BESCHWERDEKAMMERN DES EUROPÄISCHEN PATENTAMTS | **BOARDS OF APPEAL OF THE EUROPEAN PATENT OFFICE** | **CHAMBRES DE RECOURS DE L'OFFICE EUROPÉEN DES BREVETS**

Internal distribution code:

(A) [-] Publication in OJ
(B) [-] To Chairmen and Members
(C) [-] To Chairmen
(D) [X] No distribution

Datasheet for the decision of 31 May 2022

Case Number:	T 2341/16 - 3.5.06
Application Number:	05858788.2
Publication Number:	1929400
IPC:	G06F9/46, G06F9/455
Language of the proceedings:	EN

Title of invention:
PROCESSING EVENTS FOR CONCURRENT TASKS IN A VIRTUAL MACHINE

Applicant:
Oracle America, Inc.

Headword:
Processing events/ORACLE

Relevant legal provisions:
EPC 1973 Art. 56, 84, 116(1)

Keyword:
Inventive step - (no)
Claims - clarity (no)
Oral proceedings - request to hold as video conference (not allowed)

This datasheet is not part of the Decision
It can be changed at any time and without notice

Decisions cited:

Catchword:

This datasheet is not part of the Decisior
It can be changed at any time and without notic

Europäisches Patentamt
European Patent Office
Office européen des brevets

Beschwerdekammern

Boards of Appeal

Chambres de recours

Boards of Appeal of the
European Patent Office
Richard-Reitzner-Allee 8
85540 Haar
GERMANY
Tel. +49 (0)89 2399-0
Fax +49 (0)89 2399-4465

Case Number: T 2341/16 - 3.5.06

DECISION
of Technical Board of Appeal 3.5.06
of 31 May 2022

Appellant:
(Applicant)

Oracle America, Inc.
500 Oracle Parkway
Redwood City, CA 94065 (US)

Representative:

D Young & Co LLP
120 Holborn
London EC1N 2DY (GB)

Decision under appeal:

Decision of the Examining Division of the European Patent Office posted on 18 May 2016 refusing European patent application No. 05858788.2 pursuant to Article 97(2) EPC.

Composition of the Board:

Chairman M. Müller
Members: A. Teale
K. Kerber-Zubrzycka

Summary of Facts and Submissions

I. This is an appeal against the decision, dispatched with reasons on 18 May 2016, refusing European patent application No. 05 858 788.2 on the basis that *inter alia* claim 1 according to the main and first and second auxiliary requests did not satisfy Article 56 EPC regarding inventive step in view of the following document:

D1: Balfanz D. and Gong L., "Experience with Secure Multi-Processing in Java", XP002377379, Technical Report no. 560-97, 29 September 1997, Princeton University, USA.

II. A notice of appeal and the appeal fee were received on 27 May and 6 July 2016, respectively. The appellant requested that the application be granted.

III. With a statement of grounds of appeal, received on 28 September 2016, the appellant submitted amended claims according to a new first auxiliary request. The appellant requested that the board set aside the decision on the basis of the main request in the decision (present main request), said new first auxiliary request, the first auxiliary request in the decision (present second auxiliary request) and the second auxiliary request in the decision (present third auxiliary request). The appellant also requested oral proceedings should the main request not be allowed.

IV. In an annex to a summons to oral proceedings the board set out its preliminary opinion on the appeal that it had doubts regarding the inventive step, Article 56 EPC 1973, of the subject-matter of the independent method and device claims of all requests in view of D1. In claims 1 and 10 of the first auxiliary request there was no indication of the role played by retrieving the task ID of the task currently in the foreground and thus uncertainty and a lack of clarity, Article 84 EPC 1973, as to the technical effect of this feature and whether it could contribute to inventive step, Article 56 EPC 1973. Moreover the expression "Java compliant" as used in all requests had a meaning which changed with time, thus making it unclear, Article 84 EPC 1973.

V. The appellant did not file any amendments or substantive arguments in response to the board's preliminary opinion.

VI. In a letter received on 1 April 2022 the appellant referred to the Covid-19 infection rates in the United Kingdom and in Munich and requested that the oral proceedings be held as a video conference.

VII. In a communication dated 11 April 2022 the board noted that there seemed to no longer be any official limitations or impairments affecting the appellant's ability to attend oral proceedings in person. Hence the board was not convinced by the appellant's arguments and still intended to hold the oral proceedings in person.

VIII. On 13 May 2022 the appellant's representative indicated that the appellant would not be represented at the oral proceedings. The board then cancelled the oral proceedings.

IX. The application is being considered in the following form:

Description (all requests):
pages 1, 2, 4 and 6 to 14, as originally filed, and pages 3, 3a and 5, received on 24 February 2010.

Claims:
Main request: 1 to 12, received on 14 May 2012.
First auxiliary request: 1 to 10, received with the grounds of appeal.
Second auxiliary request: 1 to 6, received as first auxiliary request on 25 April 2016.
Third auxiliary request: 1 to 6, submitted as second auxiliary request during the oral proceedings before the examining division on 26 April 2016.

Drawings (all requests):
Pages 1/5 to 3/5 and 5/5, as originally filed, and page 4/5, received on 24 February 2010.

X. Claim 1 of the main request reads as follows:

"A method of processing native events by a virtual machine (112) that operates on a first platform (116), wherein said first platform is provided by a mobile device, and wherein said virtual machine concurrently supports a first and a second task on said first platform, said method comprising: receiving, by the virtual machine, a native event (El) that is associated with the first platform; determining, by said virtual machine, which one of said first and second tasks is a foreground task, wherein said foreground task is the only task that is displayed; and processing, by said foreground task, said native event."

XI. Claim 1 of the first auxiliary request differs from that according to the main request, editorial amendments aside, in the addition of the following features: storing the task ID of the task which is currently the foreground task, and retrieving, by the virtual machine, the task ID of the task currently in the foreground.

XII. Claim 1 of the second auxiliary request differs from that of the **main** request in the restriction of the virtual machine to a Java virtual machine and the addition of the following features:
- retrieving, by the Java virtual machine, the task ID of the task currently in the foreground;
- using, by the Java virtual machine, an object associated with the task ID to get a handle on an event queue and event handler for the foreground task;
- manipulating, by the Java virtual machine, said native event to be Java compliant by encapsulating the native event so that it can be represented as a Java event object;
- placing, by the Java virtual machine, the Java event object in the event queue of the foreground task;
- notifying, by the Java virtual machine, the event handler that the Java event object has been queued in the event queue, and
- the foreground task, when processing said native event, accessing the Java event object in the event queue.

XIII. Claim 1 of the third auxiliary request combines the amendments of the previous two requests.

XIV. Claims 4, 6 and 7 of the main and first auxiliary requests and claims 2 and 3 of the second and third auxiliary requests use the term "Java compliant".

Reasons for the Decision

1. Admissibility of the appeal

In view of the facts set out at points I to III above, the appeal fulfills the admissibility requirements under the EPC and is consequently admissible.

2. Summary of the invention

2.1 The invention relates to a virtual machine running on a platform, meaning a mobile device and its operating system (see [4]), the virtual machine concurrently supporting two tasks, for instance application programs; see figure 1B and amended page 3, lines 6 to 7. The virtual machine comprises a native event dispatcher which receives a native event associated with the platform and selects the "foreground" task, the only task being displayed (see page 8, lines 3 to 4, and figure 3; step 312), to process the native event; see figure 3.

2.2 Computers using the "World Wide Web" (WWW) protocol to communicate via the Internet can download and execute small applications called "applets". Applets are typically executed by a Java Virtual Machine (JVM), JVMs being available for a variety of platforms; see [2-3]. The JVM can be implemented in software by an interpreter for the JVM instruction set; see [4] and figure 1A. The JVM and its support libraries constitute a Java Runtime Environment (JRE).

2.3 The source code of programs (103) written in the Java programming language is structured in "classes" and "interfaces", referred to jointly as classes or class files. These are compiled by the Bytecode compiler

(103) to Bytecodes stored in the binary "Java class file" format; see figure 1A; 105 and [7]. The Bytecodes in the Java class file are then decoded and executed by the JVM.

2.4 According to amended page 3, lines 3 to 7, conventional virtual machines do not provide a multi-tasking environment, i.e. an environment for concurrently executing tasks, such as applets, for receiving input from the user or other sources; see page 3, lines 8 to 15. Some tasks, for instance an interactive game, require "event" processing (see [14]), for instance to receive user input from a keyboard. Such processing comprises delivering and handling external events to the appropriate task. In a virtual machine external events are typically generated, transmitted or processed by hardware or software platform components. Such platform-specific events are also referred to as "native" events. Conventional virtual machines cannot support two concurrent tasks if both require native event processing: see [15].

2.5 The application concerns enabling virtual machines to process native events for concurrent tasks in a multi-tasking environment; see page 3, last two lines, page 3a, last four lines, page 4, lines 23 to 27, and page 5, lines 4 to 14. This is achieved by an event dispatcher which delivers native events to the foreground task. As shown in figure 1B, the virtual machine 112 lies between the platform 116 and the application layer 114. Figure 1B shows two concurrent tasks (120, 122) running concurrently on the virtual machine. The event dispatcher 118 in the virtual machine receives native events (E1-4), for instance incoming data from a network device or keyboard, and routes the events to the foreground task for handling; see [26-28].

Figure 1C shows the steps carried out by the dispatcher to select a task and route an event to it; see [30].

2.6 Figure 2 shows a computing environment compliant with the Java Specification for Mobile Information Device Profile JST-37, for instance a phone or Personal Digital Assistant (PDA). A dispatcher 212 in the virtual machine 214, implemented as an event manager thread with wait-on-event 216 and event-dispatching logic 218, dispatches events (E1,E2) to two tasks (214, 216) running concurrently on the virtual machine; see [33]. The wait-on-event logic 216 causes the dispatcher to wait until an event is received, whilst the event dispatching logic selects the task to which the event is to be routed. Events arrive in the event-repository 220, a FIFO (First In First Out) queue, of the respective task and are processed by event processing logic 223 controlled by wait-on-event logic 222. The tasks can by associated with Mobile Information Device Profile (MIDP) applications 224, 226 (referred to as "midlets") in the application layer 206. In a mobile device user interactions with the foreground task generate native events which are processed by the foreground task; see [37]. The associated method steps are illustrated in figure 3; see [39-40]. Figure 3 shows the identification of each task by a task ID; see steps 312 and 324.

3. The board's understanding of the claims

3.1 The mobile device set out in the independent method claim 1 has a display and can thus, for instance, be a mobile phone; see [4], lines 2 to 5.

3.2 Two tasks, otherwise known as applications, run concurrently on the virtual machine, only one task, termed the "foreground" task, being displayed. In this context, the board understands the foreground task to have an associated window on the device display. The virtual machine determines which task is the foreground task, and native events from the platform are routed to and processed by said foreground task. The board understands user events to be, for instance, pressing a keyboard key or receiving data via a network; see [27]. Claim 10 of the main request further sets out the virtual machine comprising a native event dispatcher (118) for receiving a platform native event, deciding which of the two concurrent tasks is the foreground task and selecting it to process a native event.

3.3 In claim 1 of the first auxiliary request there is no indication of the role played by retrieving the task ID of the task currently in the foreground and thus the effect of this feature. Hence the board finds that the retrieval of the task ID cannot contribute to inventive step (see below).

4. Clarity, Article 84 EPC 1973

4.1 In claims 4, 6 and 7 of the main request, claims 4, 6 and 7 of the first auxiliary request, claims 1, 2, 3 and 6 of the second auxiliary request and claims 1, 2 and 6 of the third auxiliary request it is unclear what technical limitations on the native event are implied by it being "Java compliant".

4.2 It is firstly unclear how a native event being processed by a JVM - or a Java compliant MIDlet - cannot be "Java compliant". Secondly, given that the language definition of Java has developed over time, it is unclear what technical features are implied by a native event being compliant with "Java". The same objection applies to the features set out in the second and third auxiliary requests of encapsulating a native event so that it can be represented by a "Java event object".

5. Document D1

5.1 According to its abstract, D1 concerns using the Java platform, in practice a "Java Virtual Machine" (JVM), as a multi-processing, multi-user environment. D1 acknowledges that it was previously assumed that the JVM ran only one application, for instance a web browser fetching and executing Java "applets" from websites; see page 1, left column, lines 20 to 28. D1 considers extending the JVM so that it can run multiple applications.

5.2 D1 mentions "mobile code"; see page 1 in the paragraph bridging the two columns, on page 8, left column, line 16, on page 11, right column, line 16, and on page 12, left column, line 11. The board agrees with the appellant that this expression does not mean that the platform is a mobile device; it refers to the fact, which the person skilled in the art of computing would know, that the executable code, for instance the applets mentioned above, is "mobile", meaning that it may be transferred across the network to the platform where it is executed.

5.3 D1 also refers to "a small device" (see page 2, left column, lines 8 to 11), but the board does not understand "small" to directly and unambiguously disclose "mobile", since small computers, for instance embedded controllers, can have a permanent, fixed location.

5.4 Considering how a single application runs on a JVM (see page 3, section 3), a JVM has a number of threads, for instance for garbage collection and interpreting the Java bytecode of an application, in particular its main() method; see page 2, right column, lines 16 to 25, and page 3, figure 1. This involves linking classes as needed and performing necessary initializations before using a new class; see page 3, left column, lines 6 to 9. In the case of a Java application using AWT (Abstract Window Toolkit) components, a thread is started that dispatches events and calls to (call-back) code provided by the application; see page 3, left column, lines 45 to 47. The board understands this to mean that a user interface event, such as a mouse click in a window or a key press on the keyboard, is routed by the platform operating system to the JVM. The JVM creates an "AWT event object" and adds it to its event queue. A centralized event dispatcher thread of the JVM then takes the AWT event object from the event queue and calls the appropriate call-back method of the associated application whose window is currently being displayed; see page 3, figure 2 and section 3.2, "Event dispatching". Crucially, all callbacks are from a single event dispatcher thread; see page 4, left column, lines 2 to 13.

5.5 Section 4 concerns the features added to a single-application JVM to allow secure multi-processing. Of these, making system code multi-application-aware (page 5, feature 6) and multi-application-aware event dispatching (page 5, feature 7) are related and regarded as most relevant to the present case. According to feature 6, two users running different instances of the same program, such as a text editor, select the same GUI (Graphical User Interface) menu item "save file". The JVM event dispatcher thread must distinguish between the two cases, so that each user's file is stored in the respective user directory. According to feature 7, while every application receives the same information about the underlying operating system, different applications can have different definitions of the standard input and output streams.

5.6 According to section 5.1, an application is defined as a set of Java threads; see page 6, left column, line 17. Each application has its own name space and memory which is inaccessible to other applications; see page 6, left column, lines 41 to 45. As shown in figure 3, the threads belonging to the same application all have access to a shared application-wide state; see page 6, right column, lines 3 to 5. Whilst in the "one application" JVM the event dispatcher thread, which is not associated with any application and is started then an application opens a window, executes all callbacks from the operating system (see section 5.4, lines 1 to 5 and footnote 5), in the multi-processing case (see figure 4 on page 9), when an application opens a window, the system stores the application's identity; see page 8, section 5.4, first bullet point. Then, when a GUI event occurs, the enclosing window and its application are identified, and the corresponding AWT event is added to the event queue of that application, which was created when the application first opened a window; see page 8, right column, lines 24 to 26. A dispatcher thread belonging to the application then takes the event from the event queue and dispatches (deals with) it; see page 8, right column, lines 1 to 6. Consequently, applications queue and dispatch events independently of each other; see page 8, right column, lines 7 to 9.

5.7 Before the examining division the applicant argued that, in contrast to the invention, in D1 AWT events were handled by an AWT handler with was not part of the JVM. In particular [4] of the description stated that the support libraries were not part of the JVM. Events received and handled by the AWT were no longer "native" events once they passed to the JVM. D1 also did not mention foreground tasks, merely disclosing a mapping between applications and windows. The examining division saw the AWT component as a runtime library which was used by the JVM for event handling.

5.8 The board understands an AWT event to result from a native event being passed via the platform operating system to the JVM. It is certainly possible that the JVM calls routines from support libraries to handle such AWT events. However the board agrees with the examining division that, in doing so, the support libraries are acting as the agent of the JVM and thus form part of the functionality of the JVM.

6. Inventive step, Article 56 EPC 1973

6.1 According to the appealed decision, the subject-matter of claim 1 of the main request differed from the disclosure of D1 in that the foreground task was the only displayed task. In contrast, in D1 several application windows (foreground tasks) might be open and thus displayed. Having only one displayed task solved the objective technical problem of maximising the display space for a task, this being common general knowledge in the field of graphical user interfaces (GUIs). Moreover, it was common general knowledge to process user input events by the application to which the window belonged and in which the input occurred, this being the relevant window. If only one window/task was displayed, then it was obvious for this task to process the event. Hence claim 1 covered the approach known from D1 for the case of only one application window being open.

6.2 Turning to what are now the second and third auxiliary requests (the first and second auxiliary requests in the decision), the examining division found that the features added to claim 1 were known from D1.

6.3 In the grounds of appeal the appellant argued that conventional virtual machines lacked a mechanism for processing native events, such as keyboard input, for concurrently executed tasks. This meant that only one task requiring native event processing could be executed at a time or, as the description puts it, two tasks cannot be used concurrently if they both require native event processing; see [15] of the application as originally filed. The virtual machine according to the application overcame this restriction by providing an event-dispatcher which delivered native events to the foreground task running concurrently on a virtual machine on a mobile device. D1 related to the difficulties of using a JVM, which was usually used to run a single application, to instead run multiple applications for multiple users. This required a separate handling of user interface events for each application, each application having its own event queue and event dispatcher thread. User interface events were routed to the event queue of the application whose window the event affected. Figure 3 of D1 (page 7) showed an example of multiple applications running on a JVM, the Mtoolkit part of the AWT (Abstract Window Toolkit) adding AWT events to the event queue of the appropriate application.

6.4 Regarding the present main request (the same as that of the decision), the appellant has argued that D1 did not explicitly disclose the first platform being provided by a mobile device, nor was this implicit in D1. Although D1 referred to "mobile code" (section 1, 2nd paragraph), this referred to code which was downloaded from the network. The reference in D1 to "small" devices (see page 2, section 2, line 8) also did not disclose or suggest "mobile" devices. Moreover, whilst D1 concerned a JVM running tasks for multiple different users, mobile devices, such as smartphones and notebooks, were typically single-user devices. Hence there was no disclosure of the JVM of D1 running on, or being intended to run on, a mobile device. Moreover D1, in particular the window enclosing the GUI element at which the user event occurred and the associated application, did not disclose identifying a foreground (i.e. displayed) task. D1 merely disclosed the system maintaining a list of applications and associated windows and, when an event was received for a window, searching through the list to identify the associated application to which the event was to be forwarded. In contrast, the application stored the task ID of the foreground task; see figure 3, steps 312 and 314.

6.5 Regarding the first auxiliary request (which was not treated in the decision) and the third auxiliary request (which was, as the second auxiliary request), the appellant has argued that the amendments relate to the storing/recalling of the task ID. The amended claim wording sets out that, unlike in D1, the foreground task has already been identified prior to receipt of user input, i.e. the native event.

6.6 Regarding the second auxiliary request (the first in the decision, the appellant has argued that the decision merely dealt cursorily with the added features regarding the forwarding of native events to the foreground task. D1 did not disclose a task ID, the use of an object associated with the task-ID to get a handle on the event queue, encapsulating a native event so that it could be represented as a Java event object or the JVM notifying the event handler that the Java event object had been queued in the Java event queue.

7. The board's finding on inventive step, Article 56 EPC 1973

7.1 The main request

7.1.1 In view of the above discussion of D1, the subject-matter of claim 1 differs from the disclosure of D1 in that:

a. the first platform is provided by a mobile device (as argued by the appellant), and

b. only one task window is displayed (the "foreground task").

7.1.2 The board finds that difference features "a" and "b" have no synergistic effect. Hence their contributions to inventive step must be assessed separately.

7.1.3 The skilled person implementing the method known from D1, which mentions no specific hardware, would at the priority date have chosen a mobile device to provide the first platform (feature "a"), e.g. a laptop computer, as a usual design choice.

7.1.4 It does not automatically follow from the choice of a laptop that only one task window would be displayed at a time. However the board finds that maximising a task window, a usual measure when running a laptop application to display a window as legibly as possible, causes only one task window to fill the entire screen and thus become the "foreground" task (feature "b").

7.1.5 Hence neither feature "a" nor "b" can lend inventive step to claim 1.

7.2 The first auxiliary request

7.2.1 Editorial amendments aside, compared to the main request, claim 1 has been restricted to now also set out storing the ID of the foreground task (see [39]) and retrieving the ID of the foreground task (see [40]).

7.2.2 The appellant has argued that, unlike in D1 (see section 5.4, page 8, right column, lines 1 to 6, and figure 4), claim 1 sets out the foreground task having already been identified prior to receiving user input, i.e. the native event.

7.2.3 The board notes that the native events discussed in D1 are GUI events, i.e. they occur on some sort of display. In contrast, the native events discussed in the application are keyboard events; see page 4, lines 15 to 17. Hence, in the application the native event itself does not identify the application/task that should handle the event, while in D1 the window in which a GUI event occurs identifies the application/task. The board understands this to be the reason why in the application the identity of the foreground task has been previously stored (see [39], lines 1 to 3), so that, when keyboard input occurs, the JVM can immediately pass the native events to the foreground task.

7.2.4 Claim 1 has not however been limited to the case of keyboard input and thus covers the case in D1 of GUI events. In the case set out above for difference feature "b", in which an application/task window is maximised on the display, thus making it the "foreground" task, all native events in D1 would be passed to that application task, identified by its stored identity, to be handled, the identity of all windows and applications/tasks being stored by the JVM as soon as they are created, as too would the fact that a window had been maximised.

7.2.5 Hence the added features are unable to lend inventive step to claim 1.

7.3 The second auxiliary request

7.3.1 Compared to claim 1 of the main request, claim 1 of this request sets out the added feature, known from D1 (see abstract) that the virtual machine is a java virtual machine (JVM). The claim also sets out the following features which are not explicitly known from D1:

c. the JVM uses an object associated with the task ID to get a handle on an event queue and event handler for the foreground task;

d. the JVM manipulates the native event to be Java compliant by encapsulating it, so that it can be represented as a Java event object;

e. the JVM places the Java event object in the event queue of the foreground task and notifies the event handler of the foreground task of this fact, the handler then accessing said event object.

7.3.2 According to D1, GUI events are placed (in the board's understanding by the JVM) in the event queue of the associated application and dispatched by a thread of that application. Given this disclosure, it seems that the skilled person, filling in the gaps in the disclosure of D1, would have added features "c", "d" and "e", which set out usual measures when handling events in Java, as a matter of usual design and thus have arrived at the subject-matter of claim 1.

7.3.3 Hence the added features are unable to lend inventive step to claim 1.

7.4 The third auxiliary request

7.4.1 As claim 1 has been restricted using the added features from the two previous requests, it too lacks inventive step, Article 56 EPC 1973, for the reasons set out above for those requests.

Order

For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

Beschwerdekammern des Europäischen Patentamts
Boards of Appeal of the European Patent Office
Chambres de recours de l'Office européen des brevets

L. Stridde M. Müller

Decision electronically authenticated