US 20020162053A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2002/0162053 A1

Os (43) Pub. Date: Oct. 31, 2002

(54) USER TRANSPARENT SOFTWARE MALFUNCTION DETECTION AND REPORTING

(76) Inventor: Ron van Os, Sunnyvale, CA (US)

Correspondence Address:
BURNS, DOANE, SWECKER & MATHIS, L.L.P.
P.O. Box 1404
Alexandria, VA 22313-1404 (US)

(21) Appl. No.: 10/150,543

(22) Filed: May 17, 2002

Related U.S. Application Data

(63) Continuation of application No. 09/265,839, filed on Mar. 10, 1999.

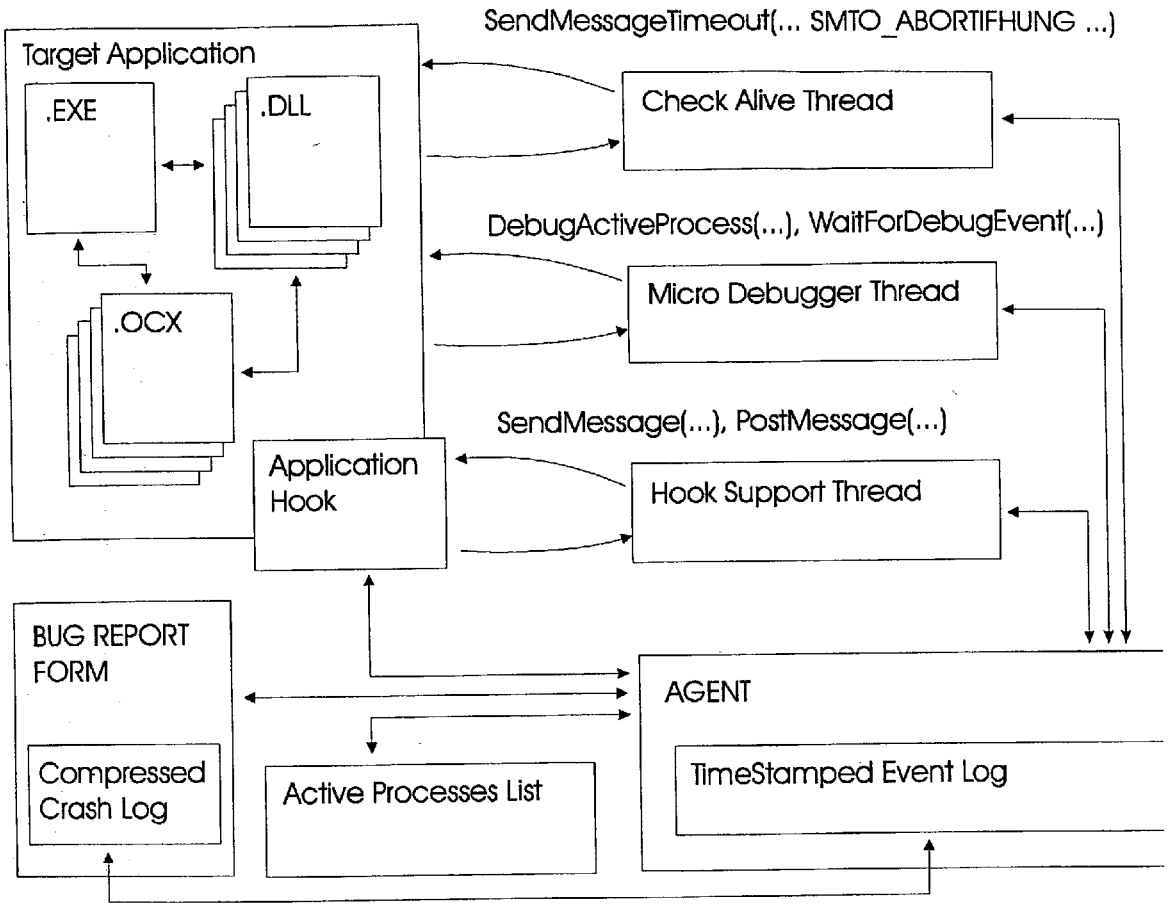
Publication Classification

(51) Int. Cl.[7] H04B 1/74

(52) U.S. Cl. 714/38

(57) ABSTRACT

The present invention, generally speaking, "instruments" an arbitrary software program, without changing the software program, to automate malfunction detection and reporting. Although users can be invited to enter a description of what the user was doing prior to receiving the error, report generation and transmission to a remote server can be fully automatic and transparent to the user. In the case of beta testing, therefore, a software developer is guaranteed to receive all pertinent information about malfunctions of an application without having to rely on "fallible humans" for this information. The effectiveness of beta testing, in terms of ultimately contributing to an improved product, is therefore greatly increased Various kinds of malfunctions may be detected and reported, including an application "crashing," becoming "hung," etc.



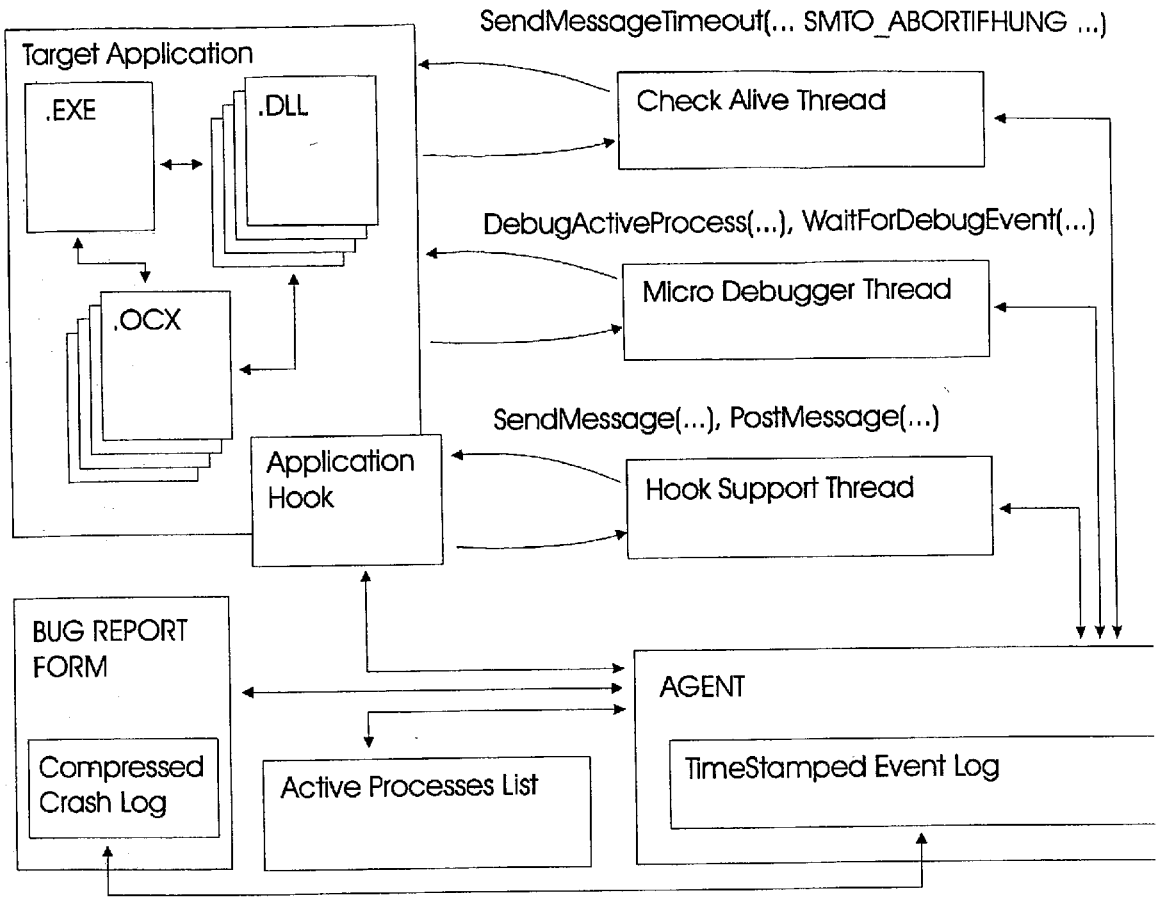

SendMessageTimeout(... SMTO_ABORTIFHUNG ...)
Target Application
.EXE
.DLL
.OCX
Check Alive Thread
DebugActiveProcess(...), WaitForDebugEvent(...)
Micro Debugger Thread
SendMessage(...), PostMessage(...)
Application Hook
Hook Support Thread
BUG REPORT FORM
Compressed Crash Log
Active Processes List
AGENT
TimeStamped Event Log

Fig. 1

```
***CASHLOG EXAMPLE *****

06:02:02-11/16/1998, CLIENT START, CLIENTID=2054
....
...
....
06:08:12-11/16/1998, WM_COMMAND, CMD=15
06:09:10-11/16/1998, LOADMODULE, C:\WinNT\OFFICE\SPELL32.DLL @ 0xBF002323
06:10:07-11/16/1998, WM_COMMAND, CMD=17
06:10:07-11/16/1998, WM_COMMAND, CMD=22
06:10:08-11/16/1998, WM_COMMAND, CMD=17
06:10:12-11/16/1998, WM_COMMAND, CMD=27
06:11:07-11/16/1998, WM_COMMAND, CMD=1024
06:11:07-11/16/1998, WM_COMMAND, CMD=29
06:14:10-11/16/1998, UNLOADMODULE, SPELL32.DLL
06:17:52-11/16/1998, WM_COMMAND, CMD=1025
06:19:47-11/16/1998, WM_COMMAND, CMD=27
06:19:48-11/16/1998, WM_COMMAND, CMD=32107
06:19:49-11/16/1998, LOADMODULE, C:\WINNT\system32\MSIDLE.DLL
06:19:49-11/16/1998, EXCEPTION, 0xC0000005: Access Violation @ 0x5F45C5B5,
MODULE: C:\WINNT\system32\MSVCRT.DLL
06:19:51-11/16/1998, CLIENT EXCEPTION LOG SAVED, CLIENTID=2054,
TRANSID=90123, AGENTID=11300
```

Fig. 2

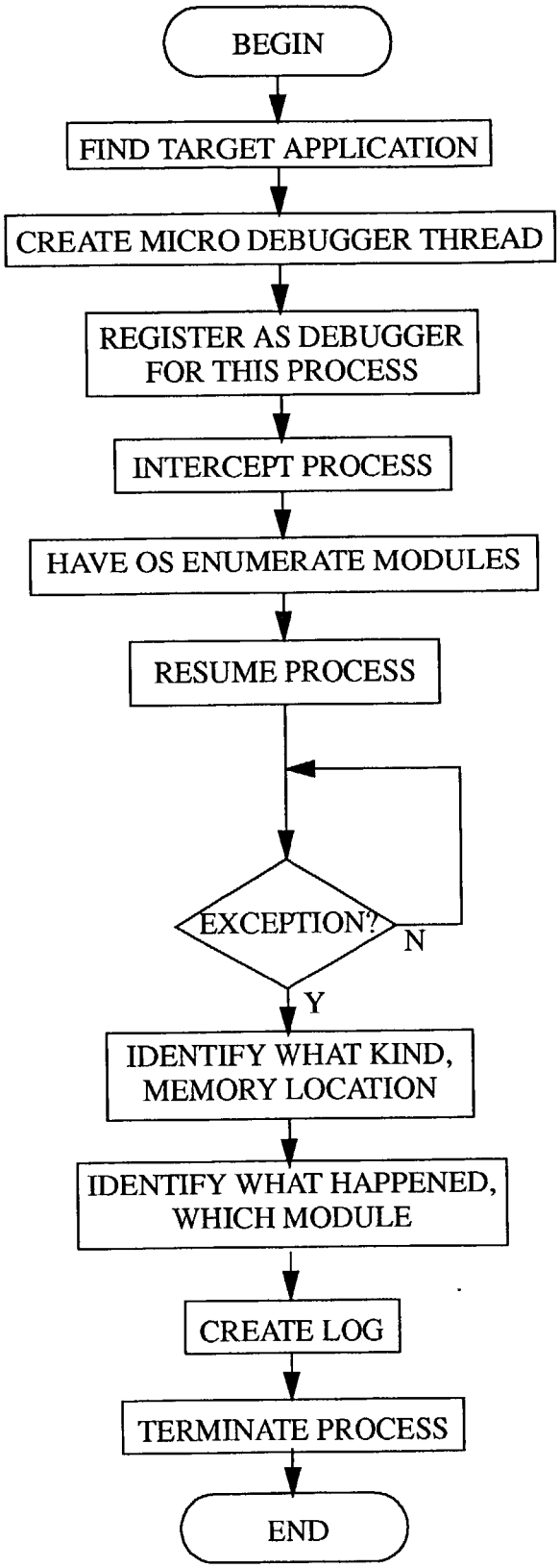

BEGIN
FIND TARGET APPLICATION
CREATE MICRO DEBUGGER THREAD
REGISTER AS DEBUGGER FOR THIS PROCESS
INTERCEPT PROCESS
HAVE OS ENUMERATE MODULES
RESUME PROCESS
EXCEPTION?
N
Y
IDENTIFY WHAT KIND, MEMORY LOCATION
IDENTIFY WHAT HAPPENED, WHICH MODULE
CREATE LOG
TERMINATE PROCESS
END

Fig. 3

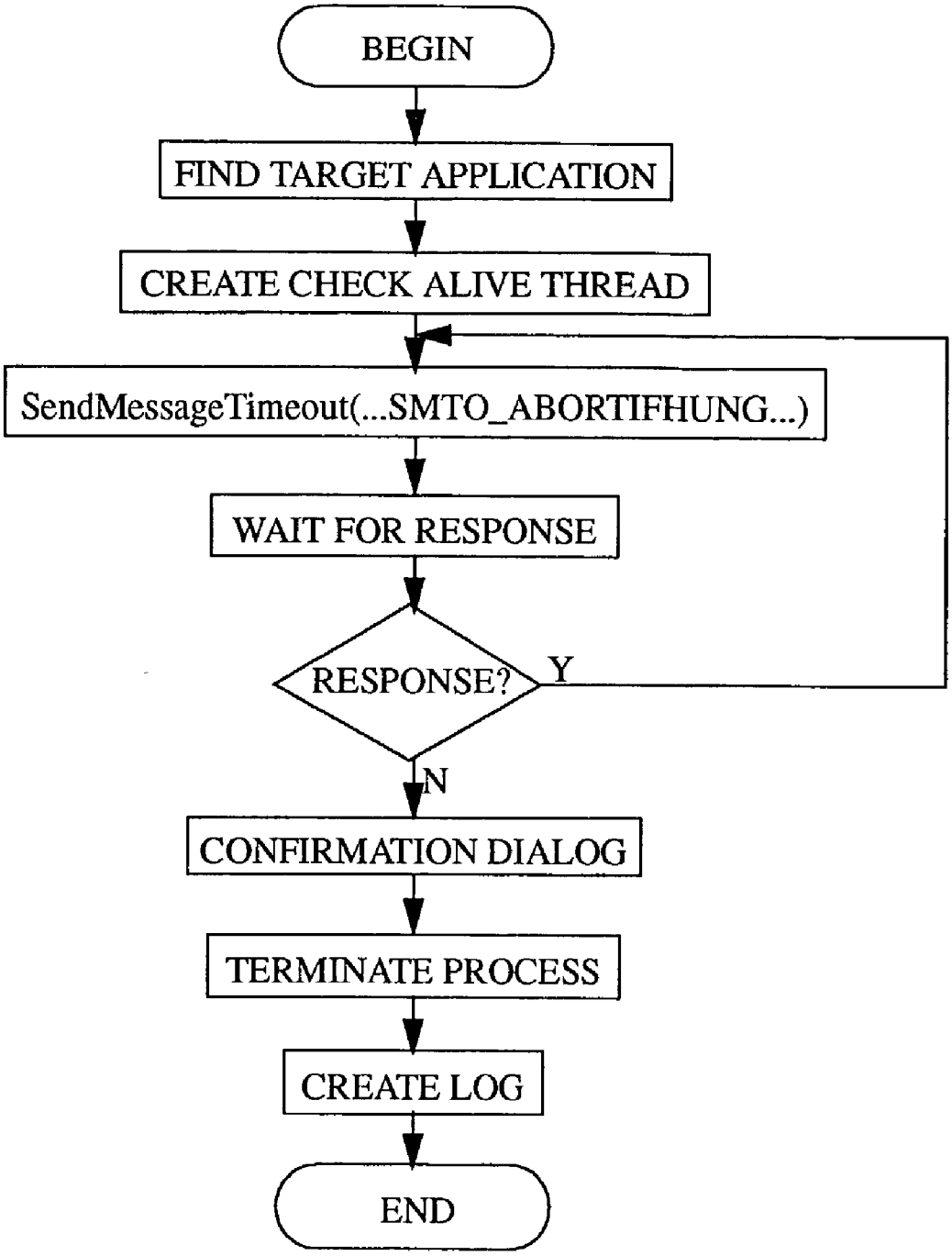

BEGIN
FIND TARGET APPLICATION
CREATE CHECK ALIVE THREAD
SendMessageTimeout(...SMTO_ABORTIFHUNG...)
WAIT FOR RESPONSE
RESPONSE?
Y
N
CONFIRMATION DIALOG
TERMINATE PROCESS
CREATE LOG
END

Fig. 4

USER TRANSPARENT SOFTWARE MALFUNCTION DETECTION AND REPORTING

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] The present invention relates to software malfunction detection and reporting.

[0003] 2. State of the Art

[0004] Software malfunction detection and reporting tools are known. One such tool is Microsoft's "Dr. Watson." Dr. Watson is a debugging tool designed to provide software programmers with detailed information on the internal state of the Windows operating system when a Unhandled Application Exception (UAE) occurs. Dr. Watson must be running at the time a UAE occurs to extract the internal information from the system.

[0005] Dr. Watson uses comparatively little memory and does not affect the performance of Windows. A software programmer therefore has no reason not to install Dr. Watson, especially if a UAE has occurred before. After Dr. Watson is installed, information is collected when a UAE occurs and written to a special file (DRWATSON.LOG) located in the Windows directory. In addition, a Dr. Watson dialog box will appear, prompting the user to enter a description of what the user was doing prior to receiving the error. A developer may choose to start Dr. Watson automatically each time Windows is started, thus allowing critical information to be collected each time a UAE occurs. After several UAEs have been logged, the log may, if desired, be sent to a remote location for diagnosis.

[0006] Dr. Watson is a diagnostic tool, not a cure for a problem. Having Dr. Watson will not prevent an error from occurring, but the information in DRWATSON.LOG often helps developers determine the cause of the error.

[0007] Another tool, the Microsoft Diagnostics (MSD) program, is designed to assist Microsoft customers and Product Support Services (PSS) technicians in solving problems with Microsoft products. MSD identifies system configuration information such as the BIOS, video card type and manufacturer, installed processor(s), I/O port status, operating system version, environment settings, hardware devices attached, and additional software running concurrently with MSD.

[0008] MSD is often used in conjunction with Dr. Watson to provide information on hardware configurations and UAEs. Error reports may be generated that include information from both the MSD program and Dr. Watson.

[0009] As may be appreciated from the foregoing description, existing software malfunction detection and reporting tools are targeted to software developers. Significant computer expertise and manual interaction is required in order to use these tools effectively. In particular, such tools are not well suited for beta users. Although beta users are supposed to report crashes and malfunctions, a beta user may experience a malfunction but, in the day-to-day rush of business, fail to report it. A need exists for a software failure and detection tool well suited for beta users such that a software developer can obtain malfunction reports and diagnostic information easily and reliably.

SUMMARY OF THE INVENTION

[0010] The present invention, generally speaking, "instruments" an arbitrary software program, without changing the software program, to automate malfunction detection and reporting. Although users can be invited to enter a description of what the user was doing prior to receiving the error, report generation and transmission to a remote server can be fully automatic and transparent to the user. In the case of beta testing, therefore, a software developer is guaranteed to receive all pertinent information about malfunctions of an application without having to rely on "fallible humans" for this information. The effectiveness of beta testing, in terms of ultimately contributing to an improved product, is therefore greatly increased. Various kinds of malfunctions may be detected and reported, including an application "crashing," becoming "hung," etc.

BRIEF DESCRIPTION OF THE DRAWING

[0011] The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

[0012] **FIG. 1** is a block diagram of a software malfunction detection and reporting system in accordance with an exemplary embodiment of the invention;

[0013] **FIG. 2** is an example of a log created by the software agent of **FIG. 1**;

[0014] **FIG. 3** is a flow chart illustrating operation of the Micro Debugger thread of **FIG. 1**; and

[0015] **FIG. 4** is a flow chart illustrating operation of the Check Alive thread of **FIG. 1**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0016] Referring now to **FIG. 1, a** block diagram is shown of an automated malfunction detection and reporting system in accordance with an exemplary embodiment of the present invention. Communication of a malfunction report may be accomplished in any of various different ways, for example via a LAN connection, with data collection occuring inside a corporation, via the Internet, of simply by means of the generation of an ASCII report which is then inserted into an email outbox. In accordance with one advantageous embodiment, to be presently described, a user machine is assumed to have installed software realizing a "polite agent" that communicates messages of abbreviated length with a remote server using a "virtual connection," e.g., an "opportunistic" Internet connection. Such a polite agent is described in U.S. application Ser. No. _____ (Attorney's Docket No. 031994-026) filed _____, incorporated herein by reference. The polite agent technology described in published International Patent Application WO 97/07656, also incorporated herein by reference, may also be used with the present invention.

[0017] The agent is provided with a list of active processes, or "target applications, with which the agent is to interact, e.g., to perform automated malfunction detection and reporting. One such target application is shown in **FIG. 1**. As illustrated, a target application will typically include a main body of executable code (.EXE), various Dynamic Link Libraries (.DLL), various Custom Control and COM

objects (.OCX), etc. The terms EXE, DLL, OCX, etc., are specific to the Window operating system. In the case of other operating systems, other terminology is applied to similar concepts. The invention may be applied in connection with any operating system.

[0018] When a process is started on the user machine, the agent "hooks" (installs code realizing an application hook for) that process. The application hook then identifies the process either as a process of interest, i.e., one of the agent's active processes, or as a process not of interest. In the latter case, the application hook removes itself. The function of the application hook is to intercept and filter messages from the target application and communicate messages of interest to the agent, which maintains a timestamped event log. The application hook and the agent communicate through a hook support thread using operating system messaging facilities, e.g., in the case of the Windows operating system, SendMessage(. . .) and PostMessage(. . .). For purposes of malfunction detection and reporting, messages of interest include menu selections, loading of a DLL, etc. These events are timestamped by the agent and entered into the event log.

[0019] Note that multiple target application may be active at the same time, in which case each target application would have its own application hook, hook support thread and timestamped event log.

[0020] The application hook allows relatively high-level activity information concerning the target application to be captured. Such activity information is readily intelligible to the average software programmer, in contrast to the much more arcane information gathered by known malfunction detection and reporting tools, such as stack frames, pointers, etc.

[0021] In addition to the application hook, two addition threads are created on a per-application basis, a Micro Debugger thread and a Check Alive thread.

[0022] The Micro Debugger thread is registered with the operating system as the debugger for the target application using the appropriate operating system call, e.g., DebugActiveProcess(. . .). This call waits until an exception occurs in the target application and then returns control to the Micro Debugger. The target application therefore runs under the control of the Micro Debugger. The Micro Debugger is notified by the operating system when a "crash" (UAE) occurs. The Micro Debugger communicates this information to the agent, performs cleanup for the target application, and terminates the target application. The agent then solicits additional information from the user concerning the malfunction.

[0023] Check Alive is an optional feature that may be specified when the instruction to monitor a particular application is given. The purpose of the Check Alive thread is to detect when the target application has hung, i.e., become non-responsive to user input. The Check Alive performs this function by repeatedly "pinging" the target application, i.e., send a message to the target application to see if the target application responds. A timeout period is set for the message such that if the target application does not respond within the timeout period, control is returned to the Check Alive thread. If Check Alive thread detects that the application is hung, it notifies the agent, which then solicits confirmation and additional information from the user.

[0024] After the target application has either crashed or becomes hung, the agent assembles and (optionally) compresses the "crash log" as part of a bug report form and schedules the bug report form for automatic upload to the server using "polite connection" methods or some other connection. An example of a crash log is shown in **FIG. 2**.

[0025] Referring now to **FIG. 3**, operation of the Micro Debugger thread will be described in greater detail. Once an application hook has found a target application, a Micro Debugger thread is created for that target application and registered with the operation system as the debugger for the target application process. The Micro Debugger then causes execution of the process to be then interrupted while the operating system enumerates modules that the process has loaded. In the case of the Windows operating system, the operating system identifies each module loaded by the target application thread along with the memory location of that module. When all of the modules have been enumerated, this information is sent to the agent to be logged, and the target application process is allowed to continue running. Any modules loaded or unloaded thereafter will be detected by the Micro Debugger and logged as described previously.

[0026] When an exception occurs, the operating system calls the Micro Debugger, which determines what kind of exception has occurred and where in memory the exception occurred. The Micro Debugger also captures additional context of the exception, e.g., what happened (divide by zero, access violation, etc.) and in which module. The Micro Debugger then sends this information to the agent to be stored in the event log. Finally, the Micro Debugger terminates the target application. Optionally, prior to terminating the target application, the Micro Debugger may perform cleanup for the target application. In order to do so, the Micro Debugger will typically require some detailed knowledge of the target application. Given such knowledge, the Micro Debugger could, for example, save a user's work prior to terminating the target application such that the user's work is not lost.

[0027] Referring to **FIG. 4**, operation of the Check Alive thread will be described in greater detail. Once an application hook has found a target application, a Check Alive thread is created for that target application. The Check Alive thread then enters a loop in which the following actions are performed. The Check Alive thread sends a message with timeout to the target application. The timeout period may be, for example, 5 seconds. The Check Alive thread then waits for a response from the target application. If a response is received within the timeout period, the Check Alive thread waits for a period of time (e.g., about 15 seconds) before sending another message with timeout. This manner of operations continues for so long as the target application is running and responsive.

[0028] If a response is not received from the target application within the timeout period, either the Check Alive thread or the agent attaches a dialog box to the application window asking the user to confirm that the target application has hung. If the user concurs, then the process is terminated. The agent then creates a crash log and schedules it for upload to the server as described previously.

[0029] Automatically collecting malfunction information on the server without the need for user intervention greatly facilitates the job of a software developer to find and fix

program bugs. A beta test coordinator, for example, may remotely log onto the server and view malfunction information. The beta test coordinator may find, for example, that of 200 installed copies of a program, five copies have crashed within a week's time. Based on this information—the reliability of which is assured—priorities may then be set to resolve the problem.

[0030] It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

What is claimed is:

1. An automated method of software malfunction detection and reporting, comprising:

detecting a software malfunction;

capturing an execution context of the malfunction; and

automatically sending malfunction information including the execution context to a remote server.

2. The method of claim 1, wherein the malfunction information is automatically sent in a manner transparent to a user of the software.

3. The method of claim 1, wherein the server is used by a software developer to gather malfunction information concerning a pre-release version of a product of the software developer.

4. The method of claim 1, further comprising selectively monitoring at least one software application.

5. The method of claim 4, wherein selectively monitoring comprises:

inserting a hook into a thread as it begins execution;

using the hook to determine an identifier of the thread; and

comparing the identifier to a list of identifiers of threads belonging to target software applications.

6. The method of claim 5, further comprising, depending on results of the comparing step, either removing the hook or allowing the hook to remain.

7. The method of claim 5, further comprising:

using the hook, logging user activity within an application; and

including user activity information within the malfunction information.

8. The method of claim 7, wherein the user activity information includes information concerning which user interface commands were selected.

9. The method of claim 8, wherein the user activity information further includes information concerning when such user interface commands were selected.

10. The method of claim 5, further comprising, for an execution thread belonging to a target software application, creating a control thread and registering the control thread as a debugger for the execution thread.

11. The method of claim 10, wherein the control thread receives an unhandled application exception message and, in response, captures the execution context of the execution thread.

12. The method of claim 11, wherein the control thread, in response to the unhandled application exception message, terminates the execution thread.

13. The method of claim 12, wherein the control thread, prior to terminating the execution thread, performs cleanup for the execution thread.

14. The method of claim 5, further comprising, for an execution thread belonging to a target software application, creating a responsiveness monitoring thread.

15. The method of claim 14, wherein the responsiveness monitoring thread periodically sends messages to the execution thread.

16. The method of claim 15, wherein the messages have a timeout period specified.

17. The method of claim 16, wherein the responsiveness monitoring thread, if it does not receive a reply from the execution thread within the timeout period, performs at least one of the following actions: displays to a user a dialog within a user interface space of the execution thread asking the user to confirm that the application has hung; captures the execution context of the execution thread; performs cleanup for the execution thread; and terminates the execution thread.

18. The method of claim 17, wherein, if the user confirms that the application has hung, the responsiveness monitoring thread terminates the execution thread.

19. A computer-readable medium containing a software agent including program instructions for:

maintaining a list of active process threads;

when a new process thread is started, if it is a target process thread belonging to a target application to be monitored, hooking the target process thread and creating one or more detection threads for automatically detecting events within and malfunction of the target process thread; and

maintaining a log of events within the target process thread.

20. The apparatus of claim 19, wherein the software agent further includes program instructions for:

when a malfunction is detected by the one or more detecting threads, creating a malfunction log including the log of events and further including execution context information of the target process thread; and

preparing the malfunction log to be sent to a remote server.

21. The apparatus of claim **20**, wherein the software agent further includes program instructions for sending the malfunction log in a manner transparent to a user of the application.

* * * * *