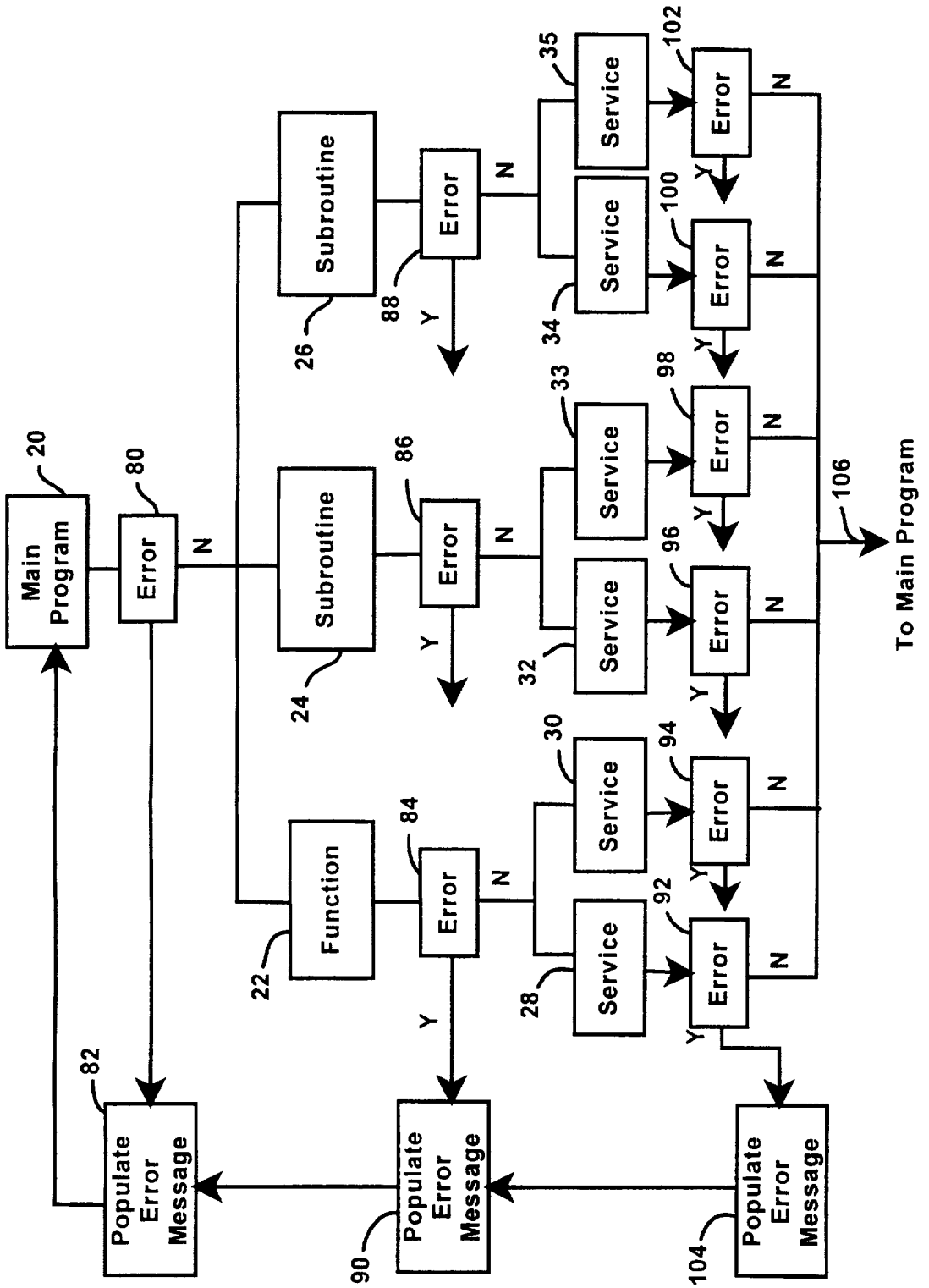

Main Program
20
Error
80
N
Y
Subroutine
26
Error
88
Subroutine
24
Error
86
Function
22
Error
84
Service
35
Service
34
Service
33
Service
32
Service
30
Service
28
Error
102
Error
100
Error
98
Error
96
Error
94
Error
92
Populate Error Message
82
Populate Error Message
90
Populate Error Message
104
106
To Main Program

FIG. 7

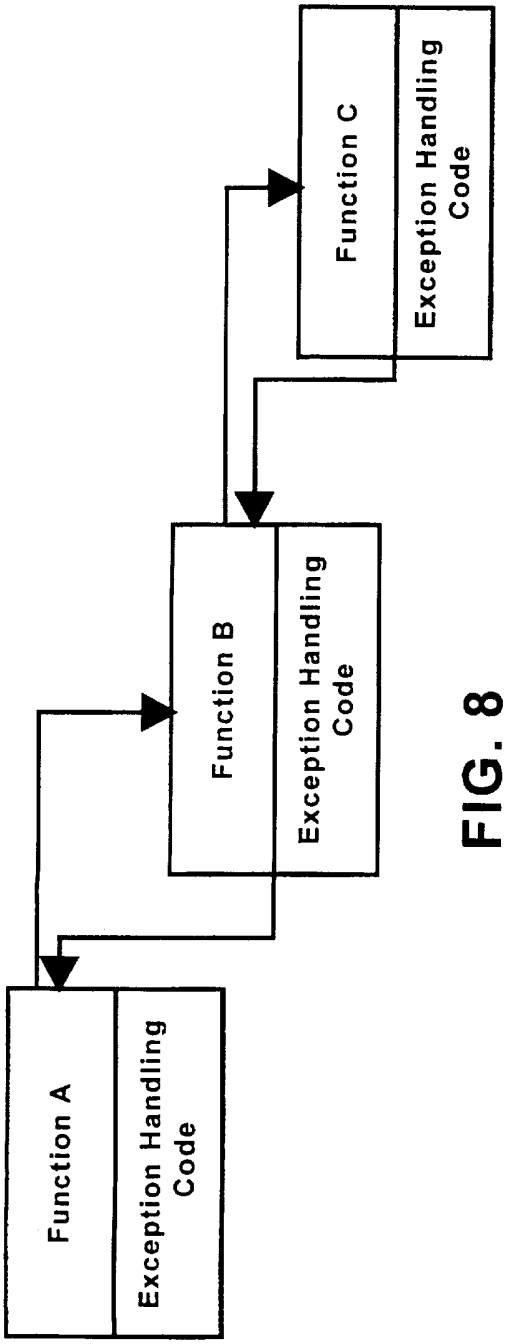

Function A
Exception Handling Code
Function B
Exception Handling Code
Function C
Exception Handling Code

FIG. 8

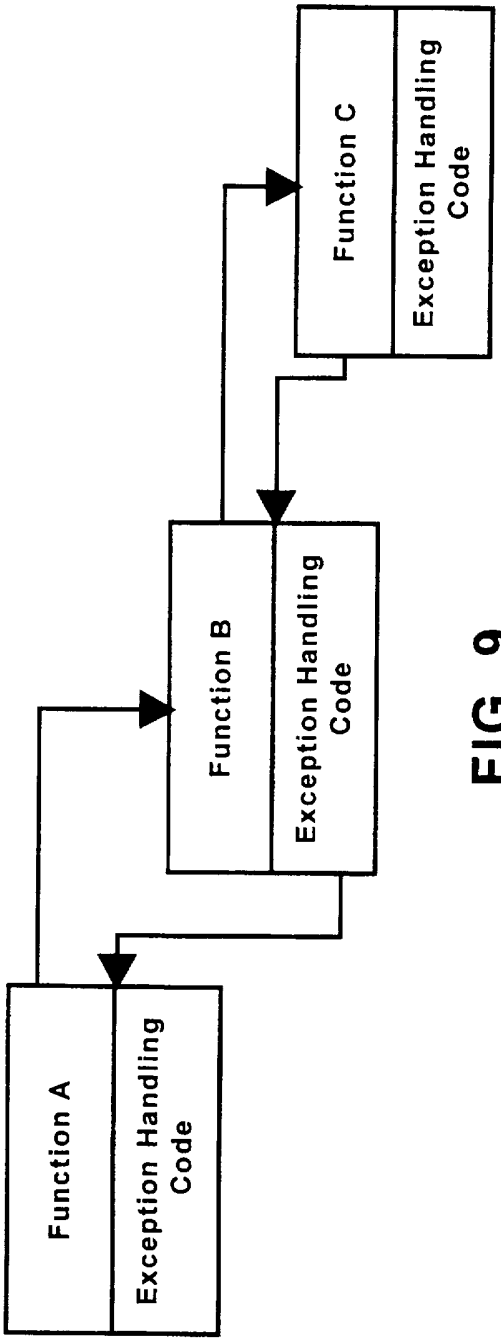

Function A
Exception Handling Code
Function B
Exception Handling Code
Function C
Exception Handling Code

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/18201

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 9/45, 11/34, 11/00

US CL :Please See Extra Sheet.

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. : 395/183.14, 183.15, 184.01, 185.01, 185.02, 601, 704, 705, 708;
364/232.3, 254.5, 267, 267.91, 275.5

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)

APS, EDS-MAYA, DIALOG, STN, EPOQUE II

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,432,795 (ROBINSON) 11 July 1995, see figures 1, 14-15; col. 3, lines 3-6; col.7, lines 12-13; col.5, lines 3-7, 15, 19-23; col. 23, lines 1-13; col. 24, lines 50-53; see col. 3, lines 1-3; col. 5, lines 1-3.	1, 3, 5, 7 2, 4, 6, 8
Y	US, A, 5,383,201 (SATTERLEE ET AL.) 17 January 1995, see figs. 1A- 1B, abstract, cols. 19-20.	1, 3, 5, 7
Y	US, A 5,450,575 (SITES) 12 September 1995, see abstract, figures 11A-11B, cols. 4-5;	1, 3, 5, 7
Y,P	US, A, 5,561,763 (ETO ET AL.) 01 October 1996, see abstract, fig. 5, col.22, lines 31-48.	2, 4, 6, 8

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"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
"A"	document defining the general state of the art which is not considered to be of particular relevance	"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
"E"	earlier document published on or after the international filing date	"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
"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)	"&"	document member of the same patent family
"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		

Date of the actual completion of the international search 21 FEBRUARY 1997	Date of mailing of the international search report 26 MAR 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer DIEU-MINH THAI LE Telephone No. (703) 305-9408

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

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/18201

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US, A, 5,047,977 (HILL ET AL.) 10 September 1991, see figures 8 and 10; col.1, lines 31-39;	1, 3, 5, 7
Y	News Release, "SQL Solutions Unveils SQR-Developer's Kit", issued 05 November 1990, page 1;	1-8
Y	New Release, "StratosWare Releases MemCheck for the Macintosh", issued 19 October 1992, pages 1-2;	1-8
Y	PC Week, volume 7, no. 47, issued 26 November 1990, pallatto, John, "SQL Solutions is broadening database line with SQR 4GL", p63 (1).	1-8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/18201

A. CLASSIFICATION OF SUBJECT MATTER:
US CL :

395/183.14, 183.15, 184.01, 185.01, 185.02, 601, 704, 705, 708;
364/232.3, 254.5, 267, 267.91, 275.5