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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 July 2012 (05.07.2012)

WIPO | PCT

(10) International Publication Number
WO 2012/090072 A1

(51) International Patent Classification:
G06F 9/455 (2006.01)

(21) International Application Number:
PCT/IB2011/003337

(22) International Filing Date:
20 December 2011 (20.12.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/428,378 30 December 2010 (30.12.2010) US

(71) Applicant *(for all designated States except US)*: **FRANCE TELECOM** [FR/FR]; 6, place d'Alleray, F-75015 Paris (FR).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **MANDANAPU, Subash** [IN/US]; 4244 Shelter Creek Ln, San Bruno, CA 94066 (US). **CHAPUT, Jean-Baptiste** [FR/US]; 3600 Fillmore St, Apt 202, San Francisco, CA 94123 (US). **MALLYA, Satya** [IN/US]; 239 Verdigris Cir, San Jose, CA 94134 (US).

(74) Agent: **FRANCE TELECOM R&D/PIV/BREVETS**; THIEL Frédéric, 38-40, rue du Général Leclerc, F-92794 Issy Moulineaux (FR).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

[Continued on next page]

(54) **Title**: SYSTEM AND METHOD FOR CROSS VIRTUAL MACHINE EXECUTION OF APPLICATIONS

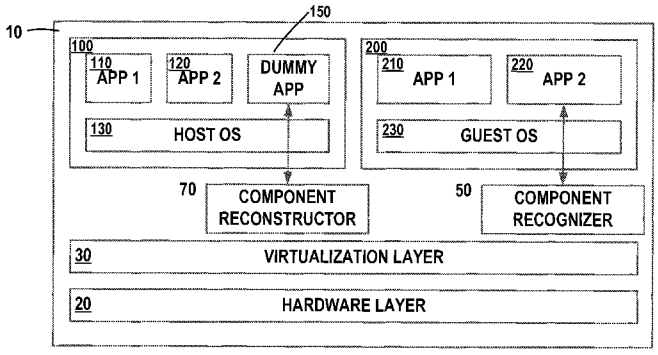


FIG. 3

(57) **Abstract**: A method for running a non-native application on a electronic device from a second non-native operating system into a first native operating system, using a dummy application in the native operating system corresponding to the non-native application, the method comprising: capturing display information created by the non-native application; parsing the captured display information into non-native application user-interface elements by a component recognizer; creating dummy application user-interface elements in the dummy application that correspond to the non-native application user-interface elements by a component reconstructor; displaying the dummy application user-interface elements within the display of the first native operating system; receiving a user interaction on one of the dummy application user-interface elements; translating the user interaction on the dummy application user-interface element into a corresponding user interaction on the corresponding non-native application user interface element; and responding, by the non-native application, to the translated user interaction.

WO 2012/090072 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

System and Method for Cross Virtual Machine Execution of Applications

Background

[0001] The invention relates generally to the field of virtual machines and the execution of an application designed for a first operating system (OS) to be accessed through the user interface of a second OS.

[0002] All OSs designed to host applications that interact with users provide tools that are made available to the applications for obtaining input from and providing output to the user (the "user interface" or UI). This has the advantage in that different applications can have a similar look and feel to the user, thereby minimizing a learning curve for interacting with the user. This has an additional advantage in that the code to handle certain aspects of the user interface can be handled by the OS tools themselves. For example, if an application developer wishes to present a list of choices to a user, he can utilize built-in user interface elements, such as push buttons, radio button boxes, drop-down menus, etc. The application developer does not have to concern himself with painting graphics on the display for presenting the user with a list, since the OS calls handle that aspect for the application developer.

[0003] It is common that a single OS controls a particular device, which permits uniformity for dealing with the device hardware. However, in some instances, it is desirable to run multiple OSs on a single device. For example, if certain applications are written for a first OS, but not for a second OS, it might still be desirable to access such applications via the second OS without having to rewrite and build the applications to be compatible with the second OS.

[0004] In virtualized environments (for both computers and portable devices such as mobile phones), it is also common to run multiple operating systems that typically work in host and guest OS modes. Each of the operating systems can host multiple application on them. There are some limited data based applications that

can be accessed via a command prompt, but this type of interface does not provide a high level user interaction. Another solution for this type of an environment is to connect to the other OS via a remote log-in solution. Such a remote log-in solution presents the application screen which is a native UI of the originating application screen, which creates a less-than-optimal user experience.

[0005] Thus, it is known in the art to permit access by a user of a host OS to a non-native application by launching a Virtual Execution Environment (VEE) within which a non-native application communicates with the host OS user. For example, a user might wish to run Microsoft Windows™ applications on her Apple Macintosh, and do this in a relatively seamless manner. In one known configuration, a separate window is provided for the Windows applications, but the applications all operate within this single window. In another known configuration, each of the Windows applications opens a new window on the host OS display. Such a windows may even be provided with the "look and feel" of the host OS by providing a host OS-based window title and border for the non-native application. The applications may be implemented within "Virtual Machines"; each Virtual Machine can have a guest operating system running within it, and each Virtual Machine can have a corresponding window on the host desktop, the window displaying a desktop of the guest operating system.

[0006] It is further known to provide an initialization menu via which a guest environment can be launched, and certain aspects of the host and guest displays are correlated to determine and create a compatibility between displays for applications within a guest environment.

[0007] Finally, it is known to adapt, by a host system, an input/output interface of a guest system to the input/output unit capability of a host system to support a service supported by an input/output unit supporting the input/output unit capability of the guest system using the at least one input/output unit of the host system. Such an adaptation could be, e.g., the adaptation of one screen resolution to another. This system has a Virtual Machine Monitor that virtualizes hardware resources to allow a guest OS to take control of the hardware resources and manages allocation of the hardware resources to the guest operating systems.

[0008] Although it is known to permit execution of guest (or “non-native”) applications in a virtual machine on a device running a host OS and providing a mechanism for providing guest application input and output from/to the user, what is not known is providing an experience for the user in which the user can interact with guest applications using low-level user interface components (buttons, menus, and the like) that have the same look and feel as the low-level user interface components of the host applications.

Summary

[0009] A preferred embodiment of the invention relates to cell phone or smart phone applications and the use of a virtual machine on such devices to permit the running of guest applications designed for a different OS than the host OS. For example, according to an embodiment, applications designed for the Android OS running on, e.g., a Motorola Droid phone can be accessed by a user using an Apple iPhone®. A number of different phone OSs exist today, including Palm OS™, webOS™, BlackBerry OS™, Windows Mobile™, Symbian™, Android™, and iPhone OS™.

[0010] Accordingly, a system and method are provided for running a non-native application on an electronic device from a second non-native operating system into a first native operating system, using a dummy application in the native operating system corresponding to the non-native application, the method comprising: initiating execution of the non-native application in the non-native operating system and capturing its display information; capturing display information created by the non-native application; parsing the captured display information into non-native application user-interface elements by a component recognizer; creating dummy application user-interface elements in the dummy application that correspond to the non-native application user-interface elements by a component reconstructor; displaying the dummy application user-interface elements within the display of the first native operating system; receiving a user interaction on one of the dummy application user-interface elements; translating the user interaction on the dummy application user-interface element into a corresponding user interaction on the

corresponding non-native application user interface element; and responding, by the non-native application, to the translated user interaction.

Brief Description of the Drawings

[0011] The invention is illustrated by reference to various embodiments illustrated in the drawings and described in more detail below.

Figure 1 is a block diagram illustrating the architecture of a known basic system having applications from both a host OS and a guest OS;

Figure 2 is a block diagram illustrating the interaction between the dummy application and an application of the guest OS;

Figure 3 is a block diagram illustrating the interaction between the dummy application and an application of the guest OS and utilizing the component recognizer and component reconstructor;

Figure 4 is a pictorial illustration of a screen on a portable device;

Figure 5 is a flowchart illustrating the UI recognizer and constructor flow; and

Figure 6 is a flowchart illustrating the UI reconstructor flow.

Detailed Description

[0012] Figure 1 illustrates a known virtualized system 10 that comprises a hardware layer 20 containing, e.g., drivers for various hardware devices, such as keyboards low-level display, communications, etc. This layer 20 occupies the lowest part of the device and has algorithms for dealing with rudimentary aspects of the hardware itself. System 10 may comprise an electronic device such as a mobile device, a tablet, a laptop or desktop computer and the likes … The here after description will be illustrated with a mobile device. This illustration is in no way limiting as the present teachings could be applied to other electronic devices operable to handle virtual machines and virtualized systems.

[0013] Above the hardware layer 20 is a virtualization layer 30 upon which a host system 100 comprising two host applications 110, 120 designed to run in a host OS 130, and a guest system 200 comprising two guest applications 210, 220 designed to run in a guest OS 230 reside.

[0014] In this configuration, if the user is interacting with an application 110, 120 on the host OS 130 and wants to run an application 210, 220 on the guest OS 230 (referred to below by the shorthand "guest application"), the user must physically switch to the guest OS 230 in order to interact with the applications 210, 220.

[0015] Figure 2 illustrates an embodiment of the invention that permits accessing an application 220 of the guest OS 230 on the host OS 130 without physically switching to the guest OS 230. As can be seen in Figure 2, when the user wishes to invoke an application 220 on the guest OS 230, a dummy application 150 is created on the host system 100 that corresponds to the application 220 on the guest OS 230.

[0016] Since the host OS 130 cannot "see" what applications are available on the guest OS 230, a predefined list of available applications must be created and loaded to an area reachable by an application in the host OS 130 so that a user can indicate on the host OS 130 which guest application 210, 220 he would like to run.

[0017] Once the user indicates on the host OS 130 the guest application 220 that he desires to run and the dummy application 150 is created, the dummy application 150 is associated with the guest application 220, and the dummy application 150 sends a message to the virtualization layer 30 to launch the application 220 on the guest OS 230.

[0018] Referring to Figure 3, once the guest application 220 is launched, a user interface component recognizer 50 analyzes a screen display generated by the guest application 220. Figure 3 shows the component recognizer 50 and reconstructor 70 outside of the host and guest system. In a preferred embodiment, the component recognizer 50 is run from within the guest system 220, and the reconstructor 70 is run from within the host system 100, since this is the least

demanding on resources. However, it is also possible to have the recognizer 50 and reconstructor 70 run within the virtualization layer 30—however, this approach is relatively demanding on resources, since the virtualization layer 30 would then have to know all UI controls of all possible different Oss and be able to associate them to the current host and guest.

[0019] The recognizer 50 performs an image analysis to determine elementary user interface components. For example, as shown in Figure 4, a display screen 300 of a portable device comprises different user interface elements such as a title bar 302, a close window button 304, and a table grid 306 comprising a heading row 308 and data rows 310.

[0020] The recognizer 50 identifies and stores information about each of the elements, including what type of element it is (button, menu, listbox, checkbox, table grid, etc., if this can be determined), its size and position on the screen, data contained within the element, event listeners (e.g., button press, mouse move) etc.

[0021] Once the recognizer 50 has identified all of the user interface elements and obtained all possible information about them, a component reconstructor 70 is activated. The reconstructor 70 creates, for each user interface element of the guest application 220 identified by the recognizer 50, a corresponding user interface element of the host OS 130 for the dummy application 150 using the host OS 130 native controls. And for each type of recognized control, an associated event handler is also created. The dummy application 150 thus displays, as its user interface, a newly created display constructed by the reconstructor 70 from the elements of the guest application 220, but using native controls and user interface element types available on the host OS 130, including layout and event listeners.

[0022] A situation may arise, however, that the recognizer 50 does not recognize a particular user interface element, or that there is no corresponding user element in the host OS 130. If there is no direct match of UI controls between the OSs 130, 230, then UI reconstructor 70 creates a custom control, which is simply a bitmap image of what was read by the image processing algorithms of the recognizer 50.

[0023] If the host OS 130 does not support a particular type of events from the guest application 220, then the UI-related activities generating events (e.g., touch screen events supported in the guest OS 230, but not in the host OS 130) then these can be accommodated in some manner by host OS supported activities and associated events. For example, a touch screen of the guest OS 230 and guest application 220 can be handled by, e.g., using a numeric keypad of the device to create a grid that simulates touch screen events. For example, if the keypad is layed out in a 3 X 3 grid for corresponding numbers 1–9, then a 1 in the upper left-hand corner of the keyboard could be pressed to simulate a screen touch in an upper left-hand corner of the screen.

[0024] Any events that occur in the user interface of the dummy application 150 that was constructed by the component reconstructor 70 are translated into the corresponding user interface events and passed on to the guest application 220 in the form of events that it is expecting.

[0025] By way of a concrete example, if an Apple iPhone is the host device running its native host OS, a parallel VM with the Android OS running separately as a guest OS may be provided. A guest application App2 in the Android environment may be a unique application that has no equivalent on the Apple Appstore.

[0026] In order to make the App2 available in the Apple VM, the dummy App2 is provided on the Apple GUI. The actions on the App2 interface are forwarded through the virtualization layer to the Android VM, and any Android based interface for App2 will be reconstructed into an Apple based interface in order to give the user a fluid experience as if the App2 was an Apple application.

[0027] Figure 5 is a flowchart that illustrates the operation of the recognizer. As can be seen in the figure, in S101, the user is provided with a list of applications that are available on the guest OS 230 and the user selects one to execute. The list of applications and respective associations have been prepared prior to this step.

[0028] This can be performed by creating some dummy icons on the host OS based on the name of the application. On the guest system, there are algorithms that go out and look for executable filenames, look at the file extensions for

executable files, and then flag those files (the guest system can expose these to the host). There may be an API that indicates if there are applications installed on the current guest system, then it sends a list of all applications currently available to the host system.

[0029] The host OS 130 creates the dummy application 150 as described above and sends a command to the guest OS 230 to start the guest application 220. The display generated by the guest application 220 is captured by the component recognizer 50 as a bitmap S102. Image processing algorithms are run S104 to locate recognizable controls on the bitmap of the guess application 220 display. Possibilities are that there is a match to a guest native control, to a host native control, or no match at all. If it finds a match of a native control in the host OS 130 for a particular UI element or control with one of the guest elements/controls S106, then this match is recorded, a newly matching native control is created in the dummy application 150, and an association is made between the two. If a match to a corresponding host native control cannot be found S112, then a bitmap image of the guest application display is created, and its properties (e.g., size, position) are created and associated with that region S114. However, in this situation where no host native control can be found, the functionality cannot be inferred (e.g., one cannot infer a corresponding event such as a button push for a pushbutton control). The associative information of corresponding host and guest controls is sent S110 to the UI reconstructor 70. So the recognizer may indicate, e.g., that this particular application has one radio button, one combination box, and a list element.

[0030] Referring to Figure 6, once the reconstructor 70 gets control information from the recognizer 50 S202, a determination is made to see if all of the controls have been processed S204 (this could be determined, for example, by a signal communicated from the recognizer 50). If so, then the UI information is sent to the host dummy application 150, where the UI of the guest application 220 is displayed using the native controls of the host OS 130.

[0031] To process a control, a determination is made S208 to see if a mapping to a native UI control element is possible. If so, a corresponding host control is created S210, and it is associated with the data obtained related to this control.

This information is then sent to the dummy application 150 on the host. If a corresponding host control does not exist, a bitmap is created and resized, if necessary, to fit into the host screen size and resolution S212. This information is then sent to the dummy application 150. Although an intelligent association of events is not possible when the control is not recognized, the user activities can still be passed off to the guest application. For example, if a user hovers her mouse over the bitmap, or clicks on it, these events can still be passed from the host to the guest application. User interactions can include any form of interaction, e.g., keystroke, a mouse move, a mouse click, a mouse drag, a mouse hover, and touchscreen equivalents.

[0032] One important aspect of this process is the event mapping. Any key event is generated through hardware interrupts. The following example is provided as an illustration. In this example a mapping is created between a non-touch host system versus a touch-based guest system. For example, a Symbian OS "list control" with a non-touch display is used as the host, and an Android with touch display is used as the guest which is running a particular application. Put differently, the user is running an Android application on a Symbian OS-based device with Symbian view.

[0033] If the user wants to select a second item in a list, she will generate a move down command by pressing a down arrow on the device, which will be translated into a hardware interrupt (e.g., INT 56). The interrupt is processed by the virtualization layer 30, and this generates an equivalent interrupt on the guest system (e.g., INT 84 for a touch down). The mapping of events between different systems may be done at the virtualization layer 30. For custom controls (e.g., games without the specific controls), the default mapping can be used (up / down / left / right / and center).

[0034] Each event is nothing but an interrupt. On the host side, the interrupt number is known for the event, and the event is passed to the virtual—the interrupt for the host OS 130 is converted into an equivalent (if possible) interrupt on the guest OS 230.

[0035] The virtualization layer 30 acts as a messenger between the OSs 130, 230. Each event / activity that occurs on the host user interface is translated and sent to the guest application. Similarly, any display update that is generated by the guest application is translated and sent to the host user interface. This continues in a cycle until termination of the guest application. The display continues to change, but the constant cycling ensures that there is a synchronization between the host side of the system and the guest side of the system.

[0036] When the user switches from a first application 210 in the guest OS to a second application 220, the responsibility of remembering context lies within the guest application. In the host OS 130, it is simply dealing with a screen of the underlying guest OS/application, so the host does not need to remember anything.

[0037] One responsibility of the virtualization layer 30 is to share resources—e.g., there is only one keyboard and one display, but they must be shared with both OSs. For any event that occurs, the virtualization layer 30 can mock it up as pertaining to one particular OS or the other OS. For example, if a user pushes a button, one can control whether this event is sent to the guest OS or to the host OS. This controlling logic lies inside the virtualization layer, and in the virtualization layer, one can also convert based on the type of event. E.g., if the device is a Windows smart phone, it knows what signal to send so that it is the exact same event—at the end-user level, if the user presses a key number nine, this might map to interrupt 86. But if this is a Windows mobile environment, it's not the same interrupt, it is expecting an interrupt 52. At the virtualization layer 30, the context can be known, i.e., the number 9 button event happened. The logic that decides what to do next resides inside the virtualization layer 30 by putting in routines for handling the interrupt, i.e., cross mapping the interrupts.

[0038] The system or systems described herein may be implemented on any form of computer or computers, although preferably on a portable device, and the components may be implemented as dedicated applications or in client-server architectures, including a web-based architecture, and can include functional programs, codes, and code segments. Any of the computers may comprise a processor, a memory for storing program data and executing it, a permanent

storage such as a disk drive, a communications port for handling communications with external devices, and user interface devices, including a display, keyboard, mouse, etc. When software modules are involved, these software modules may be stored as program instructions or computer readable codes executable on the processor on a computer-readable media such as read-only memory (ROM), random-access memory (RAM), CD-ROMs, magnetic tapes, floppy disks, and optical data storage devices. The computer readable recording medium can also be distributed over network coupled computer systems so that the computer readable code is stored and executed in a distributed fashion. This media is readable by the computer, stored in the memory, and executed by the processor.

[0039] All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to the same extent as if each reference were individually and specifically indicated as incorporated by reference and were set forth in its entirety herein.

[0040] For the purposes of promoting an understanding of the principles of the invention, reference has been made to the preferred embodiments illustrated in the drawings, and specific language has been used to describe these embodiments. However, no limitation of the scope of the invention is intended by this specific language, and the invention should be construed to encompass all embodiments that would normally occur to one of ordinary skill in the art.

[0041] The present invention may be described in terms of functional block components and various processing steps. Such functional blocks may be realized by any number of hardware and/or software components that perform the specified functions. For example, the present invention may employ various integrated circuit components, e.g., memory elements, processing elements, logic elements, look-up tables, and the like, which may carry out a variety of functions under the control of one or more microprocessors or other control devices. Similarly, where the elements of the present invention are implemented using software programming or software elements the invention may be implemented with any programming or scripting language such as C, C++, Java, assembler, or the like, with the various algorithms being implemented with any combination of data structures, objects,

processes, routines or other programming elements. Functional aspects may be implemented in algorithms that execute on one or more processors. Furthermore, the present invention could employ any number of conventional techniques for electronics configuration, signal processing and/or control, data processing and the like. The words “mechanism” and “element” are used broadly and are not limited to mechanical or physical embodiments, but can include software routines in conjunction with processors, etc.

[0042] The particular implementations shown and described herein are illustrative examples of the invention and are not intended to otherwise limit the scope of the invention in any way. For the sake of brevity, conventional electronics, control systems, software development and other functional aspects of the systems (and components of the individual operating components of the systems) may not be described in detail. Furthermore, the connecting lines, or connectors shown in the various figures presented are intended to represent exemplary functional relationships and/or physical or logical couplings between the various elements. It should be noted that many alternative or additional functional relationships, physical connections or logical connections may be present in a practical device. Moreover, no item or component is essential to the practice of the invention unless the element is specifically described as “essential” or “critical”.

[0043] The use of "including," "comprising," or "having" and variations thereof herein is meant to encompass the items listed thereafter and equivalents thereof as well as additional items. Unless specified or limited otherwise, the terms "mounted," "connected," "supported," and "coupled" and variations thereof are used broadly and encompass both direct and indirect mountings, connections, supports, and couplings. Further, "connected" and "coupled" are not restricted to physical or mechanical connections or couplings.

[0044] The use of the terms “a” and “an” and “the” and similar referents in the context of describing the invention (especially in the context of the following claims) should be construed to cover both the singular and the plural. Furthermore, recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within the range,

unless otherwise indicated herein, and each separate value is incorporated into the specification as if it were individually recited herein. Finally, the steps of all methods described herein are performable in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. The use of any and all examples, or exemplary language (e.g., “such as”) provided herein, is intended merely to better illuminate the invention and does not pose a limitation on the scope of the invention unless otherwise claimed. Numerous modifications and adaptations will be readily apparent to those skilled in this art without departing from the spirit and scope of the present invention.

What is Claimed Is:

1. A method for running a non-native application on an electronic device from a second non-native operating system into a first native operating system, using a dummy application in the native operating system corresponding to the non-native application, the method comprising:

initiating execution of the non-native application in the non-native operating system and capturing its display information;

parsing the captured display information into non-native application user-interface elements by a component recognizer;

creating dummy application user-interface elements in the dummy application that correspond to the non-native application user-interface elements by a component reconstructor;

displaying the dummy application user-interface elements within the display of the first native operating system;

receiving a user interaction on one of the dummy application user-interface elements;

translating the user interaction on the dummy application user-interface element into a corresponding user interaction on the corresponding non-native application user interface element; and

responding, by the non-native application, to the translated user interaction.

2. The method of claim 1, wherein:

operating system types comprise Palm OS™, webOS™, BlackBerry OS™, Windows Mobile™, Symbian™, Android™, and iPhone OS™;

the first native operating system is selected from the operating system types; and

the second non-native operating system is selected from the operating system types and is not the same as the first native operating system.

3. The method of claim 1, further comprising:

indicating, by a user in the first operating system, a desired execution of the non-native application.

4. The method of claim 1, further comprising:

providing a virtualization layer in the electronic device that is separate from the first native operating system and the second non-native operating system and is located between these operating systems and a hardware layer of the electronic device.

5. The method of claim 4, wherein the component recognizer is an algorithm residing within the virtualization layer.

6. The method of claim 4, wherein the component reconstructor is an algorithm residing within the virtualization layer.

7. The method of claim 1, wherein the component recognizer is an algorithm running within the second non-native operating system.

8. The method of claim 1, wherein the component reconstructor is an algorithm running within the first native operating system.

9. The method of claim 1, further comprising:

creating a list of predefined available non-native applications; and

making this list accessible to the first native operating system.

10. The method of claim 1, further comprising:

providing a virtualization layer in the electronic device that is separate from the first native operating system and the second non-native operating system and is located between these operating systems and a hardware layer of the electronic device;

wherein the initiating execution of the non-native application comprises sending, by the dummy application, a message to the virtualization layer to launch the non-native application in the non-native operating system.

11. The method of claim 1, wherein the user-interface elements are selected from the group consisting of a button, a menu, a listbox, a checkbox, and a table grid.

12. The method of claim 1, wherein the user interaction is selected from the group consisting of a keystroke, a mouse move, a mouse click, a mouse drag, a mouse hover, and touchscreen equivalents.

13. The method of claim 1, wherein:

the parsing of the captured display information comprises storing element information including a size, screen position, type, and included data for each non-native user-interface element; and

the creating of the dummy application user-interface elements comprises utilizing this stored element information.

14. The method of claim 1, wherein if the component recognizer does not recognize a non-native application user interface element as a predefined type, the method further comprises:

creating a dummy application custom control application user-interface element as a bitmap image.

15. The method of claim 1, further comprising:

creating an event handler for each type of possible user interaction on the native operating system;

creating an event handler for each type of possible user interaction on the non-native operating system; and

associating the event handlers for the native operating system with event handlers for the non-native operating system.

16. The method of claim 15, wherein the event handlers are initiated using hardware interrupts, and the associating of the event handlers comprises mapping the hardware interrupts.

17. An electronic device for running a non-native application from a second non-native operating system into a first native operating system, using a dummy application in the native operating system corresponding to the non-native application, the device comprising a processor operable to:

initiate execution of the non-native application in the non-native operating system and capturing its display information;

parse the captured display information into non-native application user-interface elements by a component recognizer;

create dummy application user-interface elements in the dummy application that correspond to the non-native application user-interface elements by a component reconstructor;

display the dummy application user-interface elements within the display of the first native operating system;

receive a user interaction on one of the dummy application user-interface elements;

translate the user interaction on the dummy application user-interface element into a corresponding user interaction on the corresponding non-native application user interface element; and

respond by the non-native application, to the translated user interaction.

18. A computer program stored on a computer readable memory medium for execution by a processor for configuring the processor to carry out a method for running a non-native application on an electronic device from a second non-native operating system into a first native operating system according one of the claims 1 to 16.

FIG. 1 (PROIR ART)

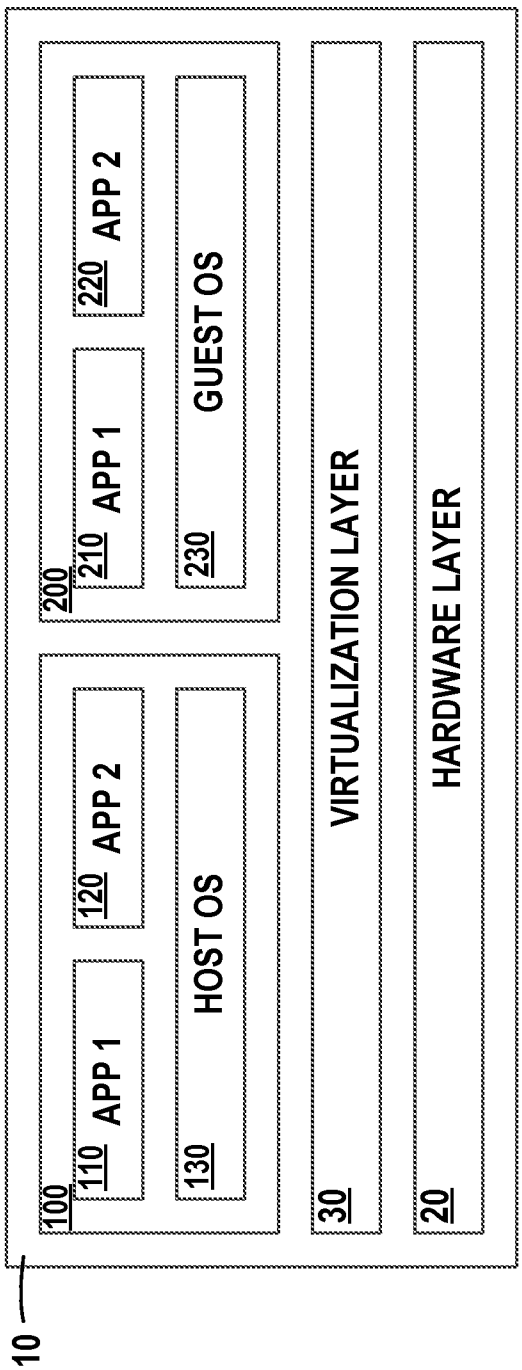
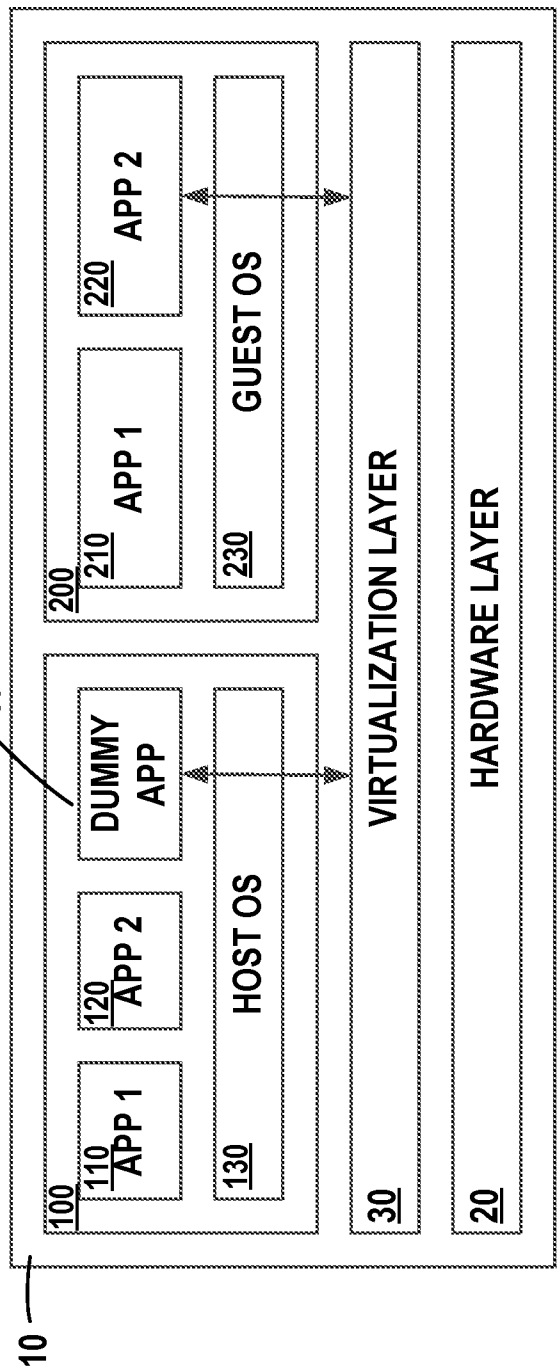


FIG. 2



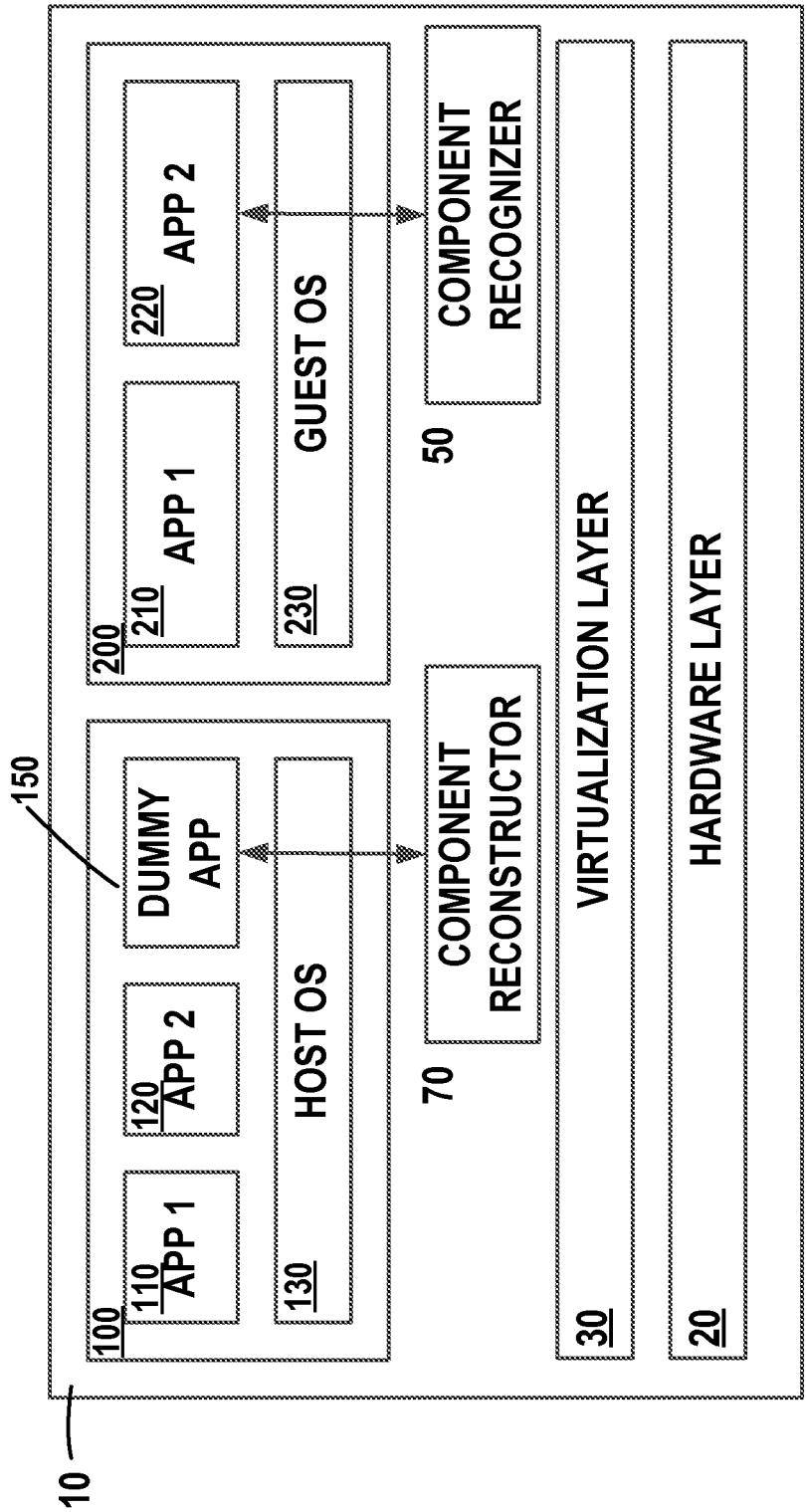

10
150
100
110
APP 1
120
APP 2
DUMMY APP
130
HOST OS
70
COMPONENT RECONSTRUCTOR
200
210
APP 1
220
APP 2
230
GUEST OS
50
COMPONENT RECOGNIZER
30
VIRTUALIZATION LAYER
20
HARDWARE LAYER

FIG. 3

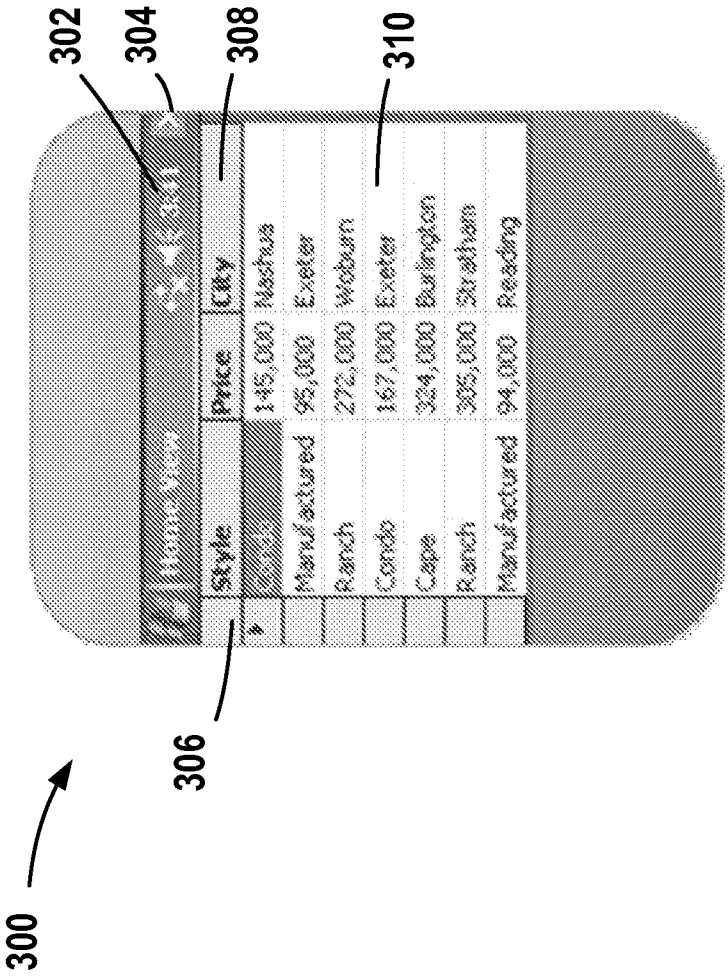

302
304
308
310
306
300
Style
Price
City
Manufactured
145,000
Nashua
Ranch
95,000
Exeter
Condo
272,000
Woburn
Cape
167,000
Exeter
Ranch
324,000
Burlington
Manufactured
305,000
Stratham
94,000
Reading

FIG. 4

UI RECOGNIZER AND CONSTRUCTOR FLOW

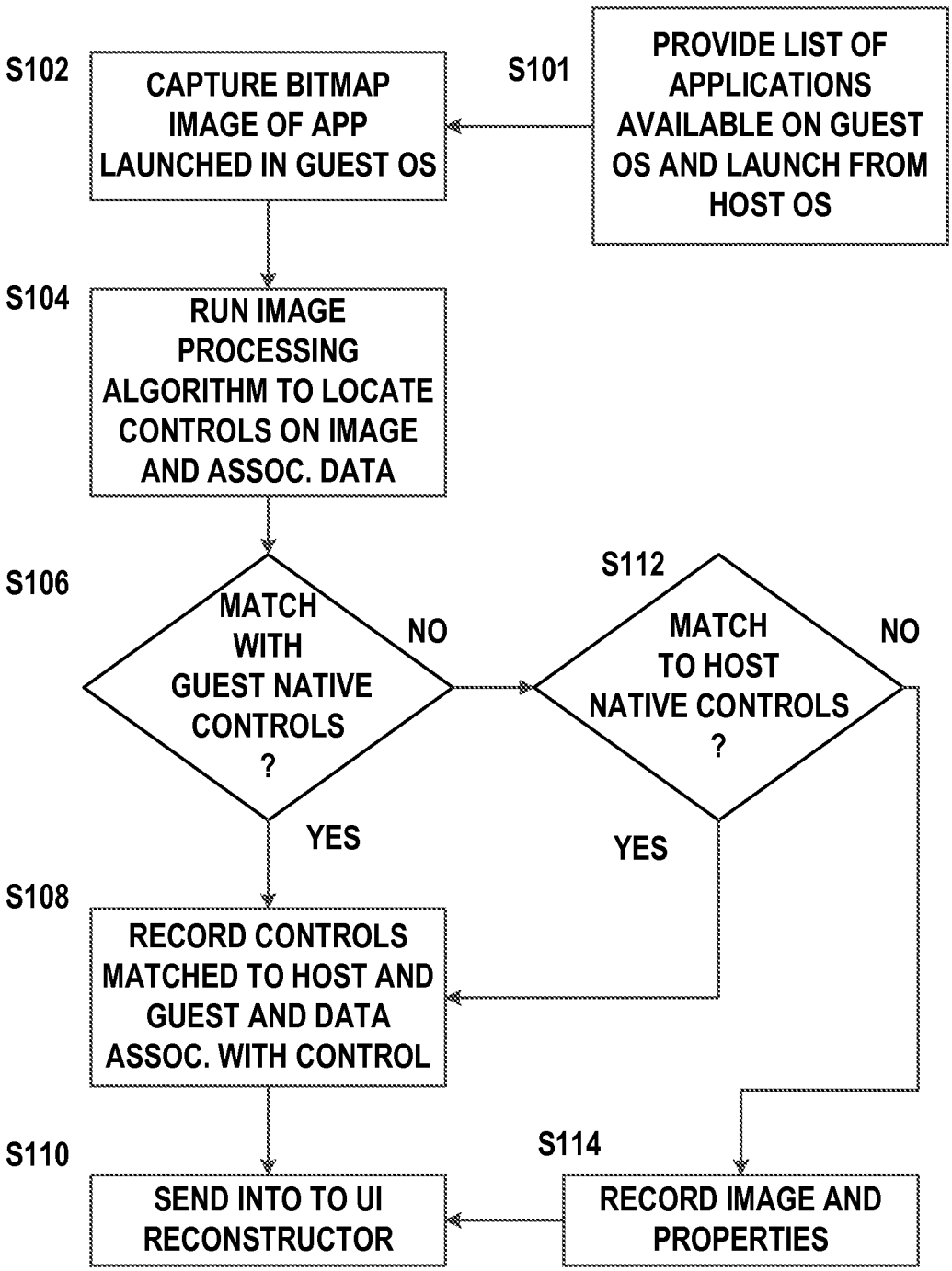


FIG. 5

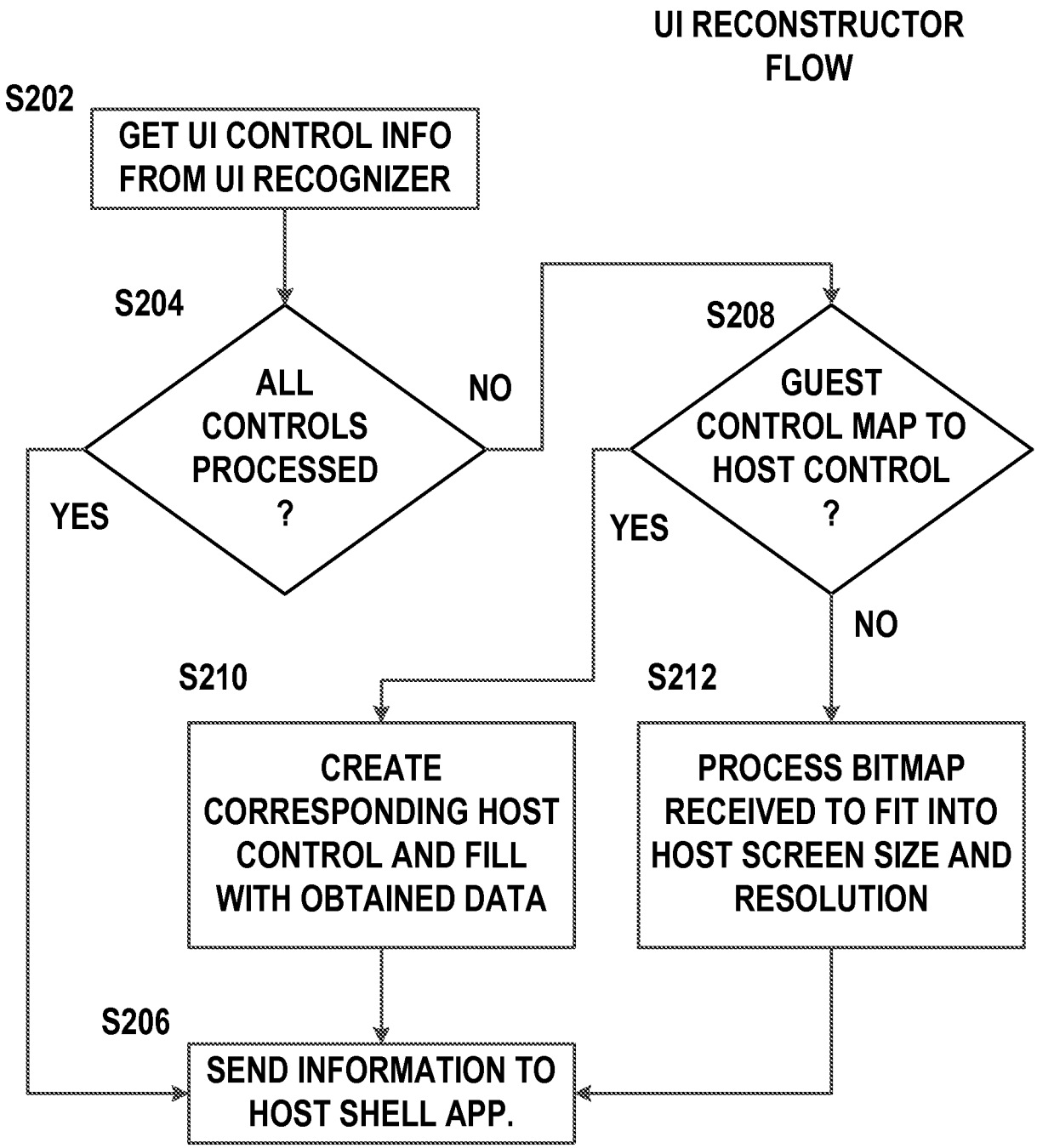

UI RECONSTRUCTOR FLOW
S202
GET UI CONTROL INFO FROM UI RECOGNIZER
S204
ALL CONTROLS PROCESSED ?
NO
YES
S208
GUEST CONTROL MAP TO HOST CONTROL ?
YES
NO
S210
CREATE CORRESPONDING HOST CONTROL AND FILL WITH OBTAINED DATA
S212
PROCESS BITMAP RECEIVED TO FIT INTO HOST SCREEN SIZE AND RESOLUTION
S206
SEND INFORMATION TO HOST SHELL APP.

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2011/003337

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/455
ADD.

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)
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 practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	ADABALA S ET AL: "From virtualized resources to virtual computing grids: the In-VIGO system", FUTURE GENERATIONS COMPUTER SYSTEMS, ELSEVIER SCIENCE PUBLISHERS. AMSTERDAM, NL, vol. 21, no. 6, 1 June 2005 (2005-06-01), pages 896-909, XP025290581, ISSN: 0167-739X, DOI: 10.1016/J.FUTURE.2003.12.021 [retrieved on 2005-06-01] abstract page 896, paragraph 1. - page 899, paragraph 3.1. page 900, paragraph 3.5. - page 904, paragraph 4.5. figures 4-6 ----- -/--	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
26 April 2012	09/05/2012

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Skomorowski, Markus

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/IB2011/003337

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1 526 424 A2 (MICROSOFT CORP [US]) 27 April 2005 (2005-04-27) abstract paragraph [0013] - paragraph [0015] paragraph [0042] paragraph [0059] - paragraph [0062] -----	1-18

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/IB2011/003337

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 1526424 A2	27-04-2005	AT 417324 T	15-12-2008
		AU 2004214609 A1	12-05-2005
		BR PI0404349 A	21-06-2005
		CA 2482281 A1	23-04-2005
		CN 1609809 A	27-04-2005
		EP 1526424 A2	27-04-2005
		HK 1076518 A1	04-09-2009
		JP 4698195 B2	08-06-2011
		JP 2005129050 A	19-05-2005
		KR 20050039530 A	29-04-2005
		MX PA04009837 A	28-04-2005
		MY 142672 A	15-12-2010
		RU 2376631 C2	20-12-2009
		US 2005091486 A1	28-04-2005
		ZA 200407587 A	31-05-2006