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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 December 2001 (20.12.2001)

PCT

(10) International Publication Number
WO 01/97039 A1

(51) International Patent Classification⁷: G06F 11/36

(21) International Application Number: PCT/IB01/01286

(22) International Filing Date: 14 June 2001 (14.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
00401687.9 14 June 2000 (14.06.2000) EP

(71) Applicant *(for all designated States except US)*: **CANAL+ TECHNOLOGIES SOCIETE ANONYME** [FR/FR]; 34, place Raoul Dautry, F-75906 Paris (FR).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **GUISSOUMA, Habib** [TN/FR]; Canal+ Technologies Société Anonyme, 34, place Raoul Dautry, F-75906 Paris (FR).

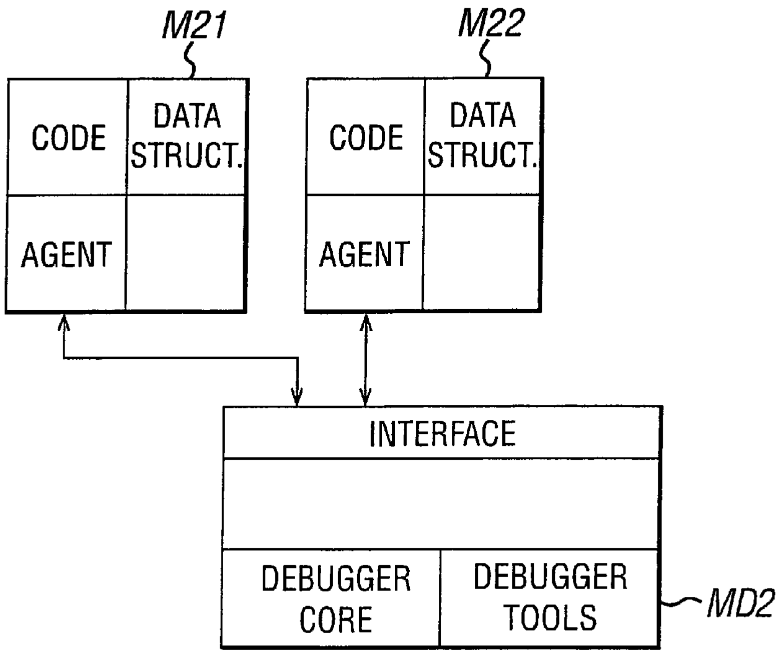
(74) Agents: **COZENS, Paul, Dennis** et al.; Mathys & Squire, 100 Gray's Inn Road, London WCIX 8AL (GB).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EC, EE, EE (utility model), ES, FI, FI (utility model), 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, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: REMOTE DEBUGGING IN AN EMBEDDED SYSTEM ENVIRONMENT



(57) **Abstract:** A software debugging provides for compiling software modules and a debugger module on a workstation WS. The obtained executable test file 8 is transferred to an end user system, such as a receiver/decoder IRD, where it is executed. The debugger module MD, MD2 may output information on commands being executed to a client on the workstation. In one embodiment specific code containing information of the software modules is included in the debugger module MD. In another embodiment the software modules register commands and a point of entry to the debugger module MD2.

WO 01/97039 A1

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

REMOTE DEBUGGING IN AN EMBEDDED SYSTEM ENVIRONMENT

The present invention relates to a method of debugging software.

The development of software generally involves a debugging process before the software is released for use. The debugging process enables errors to be removed in the software. Such errors could for example cause the software to interrupt its execution in an unpredictable way because of faulty commands, or in some cases even generate erroneous results or result in dangerous operation of machines controlled by the software.

It is known from software development in UNIX computer environments to insert debug code in the software code. The debug code usually has no useful function for the software. It will merely be used in conjunction with the debugging process. The debug code is deleted from the software once the debugging process is completed. This reduces the size of the final software code file.

The debug code enables a plurality of debugging functions. The debug code may for example define marks in the software in order to monitor how the software is executed or to indicate a halting point at which the execution of the software is to be halted. The debug code may also enable the return of a value to the user of a determined variable or memory content.

Once the software code to test and its debug code are ready to test, they may be compiled to obtain an executable debug file.

A next step in the debugging process involves executing a debugging software in a UNIX process and the executable debug file in another UNIX process. The debugging software allows for example to identify the aforementioned halting points at which the execution of the executable debug file may be halted or the printing on a screen of memory content which is or is not affected by the running of the executable debug file. Other functionalities may be provided.

executable debug file. Other functionalities may be provided.

The debugging software requires information about data structures used in the software code and during compiling of the code in order to identify these data structures during the debugging process. Such information may for example be the size of data structures measured in bits. The debugging software further requires information about the system on which the compiled software, i.e., the executable debug file is being executed in order to interact with this system. Such information may for example be which mechanisms are used by the system to initiate signal exchanges between the executed software and the system. In a UNIX software development environment which typically comprises a workstation and a mass storage medium such as a magnetic hard disk drive, the information required by the debugger software may easily be stored and retrieved.

Hence the debugger software may interact with the executable debugger file due to its knowledge of the system. The debugger software may for example cause the running executable debug file to halt by substituting an appropriate instruction to a command included in the executable debug file's program code. In a similar way the debugger software may cause the running to resume by removing the latter appropriate instruction and replacing it by the original command of the executable debug file, and trigger the further execution of the executable debug file.

The debugger software could also cause the executable debug file to be executed departing from a determined line in the program code. The determined line could for example be indicated by the user.

Another useful feature of the debugger software is the possibility to modify the value of memory content, e.g., modifying a value of a variable used by the executable debug file. This enables simulation of the behaviour of the software being tested for selected values of the concerned variable.

The software development for an embarked environment, or an end user system where the software is to be executed by an end user, may require an adapted debugging process.

An example of an embarked environment is an integrated receiver/decoder (IRD) as used for receiving and decoding TV signals and other applications, which are broadcast as a flow of digital data. An IRD comprises a storage in which software may be stored and a microprocessor for running the software. The IRD further comprises an operating system as well as appropriate drivers and components. One difference between an IRD, and embarked environments in general, and a workstation is that they generally have smaller computing and/or memory capacity.

The software development for such an end user system is typically performed using a workstation WS. The software code may be prepared with an appropriate editor and a debugging of the software may be performed on the workstation itself. The debugging could for example make use of a simulator for executing the software by simulating the end user system. An inconvenience of this way of debugging is that a simulation of the end user system is likely not to reflect the exact same behaviour as the end user system itself. This becomes even more critical if the end user system is subject to changes in its configuration and the simulator needs to be modified accordingly.

An aim of at least the preferred embodiments of the present invention is to overcome the inconveniences of the prior art.

In a first aspect the present invention provides a method of debugging software comprising at least one software module containing code for implementing at least one command, said method comprising the steps of creating on a workstation an executable test file by compiling said at least one software module with a debugger module, and transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

Preferably, during execution of said executable test file, data is exchanged between said executable test file and software running on the workstation.

Preferably, data transmitted from said debugger module to said workstation during execution is displayed on a terminal of the workstation.

In one embodiment the debugger module includes information regarding each of said at least one software module to be debugged. Preferably, said information comprises a description of the commands and data structure of each of said at least one software module. Using said information, a command implemented by said at least one software module may be executed by the debugger module during debugging.

In an alternative embodiment information provided to the debugger module by each of said at least one software module is used to debug that module. Said information may enable the debugger to execute a command implemented by that software during debugging. Preferably, an agent is included in each of said at least one software module for use in interfacing with the debugger module. During execution of said executable test file, each agent may register its associated software module with the debugger module to enable the debugger module to receive said information for that software module. Preferably an agent of a first software module has access to at least one command provided by a second software module.

In another aspect the present invention provides a workstation comprising means for creating an executable test file by compiling at least one software module containing code for implementing at least one command with a debugger module, and means for transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

Preferably the workstation comprises means for exchanging data between said executable test file and software running on the workstation during execution of said executable test file. The workstation preferably comprises a terminal for displaying data transmitted from said debugger module to said workstation.

This aspect of the present invention extends to a system for debugging software comprising a workstation as aforementioned and an end user system for receiving and executing the executable test file to cause said at least one software module to be debugged using the debugger module.

Preferably said end user system comprises a receiver/decoder of a digital broadcast system. The software may comprise software for a kernel layer of the operating system of the receiver/decoder.

In another aspect the present invention provides a computer program product adapted to create an executable test file by compiling at least one software module containing code for implementing at least one command with a debugger module, and to transfer said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

Preferably the product is adapted to exchange data with said executable test file during execution of said executable test file

The present invention also provides a computer program product adapted to carry out a method as aforementioned, and a computer program adapted to carry out a method as aforementioned.

The present invention further provides a computer readable medium having stored thereon a program for compiling at least one software module containing code for implementing at least one command with a debugger module, and transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

The present invention further provides a signal tangibly embodying a computer program for carrying out the method as aforementioned.

The present invention also provides a method of debugging a software comprising at least one software module which contains code to implement at least a command, comprising creating an executable test file by compiling the software module together with a debugger module.

The term "receiver/decoder" used herein may connote a receiver for receiving either encoded or non-encoded signals, for example, television and/or radio signals, which may be broadcast or transmitted by some other means. The term may also connote a decoder for decoding received signals. Embodiments of such receiver/decoders may include a decoder integral with the receiver for decoding the received signals, for example, in a "set-top box", such a decoder functioning in combination with a physically separate receiver, or such a decoder including additional functions, such as a web browser, a video recorder, or a television.

The term "embarked environment" includes, inter alia, the environment in which the software is to be executed by the end user, such as a receiver/decoder.

Features of one aspect may be applied to other aspects; method features may be applied to the workstation and computer product aspects and *vice versa*.

Preferred features of the present invention will now be described, purely by way of example, with reference to the accompanying drawings, in which:-

Figure 1 contains a schematic illustration of a software development environment for an end user system ;

Figure 2 illustrates logical layers of an end user system environment ;

Figure 3 contains a block diagram illustrating a method to obtain an executable file for an end user system ;

Figure 4 contains a block diagram illustrating a method to obtain an executable test file ;

Figure 5 shows a logical build-up of a debugger module;

Figure 6 contains software modules and a debugger module ;

Figure 7 contains other software modules and a debugger module.

Referring to Fig. 1 a software development system for an embarked environment, or end user system, comprises a workstation WS and, as an example of an end user system, an integrated receiver/decoder (IRD). As is known from the prior art the software code may be prepared on the workstation WS using an appropriate editor. A preliminary debugging of the software may be performed on the workstation WS itself. The debugging may make use of a simulator for executing the software by simulating the embarked environment. The software is subsequently transferred to the IRD where further debugging is done. The IRD is connected to an output device such as a television TV on which video and audio signals may be rendered.

Referring now to Fig. 2, a schematic representation of the IRD in the form of logical layers shows a hardware resource layer 1 which corresponds to low level software, such as drivers, required to make use of the IRD hardware features. The hardware resource layer 1 is typically provided with the IRD hardware by the IRD manufacturer.

A virtual machine comprising a system kernel layer 2 and an Advanced Programming Language APL kernel layer 3 provides an environment for running software written in the advanced programming language, such as for example the JAVA ™ language. The virtual machine may be downloaded to the IRD during the software development phase for testing or by the manufacturer before the IRD is delivered to customers.

A customised Application Programming Interface API 4 may be provided to be executed on top of the virtual machine. The API creates an interface between the virtual machine and the programmers of application. The API may be entirely specified, e.g. by means of an industry standard.

An application layer 5 represents software to be executed by the virtual machine. The software may for example be an interactive television guide programme which will be downloaded by the IRD when it is connected to an appropriate source such as a cable network, a satellite dish receiver or a terrestrial antenna by means of which digital TV data may be received.

Typically the APL kernel 3 software is developed using the software development system in Fig. 1 .

Referring to Fig. 3 the APL kernel software may comprise one or more modules M1, M2, ...MN which is or are subject to a compilation 6 to obtain an executable file 7. The executable file 7 is downloaded from the workstation WS to the IRD where it may be executed.

Referring to Fig. 4 the debugging process of the APL kernel software is prepared on the workstation WS by including a debugger module MD at the compilation 6 of the APL kernel software in order to obtain an executable test file 8. The executable test file 8 is downloaded from the workstation WS to the IRD where it may be executed.

During execution of the executable test file 8 typical debugging action such as halting the execution, returning values of variables, modifying the executable test file 8 and others may be undertaken as will be explained below.

First preferred embodiment

Software module(s) M1, M2, ... of APL kernel

The software module(s) of the APL kernel is/are designed to offer an Application Programmer Interface which is specified in the Advanced Programming Language interface.

Generally there may be a plurality of modules which each provide a distinct type of functionalities, i.e., one module may be dedicated to the handling of a file system in the virtual machine, another module may be dedicated to communication of signals from the application to hardware interfaces of the IRD, a further module may provide access to graphical display features, and so on.

Each software module thus provides code for implementing commands which may be used, after compilation, in the virtual machine of the IRD to access the functionalities associated with the module. The software module further provides data structures associated with the commands.

Debugger module MD

Referring to Fig. 5 a schematic representation of the debugger module MD will be used to better understand the functionalities associated to it. The representation is not intended to reflect an actual implementation in code of the debugger module MD.

The debugger module MD comprises specific code represented in boxes labelled code M1, code M2 ... code MN. This specific code contains information on each software module to be included in the compilation with the debugger module. More precisely this specific code may comprise a description of commands and data structures provided by each module.

The debugger module further comprises code for implementing debugger tools commands. These debugger tools commands may for example be the following :

- A set of functions for implementing communication with a client of the debugger module MD, i.e., a software which runs on a workstation such as WS in Fig. 1 and which enables the display of data outputted by the debugger module MD to the user or to receive instructions from a user before transmitting these to the debugger module MD. This set of functions may enable formatting and sending a chain of characters;

- A set of functions for handling instructions contained in a command line inputted from the client;
- A set of functions to control the execution of the executable test file being debugged;
- A set of functions to handle a configuration of the debugger module, i.e., to indicate how the debugger module will behave when it encounters errors during execution of the executable test file being debugged;

Generally speaking the debugger tools comprise commands which are of use for most or all of the software modules. This enables reduction of the size of the executable test file because such commands do not have to be implemented in the individual software modules.

A debugger core of the debugger module MD makes use of the specific code in boxes Code M1, Code M2, ..., and of the debugger tools to interact with the software modules M1, M2, ... during debugging and in accordance with instructions received by a user through the client of the debugger module.

Second preferred embodiment

Software modules M21, M22, ... of APL kernel

Referring to Fig. 6 and similar to what has been described for the first preferred embodiment, software modules M21 and M22 provide code for implementing commands which may be used, after compilation, in the virtual machine of the IRD, and data structures associated to the commands.

Fig. 6 contains a schematic representation of 2 software modules M21 and M22. It will be clear for a person skilled in the art that only one module or a greater number of modules may also be used.

In addition to what has been described for the first preferred embodiment, the software modules M21 and M22 each comprise an agent code which enable communication to be established with a debugger module MD2.

Debugger module MD2

In a similar fashion to the debugger module MD described in relation with the first preferred embodiment, the debugger module MD2 comprises code for implementing debugger tools commands and a debugger core.

One difference between debugger modules MD2 and MD is that the former does not necessarily contain code describing the software modules to debug. In other words the debugger module MD2 taken alone has no information about the software modules for which it will be used. Instead the debugger module MD2 will make use of information provided by each software module to debug and contained in the executable test file after compilation.

As a result the debugger module MD2 has a reduced size as compared to the debugger module MD. Furthermore the debugger module MD2 does not need to be updated if a software module to debug is added or deleted from the executable test file.

The information provided by each software module will make specific commands implemented by each software module available during debugging.

Debugging with MD2

Similarly as described above for Fig. 4, the software modules M21, M22, ... and the debugger module MD2 are compiled to obtain the executable test file.

During execution of the executable test file the agents of the software modules M21, M22, ... register their respective module at an interface of the debugger module

MD2. The debugger module thereby receives for each software module a point of entry of the agent, a list of commands that the agent takes in charge and information concerning online help for using the listed commands. The point of entry indicates to the debugger module MD2 how it can access the agent. This might for example be a logical address. This will be required for example whenever the debugger module MD2 wants to have one of the listed commands executed.

The listed commands allows to access the specific functionalities provided by the software module. Hence the debugger module MD2 may have access to all functionalities of the executable test file even if a software module is added at a further stage of compilation, e.g. if the software needs to be updated.

The debugger module MD2 may transmit to the agent(s) and via the interface a list of debugging tools and services available and required for the debugging. The debugging tools and services are similar to the ones described here above for the debugger module MD. Hence the software module may be accessed by the user by use of the debugger module's client and the software module which is incorporated in the executable test file may return output information to the client.

Sharing of software module commands

In a preferred embodiment and referring to Fig. 7, an agent of a first software module 9 may have access to a determined command provided by a second software module 10.

The determined command is listed in a list of commands 11 which comprises commands that the second software module has submitted previously to the debugger module 12 by means of its agent and further commands available from the debugger tools. Using the list of commands 11 the debugger module 12 may have access to the determined command by the point of entry of the second software module's agent. The determined command may however not be implemented directly by the debugger module as is the case for the debugging tool commands, since the corresponding code for it is comprised in the second software module. Instead an appropriate code redirects calls of the determined command to the agent of the second software module 10.

Hence the agent of the first software module 9 may call the determined command in the list of commands 11 through the interface of the debugger module 12. The debugger module 12 redirects the call to the agent of second software module 10.

Communication between debugger module and client

The exchange of information between the debugger module and the client on the workstation may be done over a network or a link which connects both. This may for example be a serial link.

In case the execution of the executable test file is halted, the debugger module transmit a request for prompt to the client. The client will receive from the user and send back to the debugger module the command(s) which are inputted e.g. over a keyboard.

During execution of the executable test file, the debugger module is always ready to receive messages (e.g. commands) from the client.

On receiving a message the debugger module will determine the agent of the software module concerned by the message. This may be done e.g. by identifying the command contained the message. If no agent may be found, the debugger module returns an error message to the client. If an agent is found then the debugger module calls a command handling function of the agent.

The command handling function may be divided as follows :

- Identification and testing of arguments in the command ;
- Execution of the command. If there is an action on the embarked environment

system then perform this action. If there is a memory content to be read out then display this memory content ;

- Display result of command ;
- Return an error code to the debugger module.

Once the handling function has been performed and unless the debugger module resumes the execution of the executable test file, the debugger module transmits a request for prompt to the client.

In a preferred embodiment it is possible to modify display options of the debugger module at the prompt. One of these options may correspond to the display of an error code value at the end of processing a command. Hence it is possible to obtain information on the success of executing a command and if no success was achieved, to be informed about the reason of the error.

Each feature disclosed in the description and (where appropriate) the claims and drawings may be provided independently or in any appropriate combination.

In summary, a software debugging provides for compiling software modules and a debugger module on a workstation. The obtained executable test file is transferred to an end user system, such as a receiver/decoder, where it is executed. The debugger module may output information on commands being executed to a client on the workstation. In one embodiment specific code containing information of the software modules is included in the debugger module. In another embodiment the software modules register commands and a point of entry to the debugger module.

Claims

1. A method of debugging software comprising at least one software module containing code for implementing at least one command, said method comprising the steps of creating on a workstation an executable test file by compiling said at least one software module with a debugger module, and transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

2. A method according to Claim 1, wherein, during execution of said executable test file, data is exchanged between said executable test file and software running on the workstation.

3. A method according to Claim 2, wherein data transmitted from said debugger module to said workstation during execution is displayed on a terminal of the workstation.

4. A method according to any preceding claim, wherein the debugger module includes information regarding each of said at least one software module to be debugged.

5. A method according to Claim 4, wherein said information comprises a description of the commands and data structure of each of said at least one software module.

6. A method according to Claim 4 or 5, wherein, using said information, a command implemented by said at least one software module is executed by the debugger module during debugging.

7. A method according to any of Claims 1 to 3, wherein information provided to the debugger module by each of said at least one software module is used to

debug that module.

8. A method according to Claim 7, wherein said information enables the debugger to execute a command implemented by that software during debugging.

9. A method according to Claim 7 or 8, wherein an agent is included in each of said at least one software module for use in interfacing with the debugger module.

10. A method according to Claim 9, wherein, during execution of said executable test file, each agent registers its associated software module with the debugger module to enable the debugger module to receive said information for that software module.

11. A method according to Claim 9 or 10, wherein an agent of a first software module has access to at least one command provided by a second software module.

12. A workstation comprising means for creating an executable test file by compiling at least one software module containing code for implementing at least one command with a debugger module, and means for transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

13. A workstation according to Claim 12, comprising means for exchanging data between said executable test file and software running on the workstation during execution of said executable test file.

14. A workstation according to Claim 13, comprising a terminal for displaying data transmitted from said debugger module to said workstation.

15. A workstation according to any of Claims 12 to 14, wherein the debugger module includes information regarding each of said at least one software module to be debugged.

16. A workstation according to Claim 15, wherein said information comprises a description of the commands and data structure of each of said at least one software module.

17. A workstation according to Claim 16, wherein an agent is included in each of said at least one software module for use in interfacing with the debugger module.

18. A workstation according to Claim 17, wherein an agent of a first software module has access to at least one command provided by a second software module.

19. A system for debugging software comprising a workstation according to any of Claims 12 to 18 and an end user system for receiving and executing the executable test file to cause said at least one software module to be debugged using the debugger module.

20. A system according to Claim 19, wherein said end user system comprises a receiver/decoder of a digital broadcast system.

21. A system according to Claim 20, wherein said software comprises software for a kernel layer of the operating system of the receiver/decoder.

22. A computer program product adapted to create an executable test file by compiling at least one software module containing code for implementing at least one command with a debugger module, and to transfer said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

23. A computer program product according to Claim 22 adapted to exchange data with said executable test file during execution of said executable test file

24. A computer program product according to Claim 22 or 23, wherein the debugger module includes information regarding each of said at least one software module to be debugged.

25. A computer program product according to Claim 24, wherein said information comprises a description of the commands and data structure of each of said at least one software module.

26. A computer program product according to any of Claims 22 to 25, wherein an agent is included in each of said at least one software module for use in interfacing with the debugger module.

27. A computer program product according to Claim 26, wherein an agent of a first software module has access to at least one command provided by a second software module.

28. A computer program product adapted to carry out a method according to any of Claims 1 to 11.

29. A computer program adapted to carry out a method according to any of Claims 1 to 11.

30. A computer readable medium having stored thereon a program for compiling at least one software module containing code for implementing at least one command with a debugger module, and transferring said executable test file to an end user system for execution to cause said at least one software module to be debugged using the debugger module.

31. A signal tangibly embodying a computer program for carrying out the method according to any of Claims 1 to 11.

32. A method of debugging substantially as herein described with reference to the accompanying drawings.

33. A system for debugging software substantially as herein described with reference to the accompanying drawings.

34. A workstation substantially as herein described with reference to the accompanying drawings.

35. A computer program product substantially as herein described with reference to the accompanying drawings

36. A computer program substantially as herein described with reference to the accompanying drawings.

37. A computer readable medium substantially as herein described with reference to the accompanying drawings.

38. A signal tangibly embodying a computer program substantially as herein described with reference to the accompanying drawings.

FIG. 1

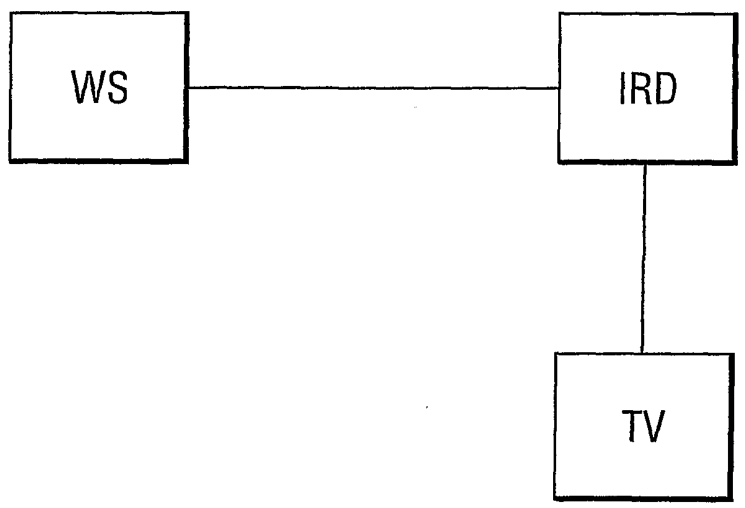

WS
IRD
TV

FIG. 2

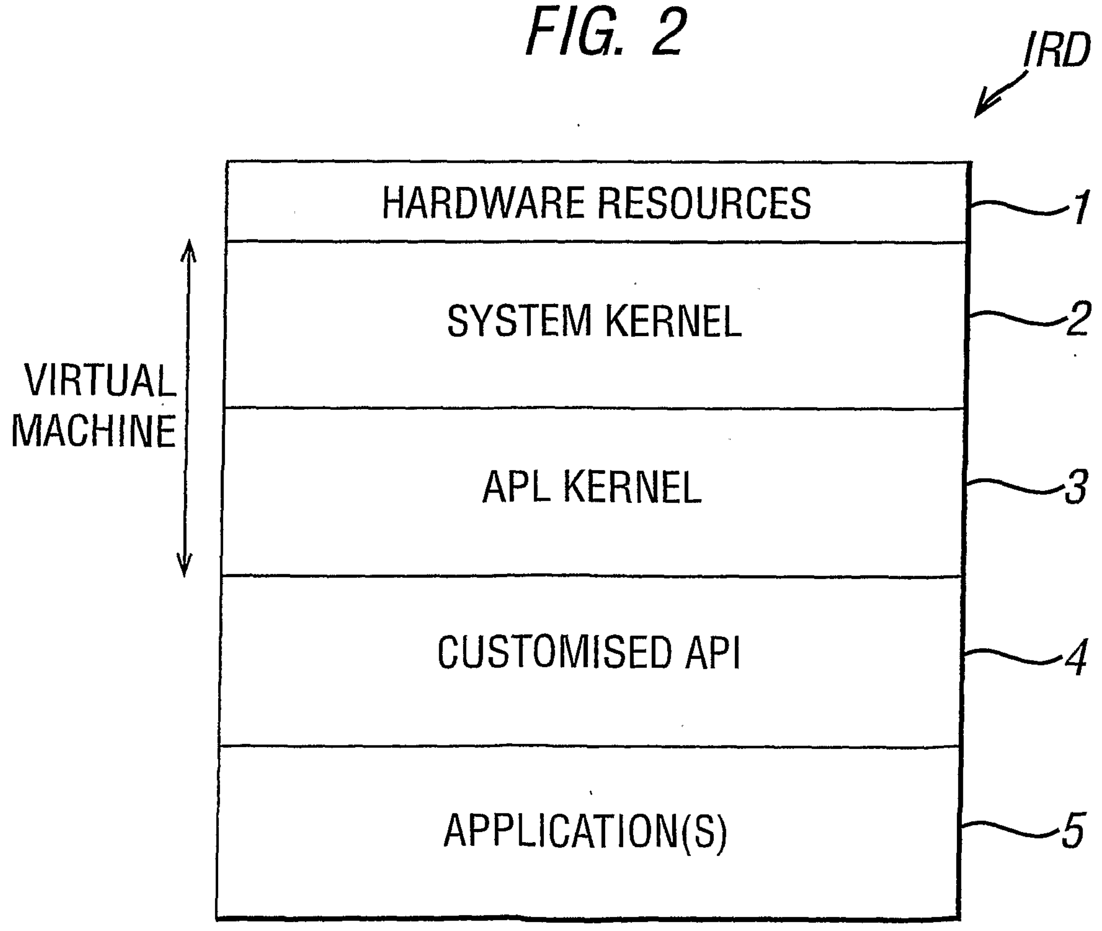

IRD
HARDWARE RESOURCES
1
SYSTEM KERNEL
2
VIRTUAL MACHINE
APL KERNEL
3
CUSTOMISED API
4
APPLICATION(S)
5

FIG. 3

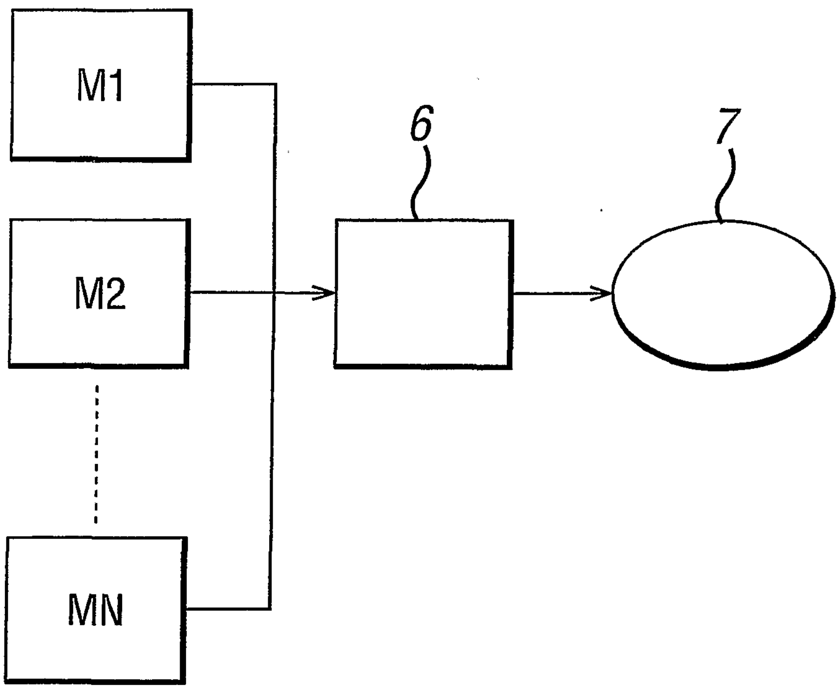
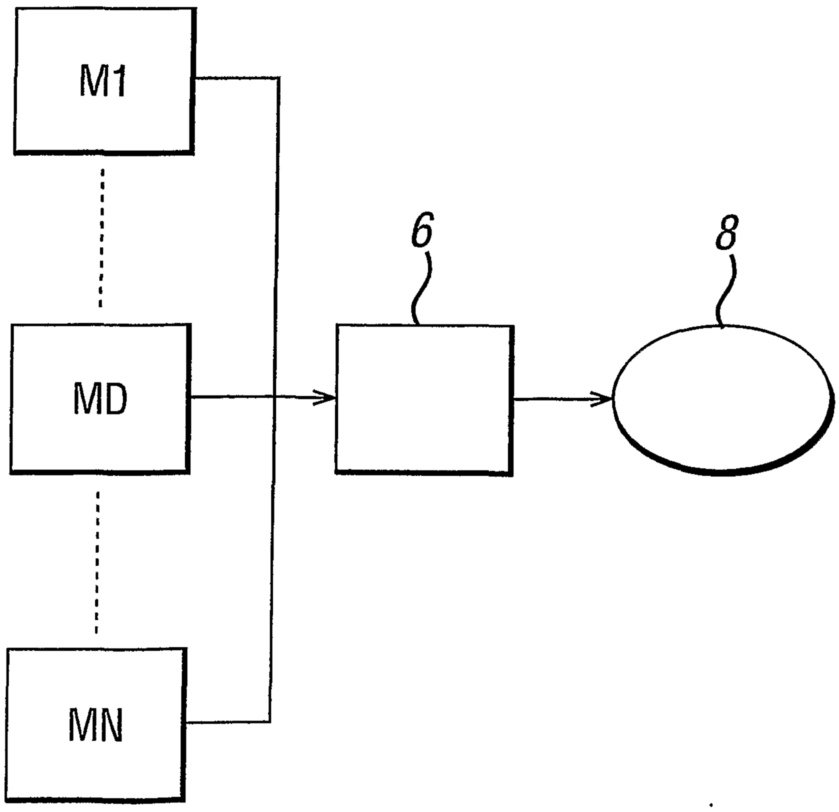


FIG. 4



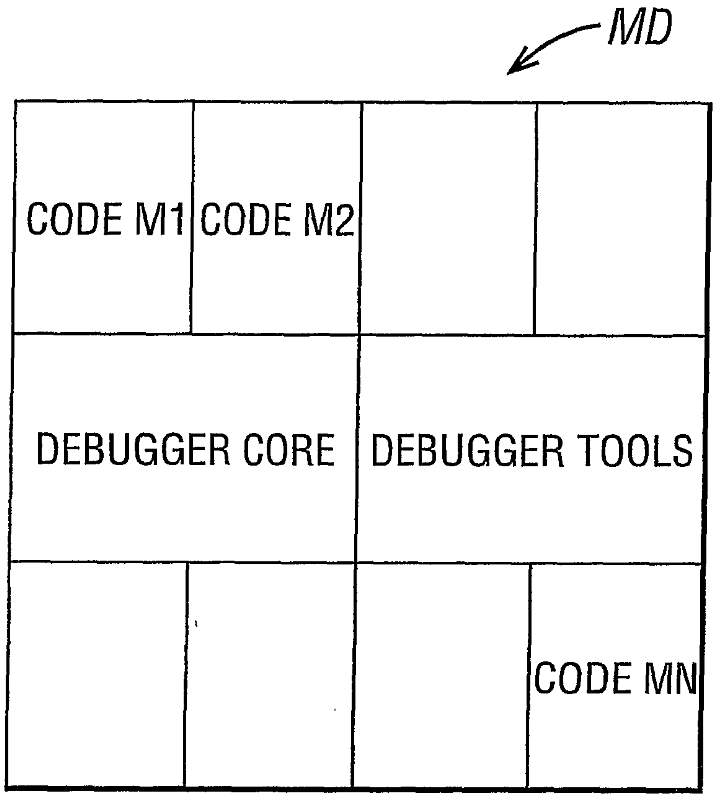

MD
CODE M1
CODE M2
DEBUGGER CORE
DEBUGGER TOOLS
CODE MN

FIG. 5

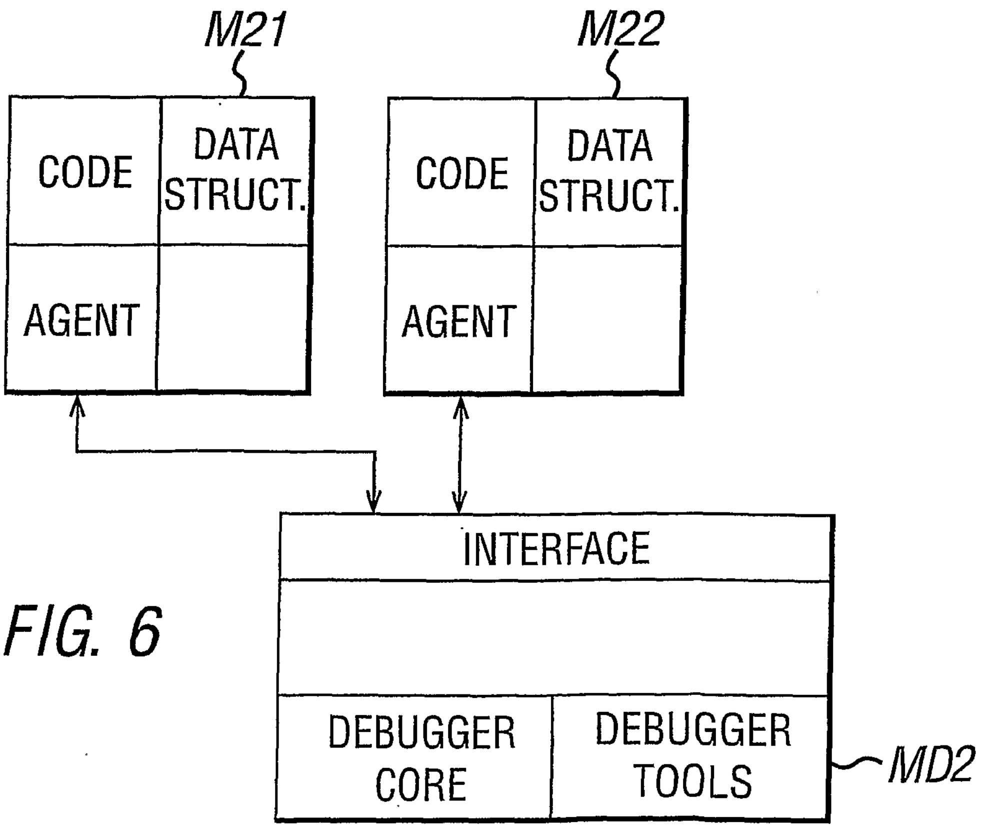
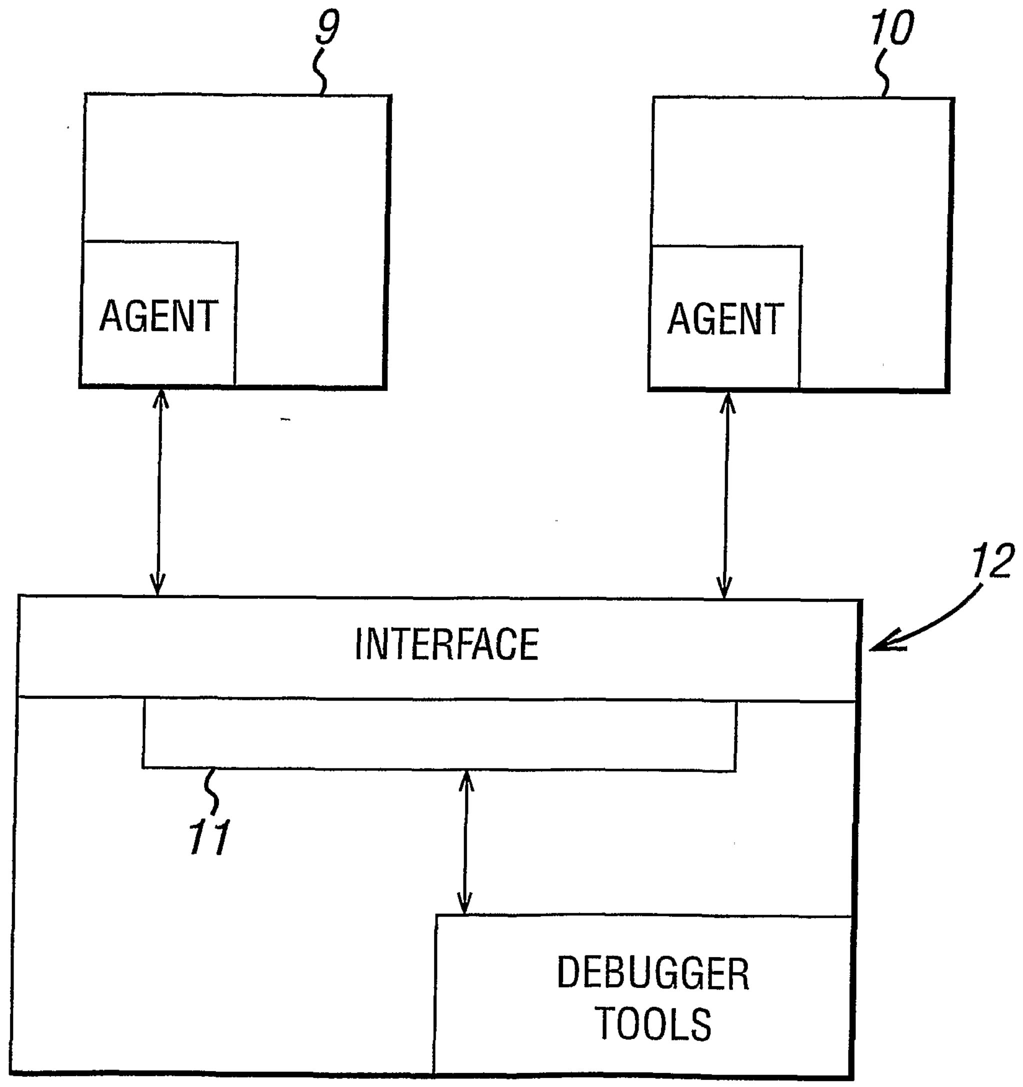

M21
M22
CODE
DATA STRUCT.
AGENT
CODE
DATA STRUCT.
AGENT
INTERFACE
DEBUGGER CORE
DEBUGGER TOOLS
MD2

FIG. 6

FIG. 7



INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB 01/01286

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/36

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 7 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)

EPO-Internal, WPI Data, PAJ, INSPEC, 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	KOLE R E: "SOFTWARE DEVELOPMENT AND DEBUGGING IN EMBEDDED SYSTEMS: MAINTAINING THE HIGH-LEVEL VIEW" ELECTRO,US,ELECTRONIC CONVENTIONS MANAGEMENT. LOS ANGELES, vol. 11, 1986, pages 22-4,1-07, XP000009785 page 4 -page 7 ---	1-8, 12-16, 19, 22-25, 28-31

-/--

[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 15 October 2001	Date of mailing of the international search report 22/10/2001
Name and mailing address of the ISA 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	Authorized officer Renault, S

INTERNATIONAL SEARCH REPORT

International Application No
PCT/IB 01/01286

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 630 049 A (CARDOZA W.M. ET AL.) 13 May 1997 (1997-05-13) abstract column 4, line 49 -column 5, line 2 column 7, line 58 -column 8, line 52 column 22, line 13 - line 67 claim 1 ---	1-8, 12-16, 19, 22-25, 28-31
A	EP 0 712 080 A (SUN MICROSYSTEMS INC) 15 May 1996 (1996-05-15) column 3, line 10 -column 4, line 31 column 5, line 3 -column 9, line 27 ---	1-31
A	US 5 715 387 A (BARNSTIJN MICHAEL A ET AL) 3 February 1998 (1998-02-03) abstract column 2, line 46 -column 3, line 28 -----	1-31

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/IB 01/01286

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 5630049	A	13-05-1997	NONE		
EP 0712080	A	15-05-1996	EP	0712080 A1	15-05-1996
			JP	8241201 A	17-09-1996
			US	5815712 A	29-09-1998
US 5715387	A	03-02-1998	US	5600790 A	04-02-1997