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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 March 2007 (15.03.2007)

PCT

(10) International Publication Number
WO 2007/028227 A1

(51) International Patent Classification:
G06F 11/36 (2006.01) *G06F 9/44* (2006.01)

(21) International Application Number:
PCT/CA2005/001380

(22) International Filing Date:
9 September 2005 (09.09.2005)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **IBM CANADA LIMITED - IBM CANADA LIMITEE** [CA/CA]; 3600 Steeles Avenue East, Markhan, Ontario L3R 9Z7 (CA).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **LO, Grace, Hai, Yan** [CA/CA]; 68 Aberfeldy Crescent, Thornhill, Ontario L3T 4C4 (CA). **FUNG, Jane, Chi-Yan** [CA/CA]; 44 Campbell Avenue, Thornhill, Ontario L4J AY2 (CA). **O'FARRELL, William, Gerald** [CA/CA]; 115 Russell Jarvis Drive, Markham, Ontario L3S 4B3 (CA). **TAN, Shu, Xia** [CA/CA]; 94 Tarragona Boulevard, Toronto, Ontario M6N 5C5 (CA).

(74) Agent: **HOICKA, Leonora, K.**; IBM Canada Ltd., 3600 Steeles Avenue East, Markham, Ontario L3R 9Z7 (CA).

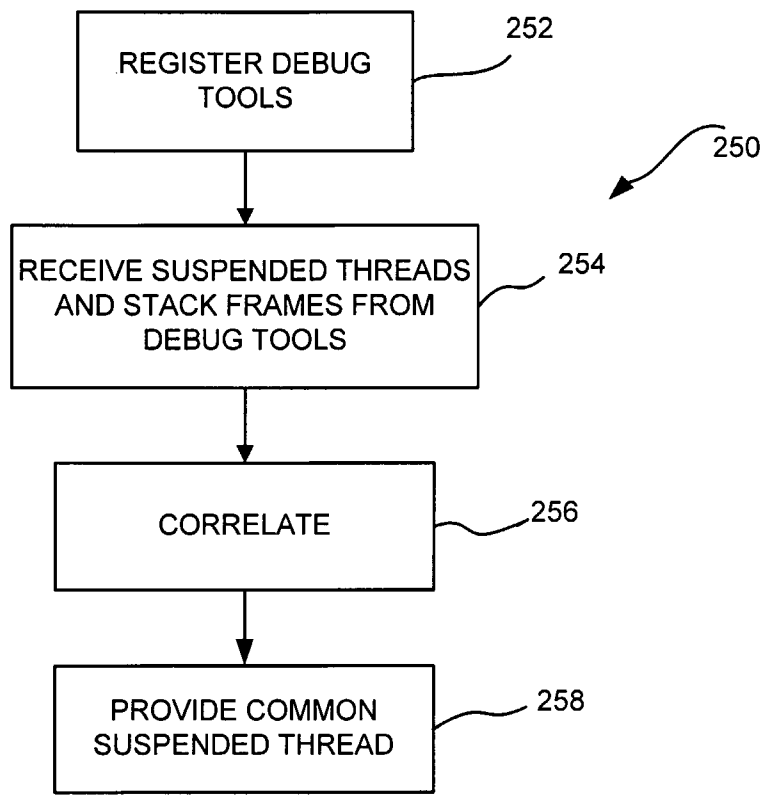
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK,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

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

Published:
— *with international search report*

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

(54) **Title:** INTEGRATING DIFFERENT PROGRAMMING LANGUAGE DEBUG TOOLS FOR OBSERVING THREAD EXECUTION



(57) **Abstract:** Software developers working on multi-language systems with various debug tools (BPEL, AE, Java, etc.) can use a common debug adaptor (CDA) apparatus. The CDA implements a method of debugging in a multi-computer program language environment. The method includes registering various debug tools associated with different programming languages in the multicomputer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment. The method can further include receiving the suspended threads and stack frames from the plurality of debug tools. The method can further include correlating the received suspended threads and stack frames under a common suspended thread; and providing the common suspended thread in a debug view. Such a method can have a number of attributes intended to assist developers facing debugging problems in multi-language systems.

WO 2007/028227 A1

COMMON DEBUG ADAPTOR

Field of the Invention

The present invention relates to the debugging of software and various software components and more particularly to systems and methods for managing various debugging tools in integrated development environments.

Background

A debugger (or debugging tool) is a computer program that is used to debug (and in some cases test or optimize) other computer programs. When the computer program crashes, the debugger generally shows the offending position or location in the original code (for source-level debuggers). A crash occurs when the computer program cannot continue because of a programming bug. Typically, debuggers offer functions such as running a program step by step (single stepping), stopping (breaking) at a particular event and tracking the values of variables.

Many software systems (multi-threaded or distributed systems) are written in more than one programming language. For example, a system may be implemented in Java™ with another language running on top of Java that needs to be debugged. Further difficulties are presented due to the lack of standardization in terms of internal structures, such as stack frames, between different programming languages.

For example, in business integration tooling, there are often different language debugger tools running on different debug runtime/test environments. In the tooling, each debugger would have its own way to show its suspended thread and stack frame. Each debugger may not know the existence of the other one. These situations create significant difficulties for software developers attempting to debug these systems.

Consequently, there exists an ongoing need for debugging technology that facilitates efficient programming by way of language, debug tool, host application and operating environment independence.

Summary

Methods and systems for use in a debugging environment that can be used by software developers working on multi-computer program language environments are described. The techniques used create a common debug adaptor that manages various debug tools (each associated with a different computer language) to provide merged information from debug events to enable debugging between multiple languages in a multi-language environment.

Certain exemplary embodiments can provide a method of debugging in a multi-computer program language environment, the method comprising: registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment; receiving the suspended threads and stack frames from the plurality of debug tools; correlating the received suspended threads and stack frames under a common suspended thread; and providing the common suspended thread in a debug view.

Certain exemplary embodiments can provide a system for debugging in a multi-computer program language environment, the system comprising: a registry module for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment; a correlation module for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and a user interface module for providing the common suspended thread in a debug view.

Certain exemplary embodiments can provide a computer program product for debugging in a multi-computer program language environment, the product comprising: a registry mechanism that is executable on the computer program for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in

response to a debug event in the multi-computer program language environment; a correlation mechanism that is executable on the computer program for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and a user interface mechanism that is executable on the computer program for providing the common suspended thread.

Certain exemplary embodiments can provide an apparatus for debugging in a multi-computer program language environment, the apparatus comprising: a processor; a memory coupled to the processor; a computer program residing in the memory; a common debug adaptor residing in the memory and executed by the processor; the common debug adaptor comprising: a registry module for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment; a correlation module for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and a user interface module for providing the common suspended thread.

Brief Description of the Drawings

Fig. 1 illustrates an example of a computing system environment in block diagram form used to implement common debug adaptor technology according to various embodiments of the present invention;

Fig. 2A illustrates an example of a common debug adaptor environment in block diagram form;

Fig. 2B illustrates an overview of common debug adaptor operational steps in flow diagram form;

Fig. 3 illustrates further details of common debug adaptor operational steps in flow diagram form;

Fig. 4 illustrates a common debug adaptor architecture overview and operational example in block diagram form; and

Fig. 5 illustrates an example debug view showing a common suspended thread (merged stack frames).

Detailed Description

Computing System Environment – Fig. 1

Fig. 1 illustrates an example of a computing system environment 100 in which embodiments of the present invention can be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the various embodiments described. Examples of other computing system environments or configurations that may be suitable include: a general purpose Personal Computer (PC); a hand-held or lap top computer; multi-processor based systems; microprocessor based systems; programmable consumer electronics; network computers, minicomputers, mainframe computers and distributed computing environments.

The computing system environment 100 includes a general purpose computing device 102. Components of the computing device 102 include, but are not limited to, a processing unit 104, an input/output interface 106, a system memory 108, and a system bus 110. The system bus 110 communicatively connects the aforementioned components and numerous other (not shown) cooperatively interactive components. The input/output interface 106 interacts with external components through an input/output unit 112 (which can include keyboard, mouse-type controllers, monitors, media readers/writers and the like). The system memory 108 instantiates various components and operations of a common debug adaptor 202 according to embodiments of the present invention described in detail in subsequent figures. The computing system environment 100 serves as an apparatus for performing common debug adaptor processes.

Architectural and Process Overview – Figs. 2A and 2B

Some embodiments will include one or more of the functional components/modules/mechanisms or process steps described. Any particular embodiment may not require all of the components or steps described, may use additional components or steps, or may use an entirely different organization without change the functionality, scope or spirit.

Fig. 2A provides a schematic representation of a debug environment 200, which includes a number of different language debug tools: debug tool A 204, debug tool B 206, and debug tool C 208. Each debug tool 204-208 has its own way to show suspended threads and stack frames. Further, each debug tool 204-208 need not be aware of the existence of any other tool. This arrangement is common in business integration tooling environments where multiple computer programming languages are used. Fig. 2B provides a flow chart of a process 250 used to manage these various debug tools 204-208. A suspended thread is a virtual process that is paused for debugging purposes. Stack frames are user interface representations of the calling stacks in a suspended thread. For example, if a user has added a breakpoint at a line in a Java program, when the breakpoint is hit, the user would see a suspended Java thread with several Java stack frames under it. The Java stack frames would show how the line with the breakpoint is called through the various Java classes and methods.

With reference to Figs. 2A and 2B, the debug tools 204-208 interact with the common debug adaptor (CDA) 202 to register (step 252) the individual debug tools, through a registry module 210; receive (step 254) suspended threads and stack frames from the different debug tools 204-208 (in response to a debug event); correlate (step 256) the received suspended threads and stack frames, through a correlation module 212; and provide (step 258) a common suspended thread (i.e., merge the stack frames and provide the relevant suspended threads for use by an operator through a user interface (UI) module 214).

Debug Tools 204, 206, 208

Each debug tool 204-208 represents an external tool written to run on its own runtime (i.e., environment/software under test). Each debug tool 204-208 is identified by an identifier (pluginID). Each debug tool 204-208 can also (a) identify a server (not shown) it is debugging (EngineID); (b) identify an original instance it is running from (Global Instance ID - GIID); and (c) identify a thread (a sequence of instructions) it is running at (ThreadID).

A debug view (native to each debug tool 204-208 and not shown in the drawings) would show (a) a launcher, (b) a debug target, and (c) a thread and stack frame at which its breakpoint is suspended.

Registry Module 210

The registry module 210 registers each debug tool 204-208 to the common debug adaptor 202 by receiving pluginID type information through a receiving mechanism 216. In particular, each participating debug tool 204-208 extends the common debug adapter 202 extension point. An extension point is similar to a plug-in and adds an extra feature to a programming language or system using well established techniques in the field of the invention. Each debug tool 204-208 creates and returns a debug target, suspended threads and stack frames to the common debug adapter 202 for handling in response to one or more debug events originating from environment/software under test.

Correlation Module 212

The correlation module 212 accommodates non-Java and Java debug tools 204-208 and enables various types of stack frames to be correlated/merged. In general, the individual debug tools 204-208 routes debug events for mixed stack frame handling to the correlation module 212 of the CDA 202. A debug event is a run-time altering condition, such as a breakpoint, that is set in an application by a debugging environment user and managed by an active debugging environment that controls the running of an application. Debug events are defined by the individual debug tool 204-208.

The correlation module 212 groups the various suspended threads and stack frames from the debug tools 204-208 using a grouping mechanism 218. The grouping is determined by one or more of the EngineID, the GIID and the ThreadID described above. In one particular example, the correlation module 212: (i) creates a launcher for each EngineID; (ii) creates a debug target for each GIID; and (iii) groups debug events from the various debug tools 204-208 with the same ThreadID into stack frames under a common suspended thread.

User Interface 214

The UI module 214 controls the display and management of information, such as the common suspended thread, that is provided in a debug view (example provided in Fig.5). In particular, if a user performs resume, step over, step into or step return on a target stack frame (under the common suspended frame) of the CDA 202, all these actions would be delegated to one of the

debug tools 204-208. Delegation is performed by a delegating mechanism 220 and is defined as follows: (i) the CDA 202 is notified of a user action to resume/step over/step into/step return; (ii) the CDA 202 identifies the debug tool 204-208 that corresponds to the selected stack frame and (iii) calls the same action on the debug tool 204-208 identified in step (ii). The UI module 212 defines various debug view elements (such as CDADebugTarget, CDAThread, CDAStackframe) handles user interface based actions and delegates certain actions (resume, step over, step into etc.) to an individual debug tool 204-208.

Process/System Example- Figs. 3 and 4

A process 300 (Fig. 3) and a system 400 (Fig. 4) of managing multiple debug tools 204-208 according to various embodiments will be described in conjunction with Figs. 3 and 4.

For the purpose of illustration in Fig. 4, the debug tools 204-208 are designated as language specific tools: tool 204 is a Business Process Execution Language (BPEL) debugger; tool 206 is a Business State Machine (BSM) debugger; and tool 208 is a transforms debugger. A Java debug manager (JDM) 402 is also illustrated. The JDM 402 acts like another debug tool to the CDA 202. The JDM 402 filters debug events, queries runtime execution for Java thread information and routes source debug information to the CDA 202 as described in more detail below.

BPEL is an XML-based language for standardizing business processes in a distributed or grid computing environment that enables separate businesses to interconnect their applications and share data. Platform-independent BPEL allows enterprises to keep internal business protocols separate from cross-enterprise protocols so that internal processes can be changed without affecting the exchange of data from enterprise to enterprise. A BPEL document, for example, keeps track of all the business processes that are connected to a transaction and ensures that the processes are executed in the correct order through the automation of messages.

The BSM and transforms debuggers, 206 and 208 respectively, are examples of other debuggers that participate through the registry module 210 of the CDA 202 and are known in the art.

The JDM 402 is mainly used to filter Java debug events that would be relevant. The JDM 402 forwards a current Java debug event (JDE) to the CDA 202 and determines whether the JDE is from any of the debug tools 204-208. If so, the JDM 402 would receive correlation information on the JDE to enable delegation (as discussed above) to one of the debug tools 204 to 208 to handle the JDE and return the corresponding stack frames.

With reference to Figs. 3 and 4, when debug events are sent 302 from a test environment 404 (such as a multi-computer program language environment) through a communication gateway 406 each debug event is analyzed and routed 304 to one of the debug tools 204-208. The analysis and routing step 304 is based on the pluginID information of the debug tool 204-208 associated with each event as described above. After routing to the appropriate debug tool (one of tools 204-208) the CDA 202 is called and provided with information vectors 306 (details of which are provided in Table A).

TABLE A

INFORMATION VECTOR	ITEMS
(1) Instance	(a) Engine identification (Engine ID) – used as a key to keep track of its related server (b) Global instance identification (GIID) – a key indicator for a debug target (c) Thread identification – a key indicator for a virtual thread (1a-c) can be used by the CDA 202 to merge stack frames across different servers, debug targets and threads respectively (d) List of virtual threads running on a server – helps the CDA 202 and debuggers 204-208 to clear obsolete threads
(2) Processed Stack Frame	(a) Processed debug adapters/tools (b) Plugin identification of the debug adapter/tool (c) Individual debug adapter identification

INFORMATION VECTOR	ITEMS
	(d) Array of stack frames (2a-d) contains results after the individual tools 204-208 process runtime information and create their own debug target, thread and stack frame.
(3) Unprocessed debug runtime event	(a) List of remaining unprocessed runtime events - contains the information from the runtime (test environment 404) that is to be processed by one of the debug tools 204-208

The correlation module 212 of the CDA 202 analyzes data from the information vectors 308 and delegates event tasks 310 to the debug tools 204-208 as described above. The analysis step 308 involves grouping suspended threads and stack frames into a common stack frame based on one or more of the EngineID, the GIID and the ThreadID as discussed in conjunction with the correlation module 212 of Fig. 2.

If all the debug events received (at step 302) from the test environment 404 have not been processed, as determined at step 312, then processing returns to step 304. If all the debug events received (at step 302) from the test environment 404 have been processed, as determined at step 312, processing continues to step 314 to construct a debug tree, which includes the common suspended thread.

The UI module 214 constructs the debug tree in a debug view by showing a launcher with EngineID information. The launcher includes a debug target with GIID as the identifier. The debug tree also includes various threads with the ThreadIDs and corresponding stack frames under each ThreadID (see Fig. 5 for an example).

When considering the JDM 402 (Fig. 4), the CDA 202 can also merge a Java stack frame with other stack frames (for the debug tools 204-208) given a Java breakpoint suspended in a Java thread. The test environment 404 can handle a Java match thread query to identify a suspended Java thread that originates from business integration components (i.e., part of the test

environment 404). When a Java debug event arrives, from the test environment 404, the JDM 402 will try to filter the event. In particular, the JDM 402 will send a query to the test environment 404 to determine if the thread originated from one of the debug tools 204-208.

The result of the query is returned to the JDM 402 which combines the event with other debug information (e.g., information vectors) and route the information to the CDA 202 for handling. The CDA 202 would in-turn delegate any required handling back to one of the debug tools 204-208 as required. The individual debug tools 204-208 handles the Java mixed stack frame by identifying where the Java code is called from. The results from the debug tools 204-208 are rerouted to the CDA 202 for further mixed stack frame handling if required. The CDA 202 then provides the complete merged stack frame (as the common suspended stack frame) in the debug view (through the UI module 214) as previously discussed.

Debug tree example- Fig. 5

Assuming a server (not shown) in the test environment 404 (e.g, business integration tool) is started in a debug mode and appropriate breakpoints have been added, various breakpoints would occur. If a breakpoint occurs at a first component (in the test environment 404) which is called by a second component (in the test environment 404) a merged stack frame will be shown under the wiring thread in a debug tree 500 shown in Fig. 5. In this example, a Wiring editor calls Adaptive Entity (AE), which then calls Business Rule (BR). The merged stack frame has an inner most layer as a BR stack frame (GuardBR), then followed by an AE stack frame (AEExample) and then the Wiring stack frame (WiringExample) as an outermost layer.

The debug tree 500 illustrates that all stack frames from the debug tools (e.g., tools 204-208) have the same GIID (not illustrated but used internally) and ThreadID (e.g, 1234). If any stack frame has a different ThreadID, those stack frames would appear under another thread in the debug target. The UI module 214 can progressively disclose such that only relevant debug elements are shown in the debug tree 500. In the example of Fig. 5, the debug targets and threads for AEExample and GuardBR are hidden. The server Java threads are also hidden as they are not considered relevant to the business integration tool of this example.

In summary, embodiments of the common debug adapter 202 are generic for different debug tools 204-208 and test environments 404 (i.e., runtimes). In practical implementation, each debug tool 204-208 implements an extension point to integrate with the system 400.

The detailed description does not limit the implementation of the embodiments of the present invention to any particular computer programming language. The computer program product may be implemented in many computer programming languages provided that the OS (Operating System) provides the facilities that may support the requirements of the computer program product. An exemplary embodiment of the present invention can be implemented in the C or C++ computer programming language, or may be implemented in any other mix of supported programming languages. Any limitations presented would be a result of a particular type of operating system, computer programming language, or database management system and would not be a limitation of the embodiments of the present invention described herein.

CLAIMS:

1. A method of debugging in a multi-computer program language environment, the method comprising:

registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment;

receiving the suspended threads and stack frames from the plurality of debug tools;

correlating the received suspended threads and stack frames under a common suspended thread; and

providing the common suspended thread in a debug view.

2. The method of claim 1, wherein each one of the plurality of debug tools includes (i) a debug tool identifier; (ii) a server under debug identifier; (iii) an instance identifier; and (iv) a thread identifier.

3. The method of claim 2, wherein registering includes receiving the debug tool identifier from each one of the debug tools.

4. The method of claim 2, wherein correlating includes grouping the suspended threads and stack frames from the plurality of debug tools based on at least one of the server under debug identifier, the instance identifier and the thread identifier.

5. The method of claim 2, wherein correlating includes: creating a launcher for each server under debug identifier; creating a debug target for each instance identifier; and grouping debug events from the plurality of debug tools having identical thread identifiers into stack frames under the common suspended thread.

6. The method of claim 2, further comprising delegating a user instruction to one of the plurality of debug tools.

7. The method of claim 6, wherein the user instruction includes one of the following actions: resume, step over, step into and step return.

8. The method of claim 7, wherein delegating includes: receiving the user instruction; identifying one of the plurality of debug tools that corresponds to a selected stack frame; and calling for execution of an identical action on the identified debug tool.

9. A system for debugging in a multi-computer program language environment, the system comprising:

a registry module for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment;

a correlation module for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and

a user interface module for providing the common suspended thread in a debug view.

10. The system of claim 9, wherein each one of the plurality of debug tools includes (i) a debug tool identifier; (ii) a server under debug identifier; (iii) an instance identifier; and (iv) a thread identifier.

11. The system of claim 10, wherein the registry module includes a mechanism for receiving the debug tool identifier from each one of the debug tools.

12. The system of claim 10, wherein the correlation module includes a mechanism for grouping the suspended threads and stack frames from the plurality of debug tools based on at least one of the server under debug identifier; the instance identifier and the thread identifier.

13. The system of claim 10, wherein the correlation module includes a mechanism for: creating a launcher for each server under debug identifier; creating a debug target for each instance identifier; and grouping debug events from the plurality of debug tools having identical thread identifiers into stack frames under the common suspended thread.

14. The system of claim 10, wherein the user interface module includes a mechanism for delegating a user instruction to one of the plurality of debug tools.

15. The system of claim 14, wherein the user instruction includes one of the following actions: resume, step over, step into and step return.

16. The system of claim 14, wherein the mechanism for delegating includes a mechanism for: receiving the user instruction; identifying one of the plurality of debug tools that corresponds to a selected stack frame; and calling for execution of an identical action on the identified debug tool.

17. A computer program product for debugging in a multi-computer program language environment, the product comprising:

a registry mechanism that is executable on the computer program for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment;

a correlation mechanism that is executable on the computer program for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and

a user interface mechanism that is executable on the computer program for providing the common suspended thread.

18. An apparatus for debugging in a multi-computer program language environment, the apparatus comprising:

a processor;

a memory coupled to the processor;

a computer program residing in the memory;

a common debug adaptor residing in the memory and executed by the processor;

the common debug adaptor comprising:

a registry module for registering a plurality of debug tools associated with different programming languages in the multi-computer program language environment, each one of the plurality of debug tools providing suspended threads and stack frames in response to a debug event in the multi-computer program language environment;

a correlation module for receiving the suspended threads and stack frames from the plurality of debug tools and correlating the received suspended threads and stack frames under a common suspended thread; and

a user interface module for providing the common suspended thread.

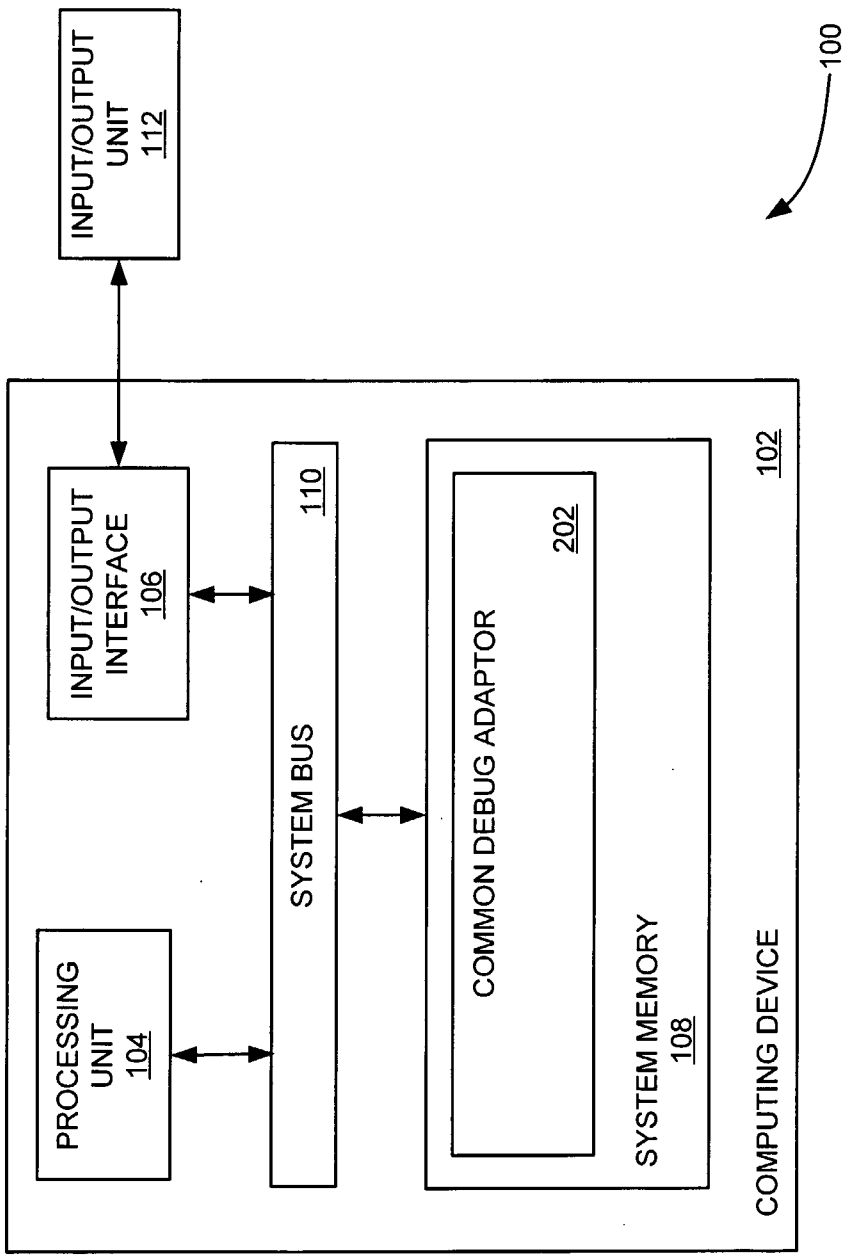

100
INPUT/OUTPUT UNIT 112
INPUT/OUTPUT INTERFACE 106
SYSTEM BUS 110
PROCESSING UNIT 104
COMMON DEBUG ADAPTOR 202
SYSTEM MEMORY 108
COMPUTING DEVICE 102

FIG. 1

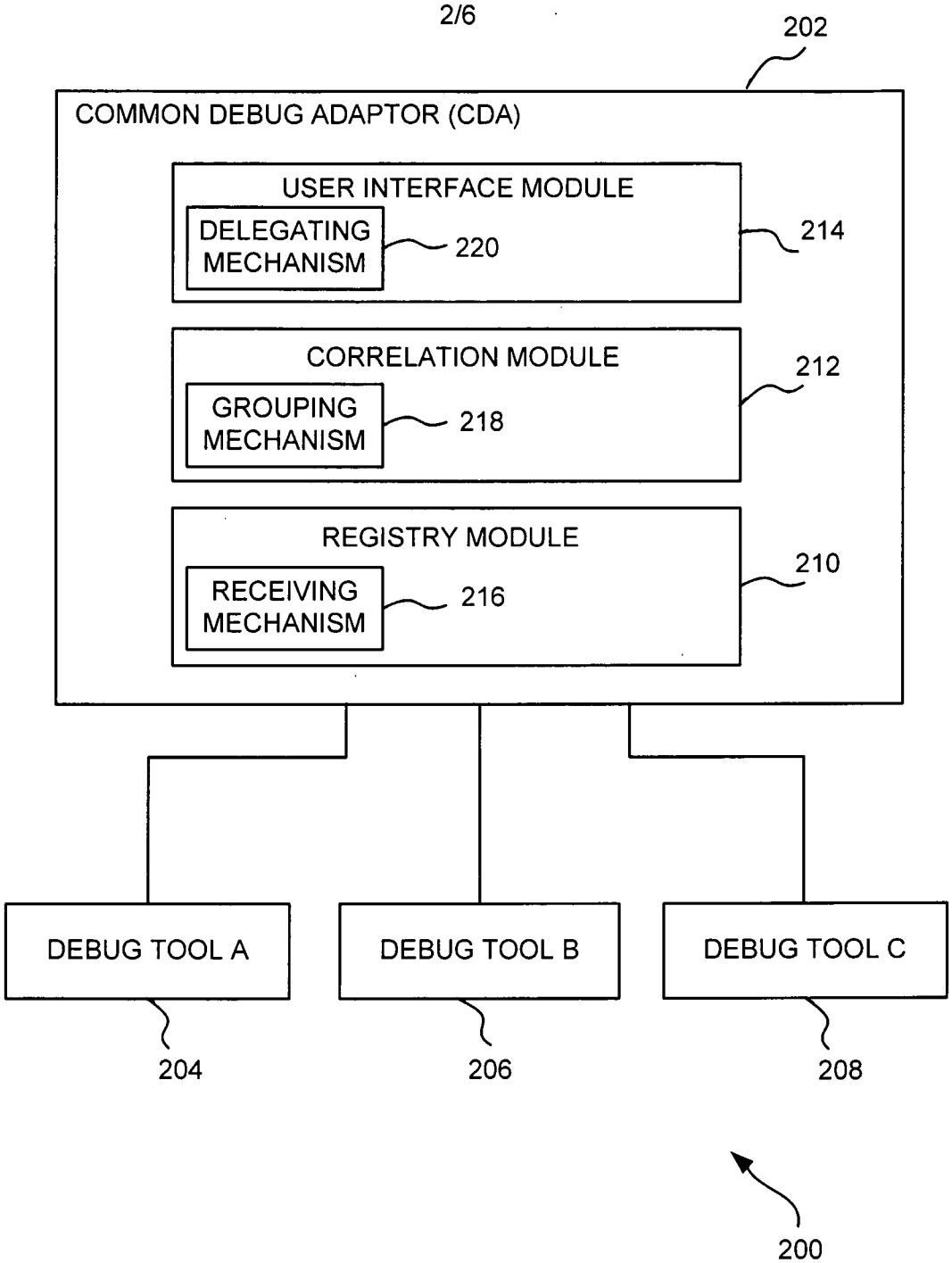

202
COMMON DEBUG ADAPTOR (CDA)
USER INTERFACE MODULE
DELEGATING MECHANISM
220
214
CORRELATION MODULE
GROUPING MECHANISM
218
212
REGISTRY MODULE
RECEIVING MECHANISM
216
210
DEBUG TOOL A
DEBUG TOOL B
DEBUG TOOL C
204
206
208
200

FIG. 2A

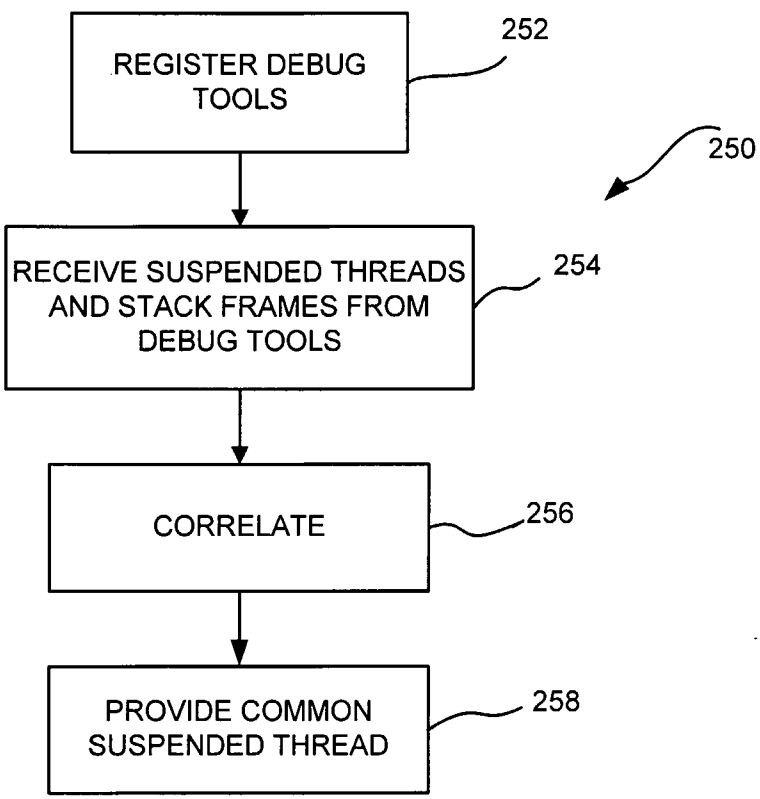

REGISTER DEBUG TOOLS
252
250
RECEIVE SUSPENDED THREADS AND STACK FRAMES FROM DEBUG TOOLS
254
CORRELATE
256
PROVIDE COMMON SUSPENDED THREAD
258

FIG. 2B

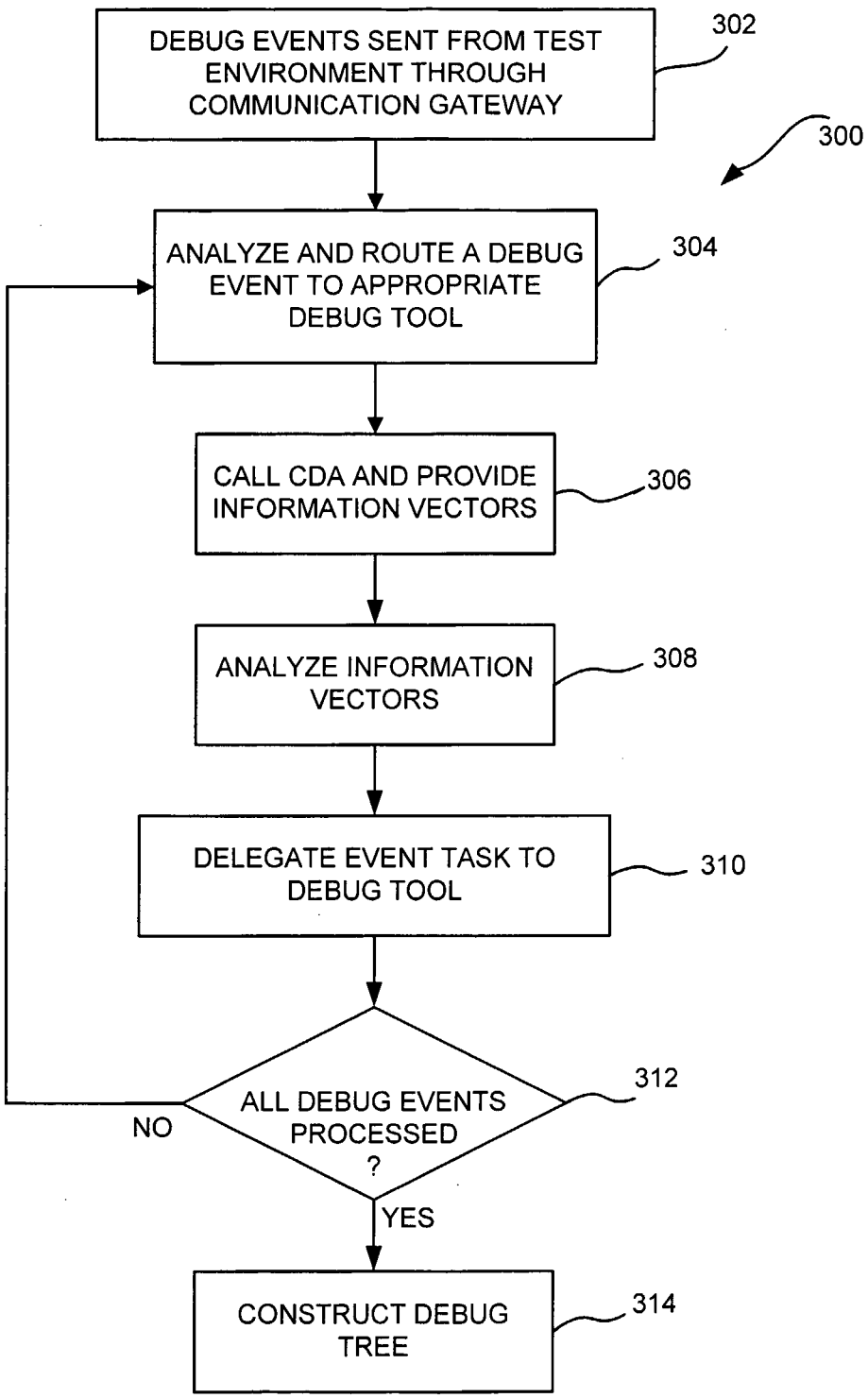

DEBUG EVENTS SENT FROM TEST ENVIRONMENT THROUGH COMMUNICATION GATEWAY
302
300
ANALYZE AND ROUTE A DEBUG EVENT TO APPROPRIATE DEBUG TOOL
304
CALL CDA AND PROVIDE INFORMATION VECTORS
306
ANALYZE INFORMATION VECTORS
308
DELEGATE EVENT TASK TO DEBUG TOOL
310
ALL DEBUG EVENTS PROCESSED ?
312
NO
YES
CONSTRUCT DEBUG TREE
314

FIG. 3

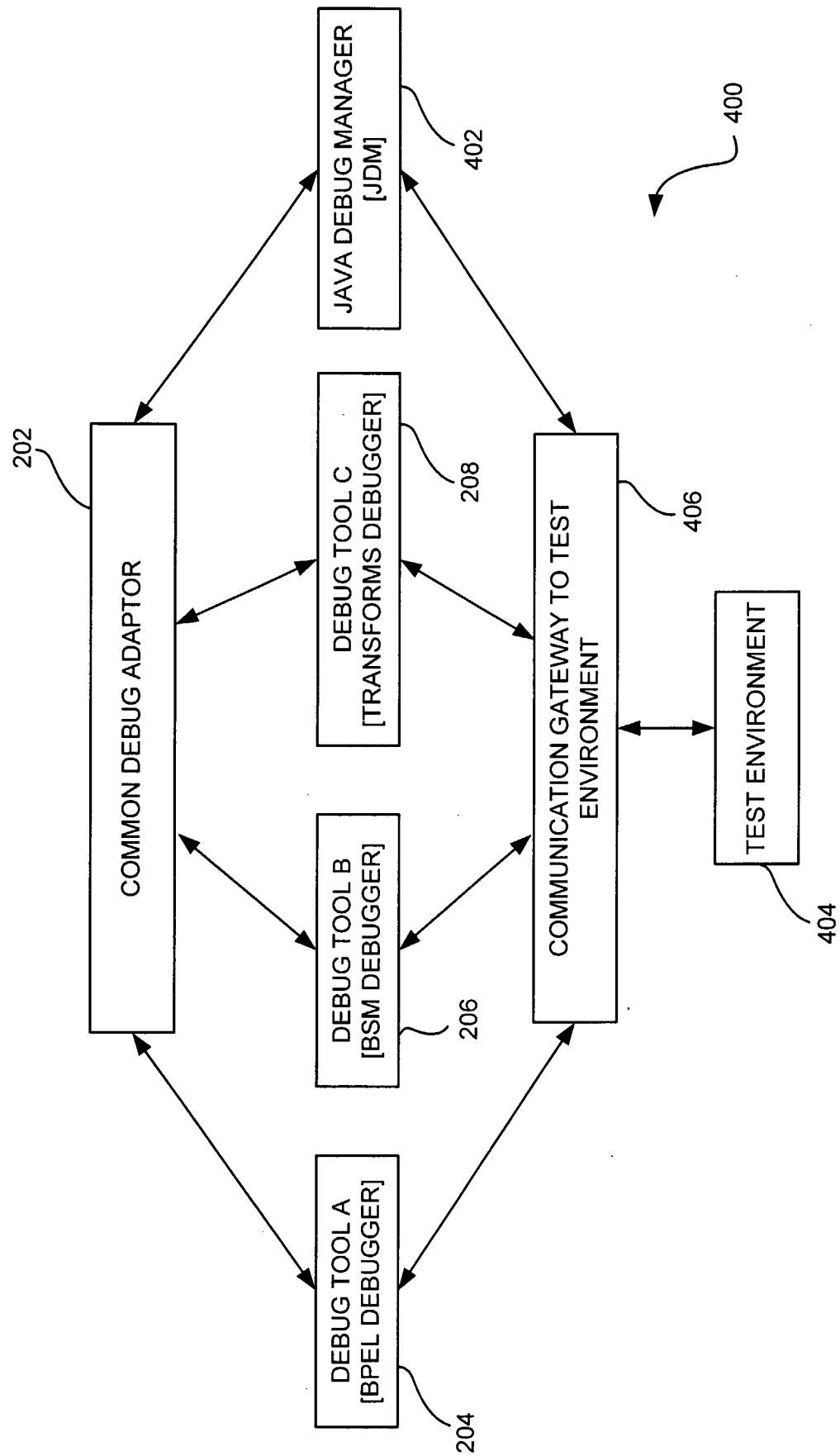

JAVA DEBUG MANAGER [JDM]
402
400
202
COMMON DEBUG ADAPTOR
DEBUG TOOL C [TRANSFORMS DEBUGGER]
208
406
COMMUNICATION GATEWAY TO TEST ENVIRONMENT
TEST ENVIRONMENT
404
DEBUG TOOL B [BSM DEBUGGER]
206
DEBUG TOOL A [BPEL DEBUGGER]
204

FIG. 4

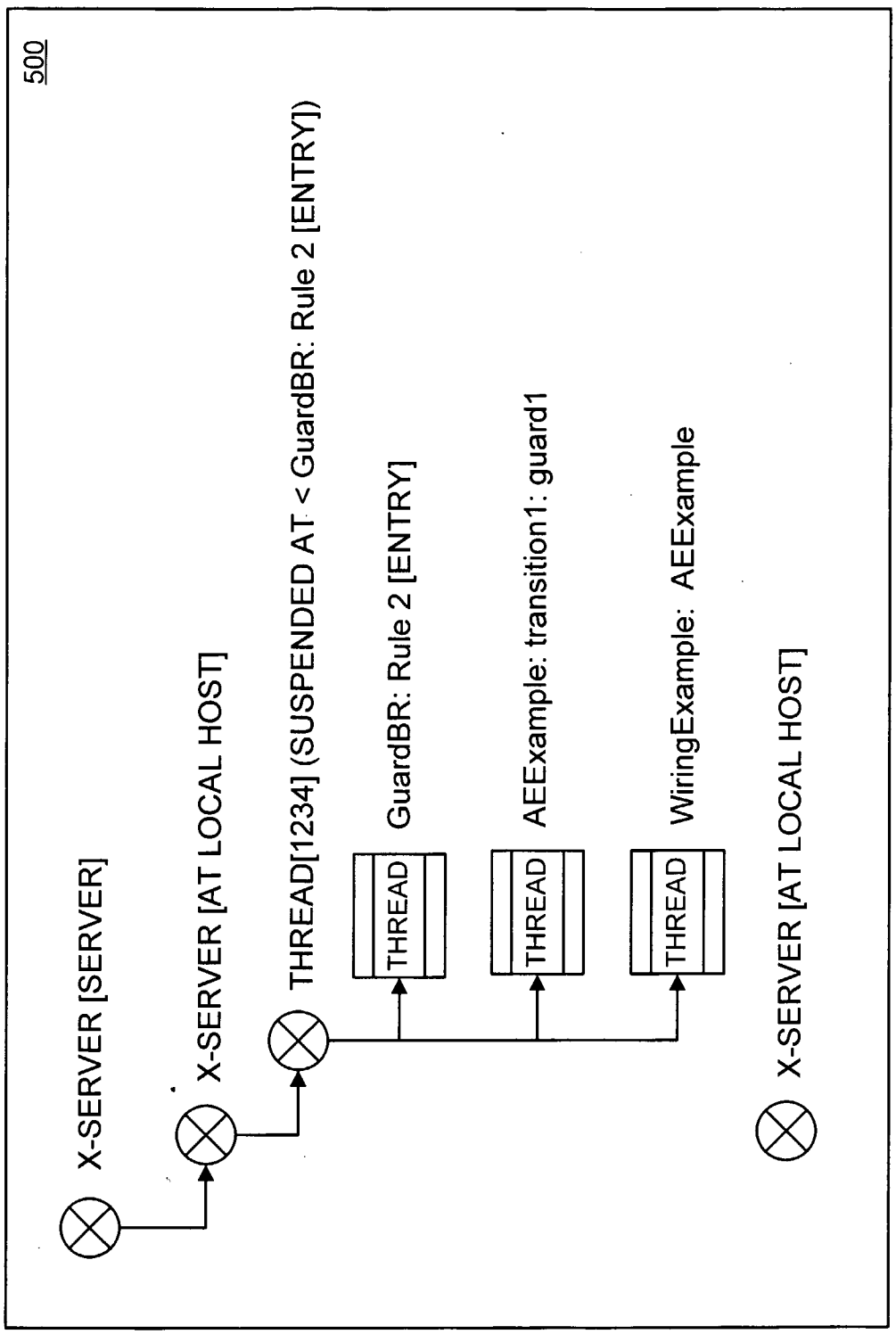

500
X-SERVER [SERVER]
X-SERVER [AT LOCAL HOST]
THREAD[1234] (SUSPENDED AT < GuardBR: Rule 2 [ENTRY])
THREAD
GuardBR: Rule 2 [ENTRY]
THREAD
AEExample: transition1: guard1
THREAD
WiringExample: AEExample
X-SERVER [AT LOCAL HOST]

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2005/001380

A. CLASSIFICATION OF SUBJECT MATTER
IPC: ***G06F 11/36*** (2006.01), ***G06F 9/44*** (2006.01)

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-11/36, G06F-9/44

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

Electronic database(s) consulted during the international search (name of database(s) and, where practicable, search terms used)
WEST, QPAT, DELPHION, IEEE, CANADIAN PATENT DATABASE (keywords included: debug, environment, thread, languages, tools, modules, interface, connect, plug, stack frame, trace, instance, step, register, name, identify, server, common thread, correlate, merge, synchronize, view and display)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6,353,923 (BOGLE PHILIP (US); KATZENBERGER GARRY (US); MCKELVIE SAMUEL (US); WELLAND ROBERT (US);) 5 March 2002 (05-03-2002) <Abstract, Column 3, line 20 to Column 4, line 35 Column 7, lines 18 to 45 Column 10, lines 23 to 34 Column 10, line 47 to Column 12, line 61 Column 13, lines 5 to 10 and lines 54 to 59 Column 14, lines 25 to 28 and lines 42 to 46>	1-18
X	US 6,480,818 (ALVERSON GAIL (US); SMITH BURTON (US); KAPLAN LAURENCE (US); NIEHAUS MARK (US);) 12 November 2002 (12-11-2002) <Abstract, Column 7, line 1 to Column 12, line 67>	1-18

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

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
5 December 2005 (05-12-2005)	03 February 2006 (03-02-2006)

Name and mailing address of the ISA/CA	Authorized officer
Canadian Intellectual Property Office Place du Portage I, C114 - 1st Floor, Box PCT 50 Victoria Street Gatineau, Quebec K1A 0C9 Facsimile No.: 001(819)953-2476	Niall Zelem (819) 953-9814

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2005/001380

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 6,721,941 (MORSHED FAROKH (US); MEAGHER ROBERT (US);) 13 April 2004 (13-04-2004) <Abstract, Column 37, lines 41 to 67 Column 49, lines 51 to 67>	1-18
A	EP 1,220,099 A2 (TABE TETSUYA (JP); NAGAO YOSHIHIRO (JP);) 03 July 2002 (03-07-2002) <ENTIRE DOCUMENT>	1-18
A	WO 01/69390 A2 (PENNELLO THOMAS (US); DAVIS HENRY (US); GAZDZINSKI ROBERT (US);) 20 September 2001 (20-09-2001) <ENTIRE DOCUMENT>	1-18
A	US 5,953,530 (RISHI ALOK (US); MASAMITSU JON (US);) 14 September 1999 (14-09-1999) <ENTIRE DOCUMENT>	1-18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CA2005/001380

Patent Document Cited in Search Report	Publication Date	Patent Family Member(s)	Publication Date
US6353923	05-03-2002	US6275868 B1	14-08-2001
		US6353923 B1	05-03-2002
		US2001005852 A1	28-06-2001
US6480818	12-11-2002	US6480818 B1	12-11-2002
		US6848097 B1	25-01-2005
		US2005034024 A1	10-02-2005
US6721941	13-04-2004	AT222007T T	15-08-2002
		AU3366400 A	04-09-2000
		AU7796498 A	08-12-1998
		CA2289024 A1	19-11-1998
		DE69807088D D1	12-09-2002
		EP1012722 A1	28-06-2000
		EP1155367 A1	21-11-2001
		US5987249 A	16-11-1999
		US6016466 A	18-01-2000
		US6186677 B1	13-02-2001
		US6314558 B1	06-11-2001
		US6332213 B1	18-12-2001
		US6643842 B2	04-11-2003
		US6701519 B1	02-03-2004
		US6721941 B1	13-04-2004
		US6760903 B1	06-07-2004
		US2001047510 A1	29-11-2001
		US2004133882 A1	08-07-2004
		WO0049502 A1	24-08-2000
		WO9852122 A1	19-11-1998
EP1220099	03-07-2002	EP1220099 A2	03-07-2002
		JP2002202899 A	19-07-2002
		US2002087952 A1	04-07-2002
WO01/69390	20-09-2001	AU5580801 A	24-09-2001
		US2001056341 A1	27-12-2001
		WO0169390 A2	20-09-2001
US5953530	14-09-1999	EP0729097 A1	28-08-1996
		JP9022370 A	21-01-1997
		US5953530 A	14-09-1999