



(51) International Patent Classification:
G06F 9/445 (2006.01) **G06F 13/10** (2006.01)

(21) International Application Number:
PCT/GB2009/001088

(22) International Filing Date:
24 April 2009 (24.04.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0807508.7 24 April 2008 (24.04.2008) GB

(71) Applicant (for all designated States except US): **NOKIA CORPORATION** [FI/FI]; PO Box 437, Nokia Group, FI-00045 (FI).

(72) Inventor; and

(75) Inventor/Applicant (for US only): **FITZGERALD, Richard** [GB/GB]; Nokia Uk Limited, 2-6 Boundary Row, Southwark, London SE1 8HP (GB).

(74) Agent: **NOKIA UK LIMITED**; 2-6 Boundary Row, Southwark, London SE1 8HP (GB).

(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, 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, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

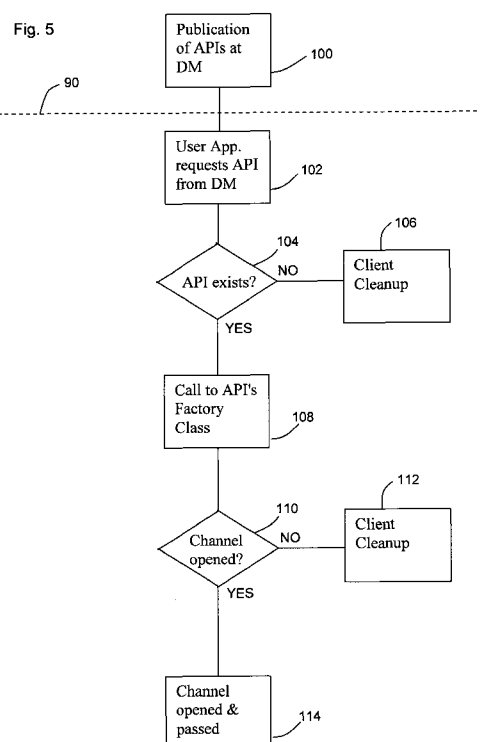
(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, 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 (Art. 21(3))

[Continued on next page]

(54) Title: **HARDWARE DEVICE COMMUNICATION**



(57) Abstract: A method and apparatus for registering application programming interfaces for hardware devices and establishing communication channels between client applications (which may be other device drivers) and a hardware device according to the application programming interface. The linking may be established dynamically, when the functionality is required.



-
- *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))*

HARDWARE DEVICE COMMUNICATION

TECHNICAL FIELD

This invention relates to communication between portions of a computing system. In particular, this invention relates to communication with hardware devices connected to a computing device.

BACKGROUND TO THE INVENTION

Device drivers facilitate communication with, and utilisation of, hardware devices such as scanners, printers, buses, cameras etc. Device drivers are implemented in a standardised way determined by the operating system under which those device drivers operate.

For security purposes, operating systems generally divide the 'space' in which executable code (processes) may operate into at least two levels: user side and kernel side. Kernel side processes are given unfettered access to memory and hardware resources. User side process, on the other hand, are more protected. Therefore, an error in a kernel side process may have wide-reaching effects such as causing the entire system to crash. Conversely, an error in a user side process generally has fewer far-reaching consequences and the operating system is more likely to be able to recover from an error in a process running in this space.

It is often necessary for device drivers to run in kernel space as these processes require low-level access to hardware resources. User applications generally run as user side processes. The operating system therefore has to facilitate communication between user side user applications and kernel side device drivers. The manner in which this is done is operating system dependent. For example, in the Symbian operating system, the kernel

manages communication with device drivers using channels. In certain flavours of the Windows operating system, communication with device drivers is handled by means of I/O request packets (IOPs).

As hardware devices have increased in complexity and the number of manufacturers
5 providing a certain type of device has increased, a need for reusable code in device drivers has arisen. Reusable code allows for the standardisation of communication between the user application and all device drivers for a certain type of hardware device according to a known application programming interface (API). Where a specific hardware device implements non-standard features, these may be implemented in the device driver over and above the
10 standard functionality.

Since device drivers may operate at kernel side whereas the user applications which communicate with these device drivers operate at user side, it is important to regulate the manner in which the device drivers are implemented. The implementation is specified within the context of the particular operating system involved.

15 In the Symbian operating system the two conflicting needs of standardisation and particularisation have given rise to logical device drivers (LDDs) and physical device drivers (PDDs). An LDD implements the functions common to all hardware devices of a common type. PDDs implement the functionality specific to a particular hardware device. Therefore, a user application will communicate with an LDD, the LDD will communicate with a PDD
20 which, in turn, communicates with the hardware device.

The LDD/PDD model suffers from certain disadvantages. Firstly, the model only implements two layers. In certain situations more than two layers would lead to more reusable code.

Each PDD is tied to a specific LDD. Therefore the model cannot be used to abstract a
25 particular hardware device which needs to be shared by different types of drivers (for

example, it would be useful to allow a power supply controller to be shared amongst various LDDs). To do so it is currently necessary to implement those as dynamic link libraries (DLLs).

Furthermore, to ensure that the communications between user side and kernel side processes are strictly controlled, the kernel only permits an LDD to use one PDD and PDDs cannot be shared amongst LDDs. It is therefore necessary to implement functionality to be shared amongst LDDs in DLLs.

The link between an LDD and its corresponding PDD can be established at runtime as LDDs and PDDs are implemented as polymorphic DLLs (this minimises the amount of memory used by the kernel at any one time). However, if an LDD relies on a further DLL for shared functionality, the advantages of runtime loading are lost, since it is necessary to load DLLs when the system starts.

As shown, using the current systems of a two-tier LDD and PDD arrangement, code abstraction is limited in the situation where it is desirable to share functionality among different hardware devices.

SUMMARY OF THE INVENTION

An embodiment of the invention provides for a method of forming a communications channel between a client application and a hardware device which comprises the steps of:

- (a) publishing an interface to said hardware device;
 - (b) receiving a request for communication with said hardware device from said client application;
 - (c) attempting to establish a communications channel between said device and said client application at runtime according to said published interface,
- wherein said client application may be a device driver or a user application.

The published interface stipulates the manner in which the communication channel may be established. Unlike prior art systems where the communication is mandated by the operating system, embodiments of the invention are able to open communication channels directly between device drivers without having to rely on extraneous components such as
5 binary files in the form of DLLs.

Preferably, runtime is a time when functionality provided by the interface is required by the client application. Because the interface for the driver is independent of the driver, the establishment of the communications channel between the device driver and the hardware client application occurs independently of how the interface is implemented. Therefore, the
10 interface may be implemented as part of the same binary file as the client application or as a distinct binary. Furthermore, it is irrelevant which type of binary is used to implement the interface (for example, this may be a generic DLL, a PDD or an LDD).

Furthermore, as the communication channel may be opened at the point in time when the interface is needed, there is no need to have this interface continually resident in memory.
15 Therefore, devices incorporating embodiments of the invention require less memory than known computing devices.

A further advantage to the invention arises in that an interface may be updated while the device is operational as it is not necessary for the entire device to be restarted, merely for the updated interface to be republished.

20 Preferably, the step of publishing said interface comprises the step of publishing a number of channels which the hardware device is capable of supporting. This allows embodiments of the invention to support many different types of device driver.

The interface may be identified by an identity label and the client application may identify the interface with reference to said identity label.

25 Preferably, a single interface is capable of supporting a plurality of channels.

Importantly, the identification corresponds to the interface, rather than to a binary, as occurs in the prior art. By forming this correspondence between the interface and the identification, the communication channel may be established at runtime.

The method may further comprise the step of establishing said communication channel directly between said device and said client application. In the known methods of implementing communications between devices and client applications a structure is imposed on the communication so that a hierarchy of access rights are imposed on the communication. The current invention provides for a direct communication and relies on implementation to establish trust and security checks.

The hardware devices may be categorised into one of four or more types and the step of creating the communications channel for a device may then be dependent upon a type of the device.

The device types may include single-channel, multi-channel, limited multi-channel and unit-based multi-channel, wherein:

- single-channel devices support only one channel of communication;
- multi-channel devices support unlimited channels of communication;
- limited multi-channel devices support a predetermined number of channels of communication, but more than one; and
- unit-based multi-channel devices support a predetermined number of channels of communication wherein each channel may be formed with only one client application.

Each channel may relate to a corresponding client application.

The step of attempting to establish the communications channel may comprise the steps of establishing that the request for communication cannot be fulfilled and informing the client application thereof.

The interface may be published by a device manager and in this case the device manager may receive the request for communication from the client application.

The method may include the step of registering one or more interfaces corresponding to the hardware device with the device manager. Preferably, the method comprises the additional step of registering a plurality of interfaces for a plurality of hardware devices with the device manager.

A further embodiment of the invention provides for a device manager for establishing communication channels between hardware devices and client applications, the device manager comprising:

- 10 means for registering an interface corresponding to the hardware device;
- means for receiving a request for a communication channel with the hardware device and a client application; and
- means for attempting to establish a communication channel between the hardware device and the client application according to the interface.

15 The device manager is preferably adapted to publish the registered interface.

The device manager is preferably adapted to publish a number of channels which the corresponding hardware device is capable of supporting.

Each interface may be identified by an identity label and in this case the client application may identify the interface with reference to the identity label.

20 The device manager may be adapted to support at least four or more device types, the device types including one or more of: single-channel, multi-channel, limited multi channel and unit-based multi-channel and wherein:

- single-channel devices support only one channel of communication;
- multi-channel devices support unlimited channels of communication;

limited multi-channel devices support a predetermined number of channels of communication, but more than one; and

unit-based multi-channel devices support a predetermined number of channels of communication wherein each channel may be formed with only one client application.

Each of the interfaces may be derived from a common class.

A further embodiment of the invention provides for a computing device comprising a kernel and a hardware device, the computing device being adapted to operate a user client application for communicating with the hardware device, the computing device further comprising a device driver corresponding to the hardware device, the kernel including a device manager for establishing communication with said device driver, the device manager being adapted to perform the method as hereinbefore described.

The device manager may be adapted to publish a plurality of interfaces and each of the interfaces may then be derived from a common class.

At least one of the interfaces may be adapted to perform a security check on said user client application.

The testing procedure may include a strong and/or a weak parameter check.

The client application testing procedure may be dependent on the type of the client application.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention are hereinafter described with reference to the accompanying drawings where like numerals are used to denote like features and wherein:

Figure 1 is an illustration of a mobile computing device operating in accordance with a preferred embodiment of the invention;

Figure 2 is a schematic illustration of various components of the mobile computing device of Figure 1;

Figures 3A to 3C illustrate various known hardware arrangements and corresponding driver arrangements;

5 Figure 4 is a schematic diagram of a portion of a computing device implementing an embodiment of the invention;

Figure 5 is a flow diagram of the operation of an embodiment of the invention;

Figures 6A to 6C illustrate driver arrangements according to embodiments of the invention corresponding to the hardware arrangements of Figures 3A to 3C; and

10 Figure 7 illustrates an alternative driver arrangement according to a further embodiment of the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

Figure 1 illustrates a mobile computing device 10 which includes a casing 12, a keypad 14, a screen 16, a speaker 18, a microphone 20 and an antennae 22. The mobile computing device 10 may be used as a mobile phone in a manner known in the art. The keypad 14, screen 16, speaker 18 and microphone 20 are hardware devices and their interactions with one another and with a central processing unit (not shown in the diagrams) is controlled by an operating system. In the embodiment shown, the operating system used is the Symbian operating system produced by Symbian Software Ltd., UK although it will be realised that the invention is not limited in this respect.

Figure 2 is a schematic illustration of certain components of the mobile computing device 10. The operating system is represented by kernel 22. A user application 24 communicates with the kernel 22 which is connected to system memory 26. Likewise device drivers 28, 30 and 32 are connected to kernel 22 and control the operation of the respective devices: keypad 14, screen 16 and NAND flash memory 34. Keypad 14, screen 16 and

memory 34 are provided by way of example; the device 10 includes a number of additional devices not illustrated in Figure 2. The kernel 22 controls the operation of the hardware devices 14, 16 and 34 by means of the device drivers 28, 30 and 32.

Figures 3A to 3C illustrate various hardware arrangements and the limitations in the known Symbian OS arrangement are described with reference to these Figures. Figure 3A illustrates a portion of a hardware arrangement 300 comprising a central processing unit (CPU) 302 with an integrated graphics processing capability, here denoted GFX 304. CPU 302 is linked to an off chip image processor 306 which is in turn linked to a camera A 308. In practice, image processor 306 and camera A 308 would be provided together as an integrated device.

Also illustrated in Figure 3A is the arrangement 310 of drivers (in the form of DLLs and PDD) corresponding to the hardware arrangement 300. A GFX DLL 312 corresponds to the GFX processor 304 integrated with the CPU 302. Camera A PDD 314 has been specified for the camera A 308. I/O Driver DLL 316 controls input and output to the CPU 302 whereas image processor DLL 318 corresponds to image processor 306 and controls the operation of this component. It will be apparent to one skilled in the art which of the DLLs illustrated may be implemented as polymorphic DLLs and therefore the label "DLL" is used to refer to LDDs too.

The situation illustrated in Figure 3A corresponds to the camera A 308 utilising the image processor 306 to process captured images and communicate these to CPU 302 by means of the input/output functions of that chip. Therefore, links 320 are established between the camera A PDD 314 and the I/O driver DLL 316, and between the camera A PDD 314 and the image processor DLL 318. The camera A PDD 314 contains code specific to the corresponding hardware device (camera A 308) and the links 320 are established to allow the camera A 308 to communicate with the image processor 306 and CPU 302.

Figure 3B illustrates a hardware arrangement 330 where a different camera, camera B 332, is connected to CPU 302. Camera B has a lower cost than camera A and requires the use of the integrated graphics processor GFX 304. The corresponding driver arrangement 340 illustrates the PDD 342 for camera B. The camera B PDD 342 links to the I/O Driver DLL 316 to facilitate communication between camera B 332 and the CPU 302 by means of link 322 similar to the links 320 illustrated in Figure 3A. Furthermore, the camera B needs to communicate with the GFX DLL 312 to take advantage of the integrated graphics processor 304. This may be done by two alternative arrangements, illustrated in dashed outline in Figure 3B. As a first alternative a dummy image processor DLL stub 344 is provided and the camera B PDD links 324 to the stub 344, but the stub 344 links by link 326 to the GFX DLL 312. As a second alternative, a new camera B PDD 346 is provided which links 328 directly to the GFX DLL 312. Both alternatives have drawbacks: a stub DLL requires unnecessary, disorganised coding, whereas a revised PDD requires redevelopment of this component. Neither solution provides for reusable code.

Figure 3C illustrates a hardware arrangement 350 where both camera A 308 and camera B 332 are connected to the CPU 302 (camera A 308 being connected by means of image processor 306). Figure 3C further illustrates an arrangement 360 of the drivers corresponding to the hardware arrangement 360. In the driver arrangement 360 it is not possible to use a stub for the image processor DLL as the actual image processor DLL 318 is needed by the camera B PDD 346.

Figure 4 illustrates a portion of the computing device 10 of Figure 2 and illustrates a hardware device 50 and a device driver 52. For the purposes of describing the invention, the hardware device 50 may be any one of the hardware devices of the mobile computing device 10 illustrated in Figure 2.

The client application 24 communicates with the hardware device 50. A device manager 62 regulates this communication between the client application 24 and the hardware device 50 in the manner herein described. The client application may be a software application such as photo capture software, or may be another device driver (e.g. a camera device driver interacting with a bus device driver). An important aspect of embodiments of the invention is that any application may act as a client application provided that the correct interface is requested in the correct manner by that application.

The device driver 52 includes a name 54 which acts as an identity label for that device driver. The device driver includes different APIs corresponding to varying functionality. In the embodiment shown, the device driver 52 includes three APIs 56A, 56B and 56C and corresponding unit numbers 58A, 58B and 58C. The manner in which units and corresponding APIs are implemented is described in greater detail below.

Device driver 52 supports unit-based multi-channel communication and therefore the APIs in this device driver have corresponding unit numbers. Other types of device drivers do not require the use of unit numbers, as described below.

The device manager 62 publishes the APIs of the device drivers of the computing device 10. Therefore the device manager 62 includes the names 64 of the device drivers and, for each device driver, a listing of the possible units 66, if applicable, and corresponding APIs 68 for that device driver. In the embodiment illustrated in Figure 3, the entry 64A in device manager 62 corresponds to device driver 52. Therefore a pointer to the name 54 of device driver 52 will be stored at position 64A. Pointers to the unit numbers 58A, 58B and 58C will be stored in positions 66A, 66B and 66C. Similarly, pointers to the APIs 56A, 56B and 56C of the device driver 52 will be stored at positions 66A, 66B and 66C of the device manager 62.

Each API for all device drivers will be derived from a common base object. This provides consistency in the implementation of the APIs and ensures that each API inherits the requisite interface framework. However, the implementation requirements of each API is kept intentionally unspecific, allowing each API to define the degree to which the API will trust the user application calling that API (e.g. whether to implement strong parameter checking or not). This allows direct communication between the client application and the hardware device (i.e. the specifics of the communication channel are specified by the interface and not by the operating system). In the embodiment shown, the APIs are implemented in C++ and appropriate classes for using the API are defined in header files (not shown), in a manner known in the art. Therefore, checking (such as parameter or permission checking) may be implemented in a manner known in the art.

The device manager 62 includes a further row of entries corresponding to a further device driver with name stored at position 64B. As illustrated in Figure 4, this further device driver includes three units with corresponding APIs. The device manager 62 includes a further device driver with name stored at position 64C. This device driver includes a single unit and corresponding API. In this instance, the unit number is not necessary and, in an alternative embodiment of the invention, is omitted for all APIs where not required.

Figure 4 further illustrates the user application 24 which communicates with the device manager 62. Generally in Figure 4, dashed lines 72 denote communication between the illustrated components.

It is to be realised that the implementation illustrated in Figure 4 is symbolic and that in practice many more elements, such as APIs and device drivers, than those illustrated will be involved. Although the implementation has been illustrated with the use of tabular structures, a person skilled in the art will realise that many other implementations are possible (for example, a mark-up language such as XML may be used).

Figure 5 is a flow diagram illustrating the manner in which an embodiment of the invention functions. At block 100 all of the APIs of all device drivers of the computing device 10 are published. This involves each of the device drivers (such as device driver 52, Figure 4) communicating with the device manager 62 to inform the device manger 62 of the APIs and corresponding unit numbers (if applicable) for each API of that device driver. The device manager 62 publishes the device driver specific information by making the information available in the form illustrated in Figure 4. The process of publication represented by block 100 will occur at the time when the device manager 52 starts up. Dashed line 90 represents an imaginary boundary between start-up and runtime. The start-up of the device manager 52 may occur when the device starts up or when the application representing the device manger 52 is started.

At runtime, the client application 24 (which may be a user application or another device driver) will request an API by referring to the device driver unique identity name and the unit number (if applicable) of that API. The client applications are expected to know the requisite information to request the relevant API. Therefore, the manufacturer of the device driver will have to publish the details of what APIs, units and device names correspond to a particular device driver.

At decision block 104 the device manager 62 determines whether the requested API exists or not. If the API does not exist, the client application 24 is informed of this by the device manager 62 in process block 106 and the client will then perform the appropriate clean up. If the API does exist, the process proceeds to block 108 where a call is made to the device driver's factory class. Each device driver will include a factory class (not shown) which determines in what manner communications between a client application and that device driver may be implemented. There are four distinct types of device driver based on the communication channels it is capable of supporting:

Single channel: the driver can only support a single communication channel to a single client. This applies where there is a single hardware device capable of supporting only a single communication channel (and therefore only one client application) at any one time. An example of such a device is a single UART chip.

5 Limited multi-channel: the driver supports a predefined number of channels of communication. This driver type is implemented where a single hardware device supports a predefined number of multiple simultaneous operations or clients. For example, a hardware encryption block which can encrypt four data streams in parallel or a 3D graphics engine capable of rendering up to eight simultaneous channels to a display. Depending on the
10 particular hardware involved (e.g. whether the hardware implements registers which may be shared between threads) the device driver may, or may not, implement some form of thread safety.

Multi-channel: the driver can support an unlimited number of channels of communication. Two types of corresponding hardware device are supported by multi-
15 channel device drivers. Firstly, hardware devices which are single channel or limited multi-channel, but the driver emulates multi-channel communication. Examples of this are buses such as MMC/SD, I2C or SPI buses where only one client transaction may be on the bus at any one time, but bus access by an unlimited number of clients may be accommodated by queuing. In a further example a limited multi-channel device such as a 16-channel DMA
20 device driver may emulate additional channels and thereby act as a multi-channel device.

The second type of hardware device which employ multi-channel device drivers are those hardware devices which support unlimited use. For example, a hardware RTC (real-time clock) supports an unlimited number of calls for the time.

It should be noted that the APIs for such multi-channel device drivers will have to provide techniques to accommodate multiple threads, such as calls to lock and unlock functions or, alternatively, each client could be protected by a mutex.

Unit-based multi-channel: the driver supports a predetermined number of channels
5 and each channel is identified by a corresponding unit number. This is a manner of implementing the aforementioned limited multi-channel device drivers, but there is a distinction between the various channels and it is therefore necessary to identify which channel is established. An example of this would be the use of a particular RS232 port. As previously stated, the unit number of that communication channel may correspond to a
10 particular API or the same API may be allocated to each unit. In the embodiment illustrated in Figure 4, the device driver 52 is a unit based multi-channel device driver.

Referring back to Figure 5, the device manager 62 will, at block 108, instruct the device driver 52 to attempt to establish a communication channel between the device driver 52 and the client application 24 according to the specifics of the factory class (the class
15 responsible for establishing communication) for that type of device driver.

At block 110, the device manager 62 will determine if the opening of the communications channel was successful. If the channel was not successfully opened, the procedure will move to block 112 where the client application 24 will perform clean up. Alternatively, if the channel has been successfully created between the client application 24
20 and the device driver 52, the process will move to block 114 where the device manager 62 passes the channel to the client application 24.

As stated, the client application 24 may be another device driver, and the aforementioned embodiment therefore allows linking between device drivers without the use of a DLL which needs to be loaded at startup. Instead, the linking between the device drivers
25 may be implemented at runtime.

The operation of the invention will now be considered with reference to a specific example. The example used here refers to the manner in which the Symbain operating system uses a NAND flash drive (such as the NAND flash memory 34 illustrated in Figure 1) under normal use and under use by a crash logger. Traditionally, a normal media driver
5 would entail medusii.pdd, the physical device driver implementing the general media driver interface. This uses the usii_pil.dll, the platform independent layer for general NAND management which in turn uses usii_psl.dll, the platform specific layer which will correspond to a particular NAND flash drive.

The crashlogger, on the other hand, comprises the core part of the crash logger
10 exmonlog.dll, the same platform independent layer usii_pil.dll as well as a platform specific crashlog layer, usii_psl_crashlog.dll.

Under the previously known manner of implementing device drivers, usii_pil.dll can only link to both usii_psl.dll and to usii_psl_crashlog.dll when both of the platform specific layers are resident in the system memory at the same time.

15 Under the framework of an embodiment of the current invention, both usii_psl.dll and usii_psl.dll are implemented as two separate runtime APIs. For both implementations, the first API corresponds to the normal use and the second API corresponds to the crashlog use. All APIs are published by the device manager 62.

When a normal NAND driver is required, the following steps are followed:

20 1) medusii.pdd requests the device manager for the first API of the platform independent layer.

2) The platform independent layer will then open a channel to the medusii.pdd providing access to the requested API.

3) The platform independent layer will, in turn, request the first API for the platform
25 specific layer from the device manager.

4) The platform specific layer could then request the API for any further hardware required.

In the case of the crashlogger, the following steps are followed:

1) exmon.dll requests the device manager for the second API corresponding to the platform independent layer.

2) The platform independent layer will then open a channel to the medusii.pdd providing access to the requested API.

3) The platform independent layer will, in turn, request the second API for the platform specific layer from the device manager.

4) The platform specific layer could then request the API for any further hardware required.

From this it can be seen that the link between the device driver and the client application is established at runtime (i.e. when the client application requires the functionality provided by the requested API of the device driver). Therefore, it is not necessary for that API to exist in memory prior to the request being initiated. Instead, the API may be loaded at runtime.

Figures 6A to 6C illustrate a further implementations of the invention. These Figures correspond to the hardware arrangements shown in Figure 3A to 3C. Figure 6A illustrates a driver arrangement 600 according to an embodiment of the invention corresponding to the hardware arrangement 300 of Figure 3A. The I/O driver DLL 616, image processor driver DLL 618 and GFX driver DLL 612, corresponding to the hardware components of Figure 3A, register corresponding APIs with device manager 62 in the manner described above with reference to Figure 4 and illustrated by arrows 620, 622 and 624. The camera A PDD 614 is the client application in this arrangement and requests the APIs from the device manager 62 in the manner described. Arrows 626 and 628 illustrate these requests and the subsequent

communication between camera A PDD 614 and image processor DLL 618 and GFX DLL 612. Referring to the hardware arrangement 300 of Figure 3A, the camera A 308 is thereby able to communicate with the image processor 306 and thereby with the CPU 302.

Figure 6B illustrates a driver arrangement 640 corresponding to the hardware arrangement 320 of Figure 3B. In this embodiment, the device drivers GFX DLL 612 and I/O driver DLL 618 register APIs with the device manager 62 and these APIs are passed along to the camera B PDD 642. Therefore, camera B 332 (Figure 3B) is able to communicate with CPU 302 and utilise the integrated graphics processor 304. In an alternative embodiment the GFX DLL 612 publishes an image processor API, and the client application, camera B PDD links to this published API. As can be seen, the invention obviates the need to provide a DLL stub or re-engineer the PDD, as required in the prior art arrangement previously discussed.

Figure 6C illustrates a driver arrangement 650 according to an embodiment of the invention corresponding to the hardware arrangement 350 of Figure 3C. In this arrangement image processor DLL 618, I/O Driver DLL 616 and GFX DLL 612 register corresponding APIs with device manager 62 as illustrated by arrows 620, 622 and 624. These APIs are then requested by the two camera PDDs 642 and 614 as illustrated by arrows 652, 654, 656 and 658. As illustrated, the I/O driver DLL 616 is able to publish APIs to enable communication between the image processor DLL 618 and both camera PDDs, as shown by arrows 654 and 656. This allows both camera A and camera B to communicate with the CPU 302. The APIs are implemented as multi-channel device drivers.

Figure 7 illustrates an alternative driver arrangement 700 corresponding to the hardware arrangement 350 illustrated in Figure 3C. The arrangement 700 differs from the arrangement 650 illustrated in Figure 6C in that a modified GFX DLL 712 publishes the API for the integrated graphics processor 304 (Figure 3C) and the API for the I/O functions of

CPU 302, as illustrated by respective arrows 722 and 724. The remainder of the arrangement of Figure 7 is similar to that of Figure 6C and like numerals have been used to denote like components.

As illustrated in Figures 6C and 7, an API may be provided in one of many binary
5 DLLs in embodiments of the invention. Therefore, embodiments of the invention provide flexibility to hardware manufacturers, particularly where hardware manufacturers need to rely on software provided by a third party for an integrated hardware device to operate (such as a mobile phone).

It is to be realised that the term "device driver" as used herein refers to a collection of
10 APIs which may be considered to correlate to a particular hardware device. However, according to embodiments of the invention, the inclusion of a particular API governing the operation of a particular hardware device need not be provided in a device driver specifically associated with a hardware device. In fact, where a component for a computing device includes a number of co-operating hardware devices, all of the necessary APIs may be
15 provided in a single binary providing one 'device driver' for that hardware device.

Claims

1. A method of forming a communications channel between a client application and a hardware device which comprises the steps of:

- 5 (a) at runtime, publishing an interface to said hardware device;
- (b) receiving a request for communication with said hardware device from said client application;
- (c) attempting to establish a communications channel between said device and said client application according to said published interface,
- 10 wherein said client application may be a device driver corresponding to a further hardware device or a user application.

2. The method according to claim 1 wherein said step of publishing said interface comprises the step of publishing a number of channels which the hardware device is capable
15 of supporting.

3. The method according to claim 1 or claim 2 wherein said interface is identified by an identity label and wherein said client application identifies said interface with reference to said identity label.

20 4. The method according to any preceding claim further comprising the step of establishing said communication channel directly between said device and said client application.

5. The method according to any preceding claim wherein said hardware devices are categorised into one of four or more types and wherein said step of creating said communications channel for a device is dependent upon a type of the device.

5 6. The method according to claim 5 wherein said device types include single-channel, multi-channel, limited multi channel and unit-based multi-channel and wherein:

 single-channel devices support only one channel of communication;

 multi-channel devices support unlimited channels of communication;

10 limited multi-channel devices support a predetermined number of channels of communication; and

 unit-based multi-channel devices support a predetermined number of channels of communication wherein each channel may be formed with only one client application.

15 7. The method according to any preceding claim wherein said step of attempting to establish said communications channel comprises the steps of identifying that the request for communication cannot be fulfilled and informing the client application thereof.

20 8. The method according to any preceding claim wherein said interface is published by a device manager and wherein said device manager receives said request for communication from said client application.

 9. The method according to claim 8 further comprising the step of registering one or more interfaces corresponding to said hardware device with said device manager.

25

10. The method according to claim 9 further comprising the step of registering a plurality of interfaces for a plurality of hardware devices with said device manager.

11. A device manager for establishing communication channels between hardware
5 devices and client applications, said device manager comprising:

means for registering an interface corresponding to said hardware device;

means for receiving a request for a communication channel with said hardware
device and a client application; and

10 means for attempting to establish a communication channel between said
hardware device and said client application according to said interface.

12. The device manager according to claim 11 adapted to publish said registered
interface.

15 13. The device manager according to claim 12 adapted to publish a number of
channels which the corresponding hardware device is capable of supporting.

14. The device manager according to any one of claims 11 to 13 wherein each
interface is identified by an identity label and wherein said client application identifies said
20 interface with reference to said identity label.

15. The device manager according to any one of claims 11 to 14 adapted to support at
least four or more device types, said device types including one or more of: single-channel,
multi-channel, limited multi channel and unit-based multi-channel and wherein:

25 single-channel devices support only one channel of communication;

multi-channel devices support unlimited channels of communication;

limited multi-channel devices support a predetermined number of channels of communication; and

unit-based multi-channel devices support a predetermined number of channels of communication wherein each channel may be formed with only one client application

16. The device manager according to any one of claims 11 to 15 wherein each of said interfaces is derived from a common class.

17. A computing device comprising a kernel and a hardware device, said computing device being adapted to operate a user client application for communicating with said hardware device, said computing device further comprising a device driver corresponding to said hardware device, said kernel including a device manager for establishing communication with said device driver, said device manager being adapted to perform the method of any one of claims 1 to 6.

18. The method according to claim 17 wherein said step of publishing said interface includes the step of publishing a plurality of interfaces and wherein each of said interfaces is derived from a common class.

19. The method according to claim 17 or claim 18 wherein at least one of said interfaces is adapted to perform a security check on said user client application.

20. The method according to claim 19 wherein said security check includes check to assess whether said client application includes strong and/or weak parameter checking.

21. The method according to claim 19 or claim 20 wherein said client application testing procedure is dependent on the type of the client application.

22. A computing device comprising two hardware devices, said computing device comprising instructions adapted to implement the method of any one of claims 1 to 10 to thereby allow said hardware devices to communicate with one another.

10

23. A method of operating a hardware device attached to a computing system which includes the steps of:

on said computer system, publishing an interface determining one or more operations of said hardware device;

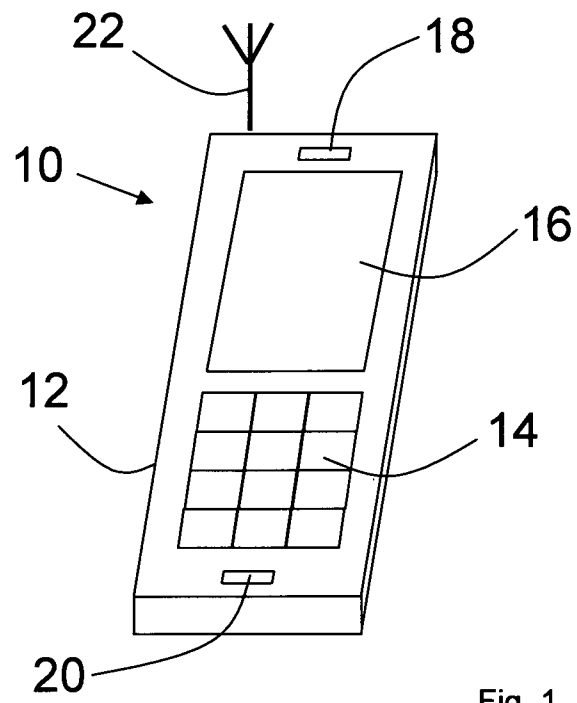
15 at runtime, allowing a client application on said computing system to perform at least one of said operations of said hardware device according to said interface, wherein said client application may be an interface for a further hardware device.

24. The method according to claim 23 wherein a plurality of client applications may perform one or more operations of said hardware device according to said interface by multiple instantiations of said interface.

25. The method according to claim 24 wherein each instantiation of said interface is identified by a corresponding unit number.

25

26. The method according to any one of claims 23 to 25 wherein each interface is associated with a corresponding device driver.



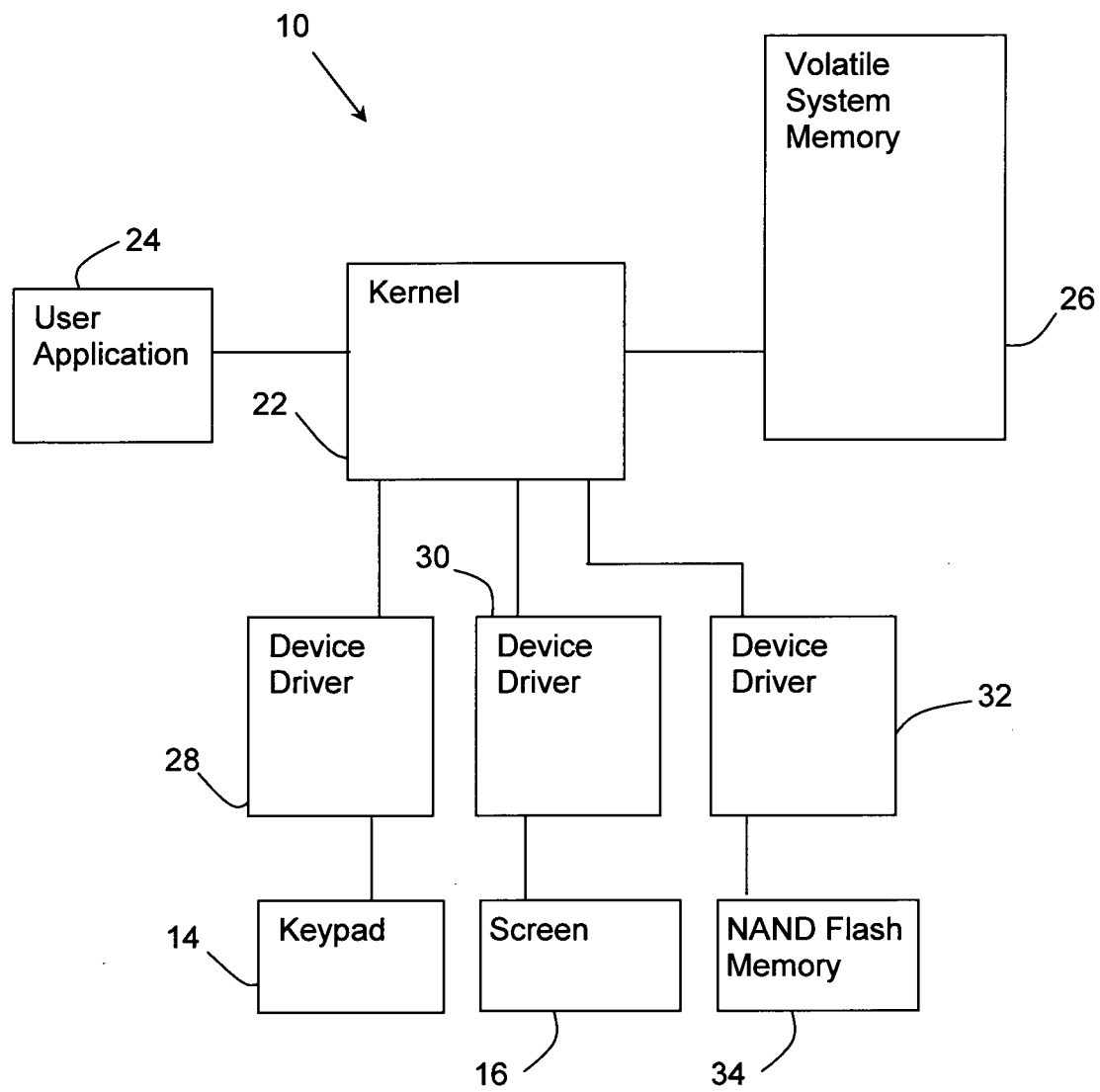


Fig. 2

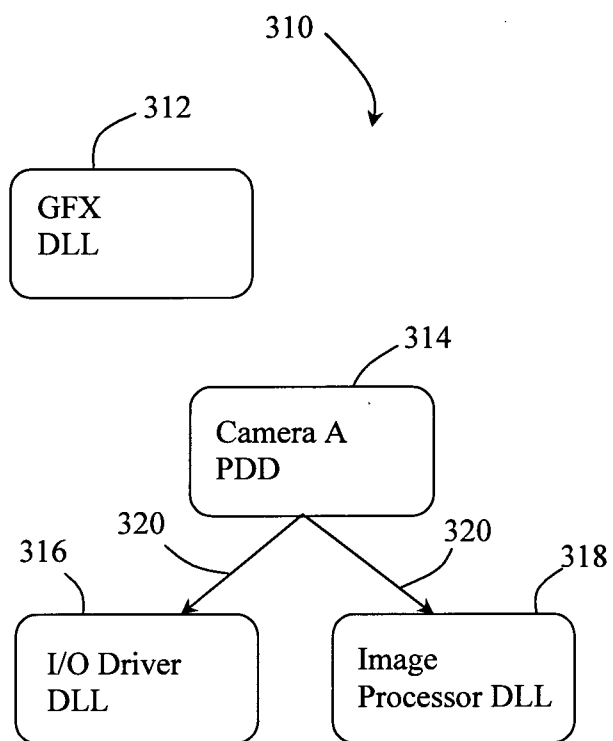
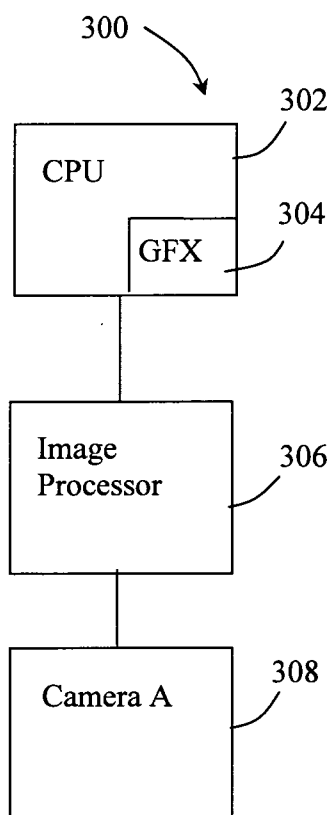


Fig. 3A

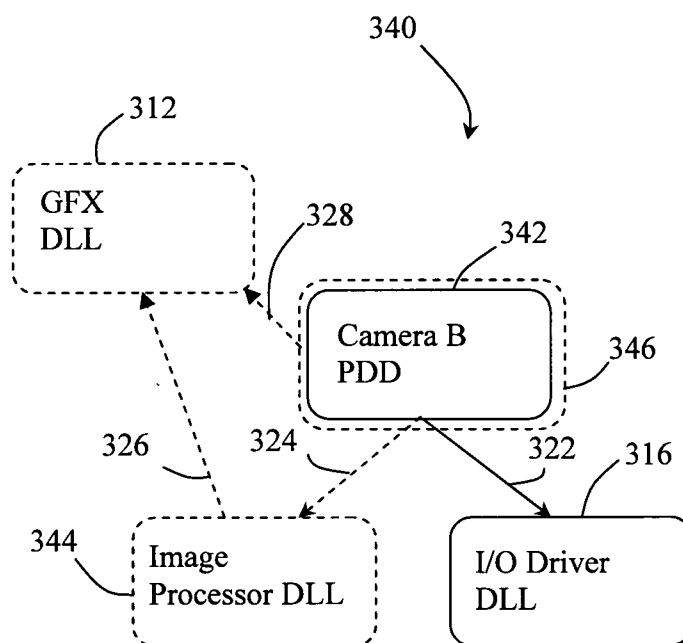
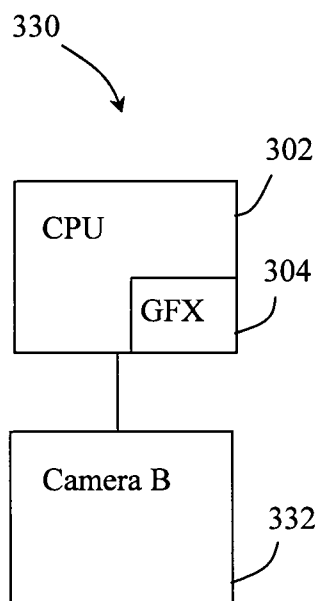


Fig. 3B

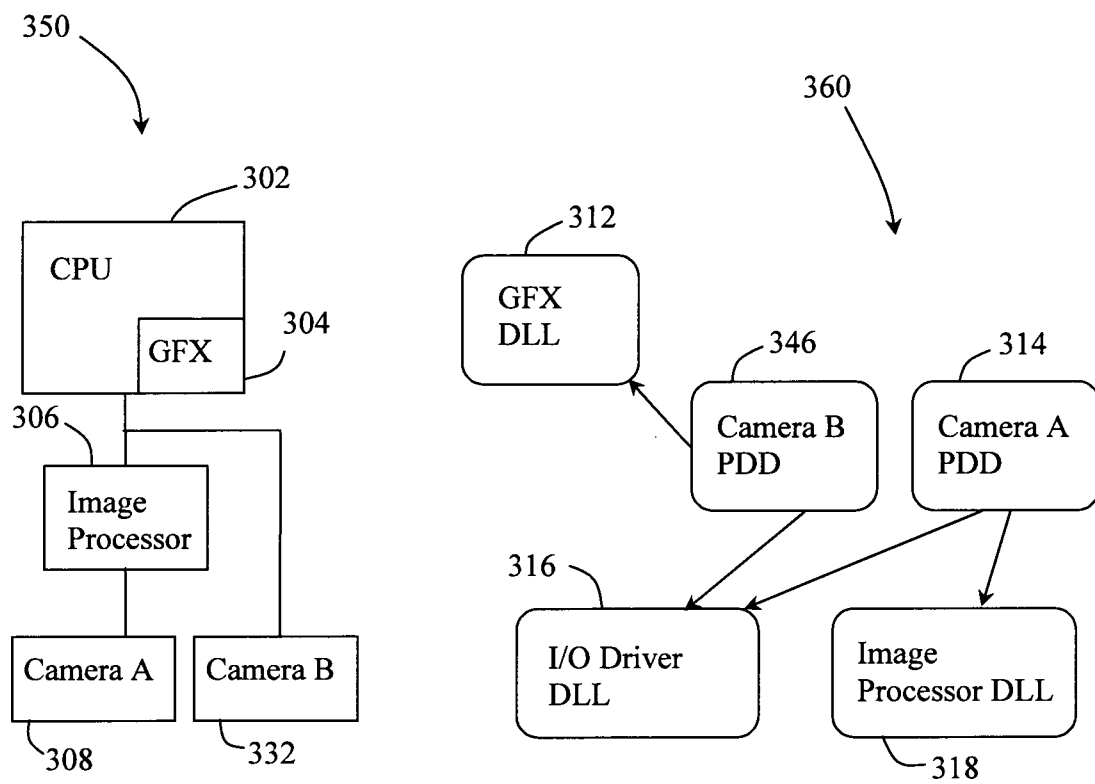


Fig. 3C

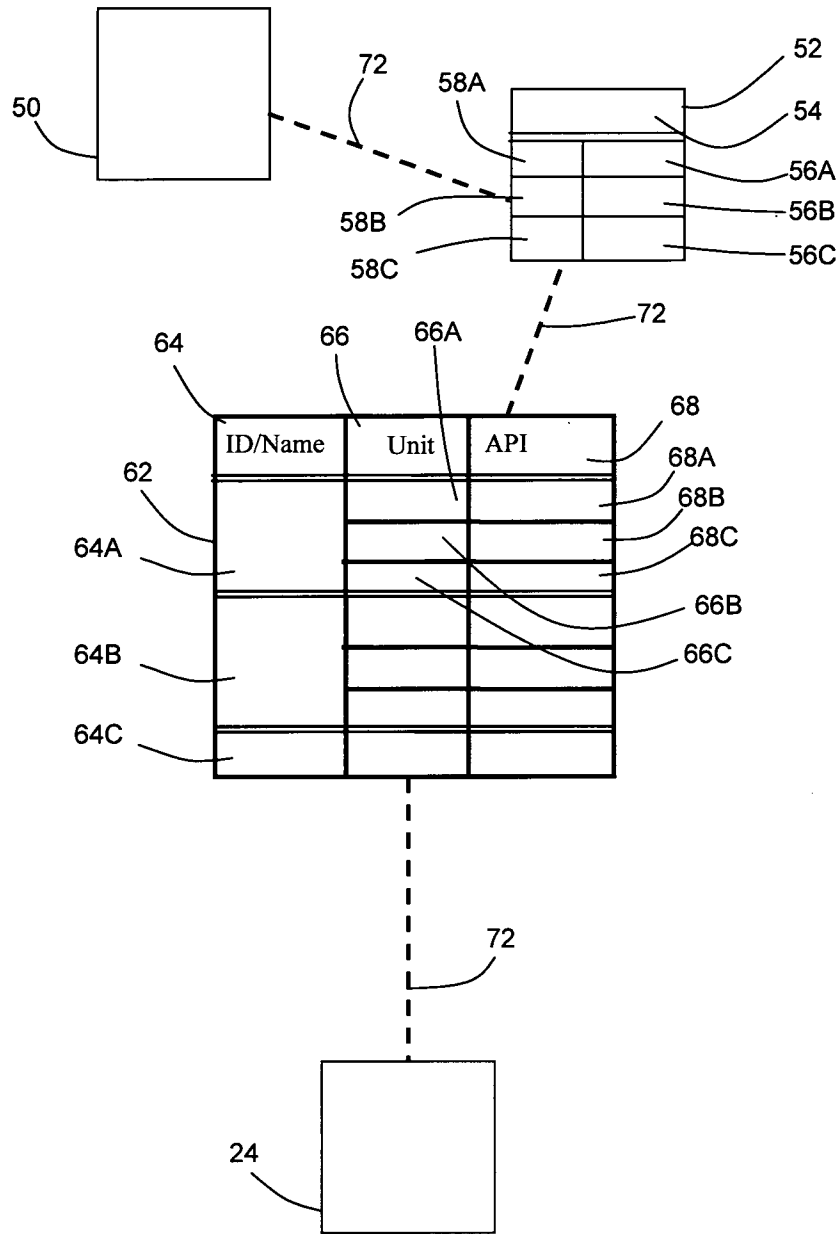


Fig. 4

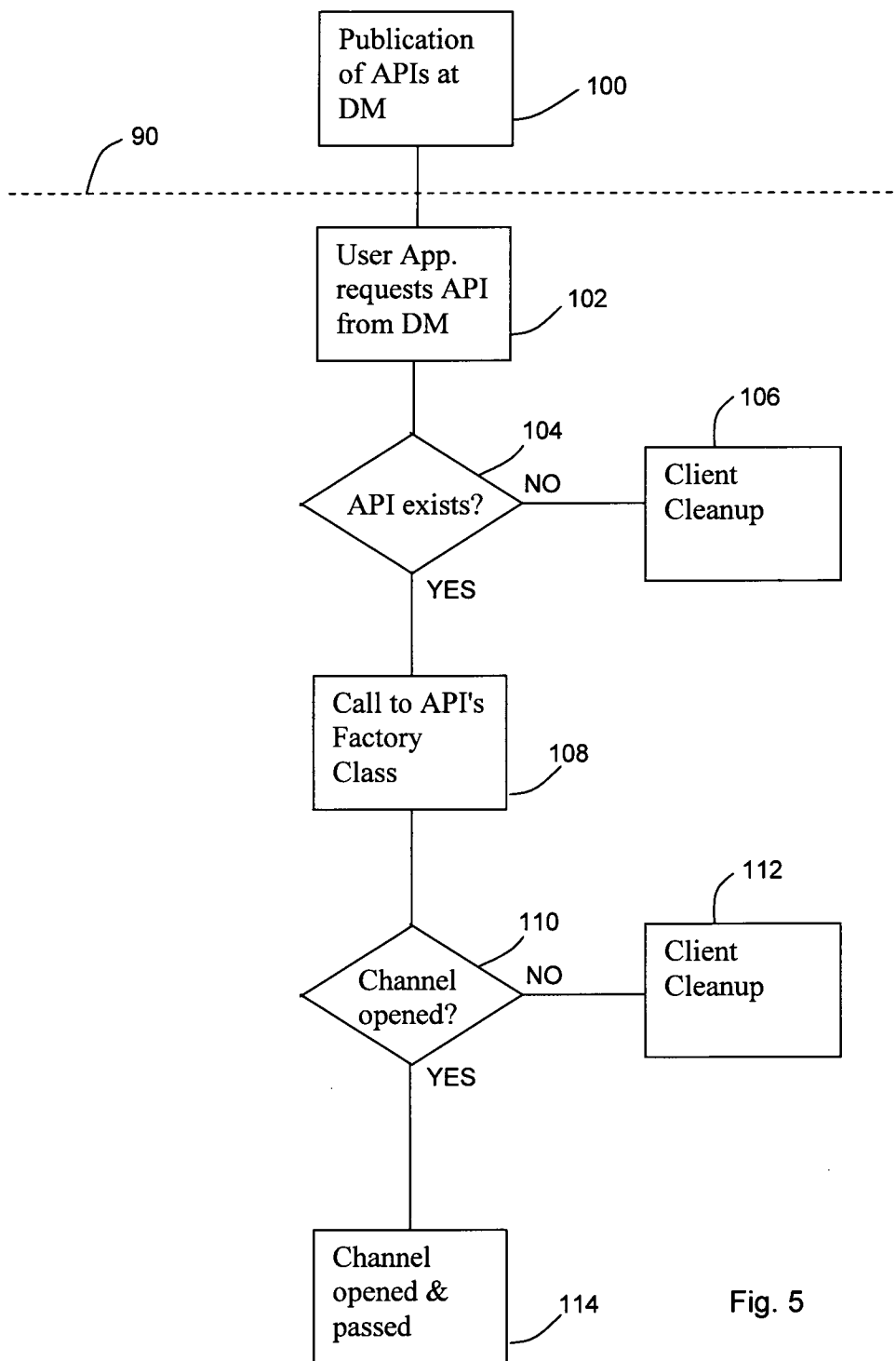


Fig. 5

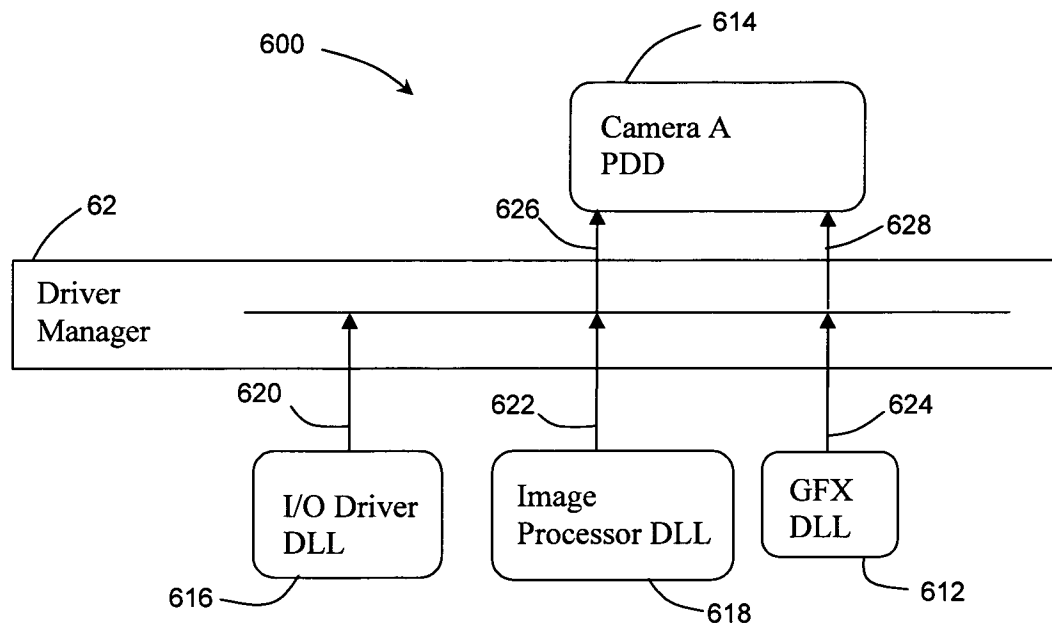


Fig. 6A

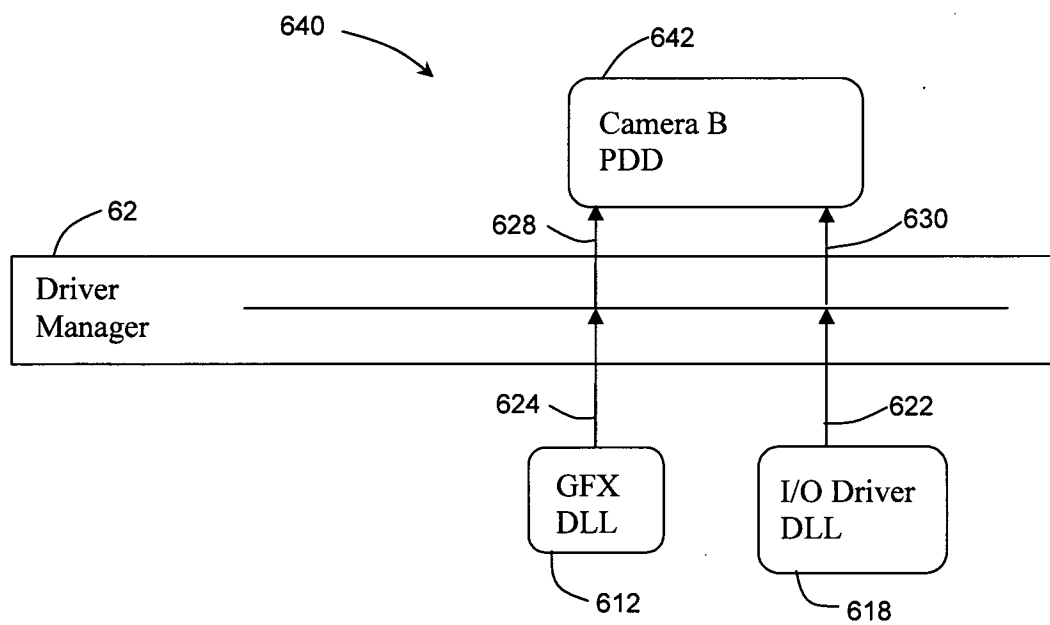


Fig. 6B

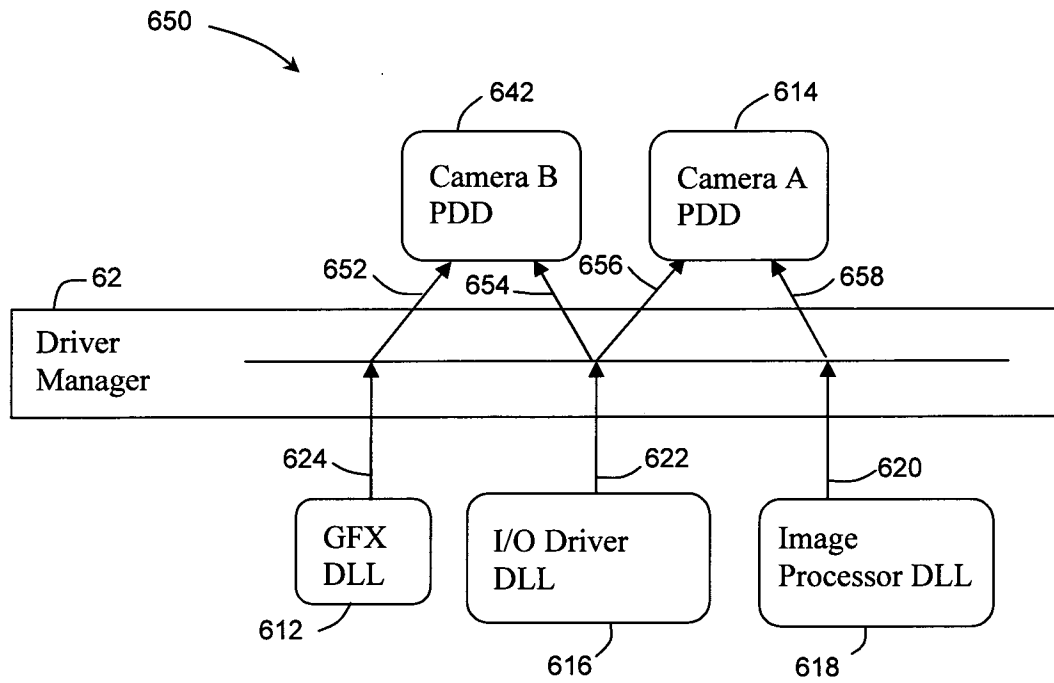


Fig. 6C

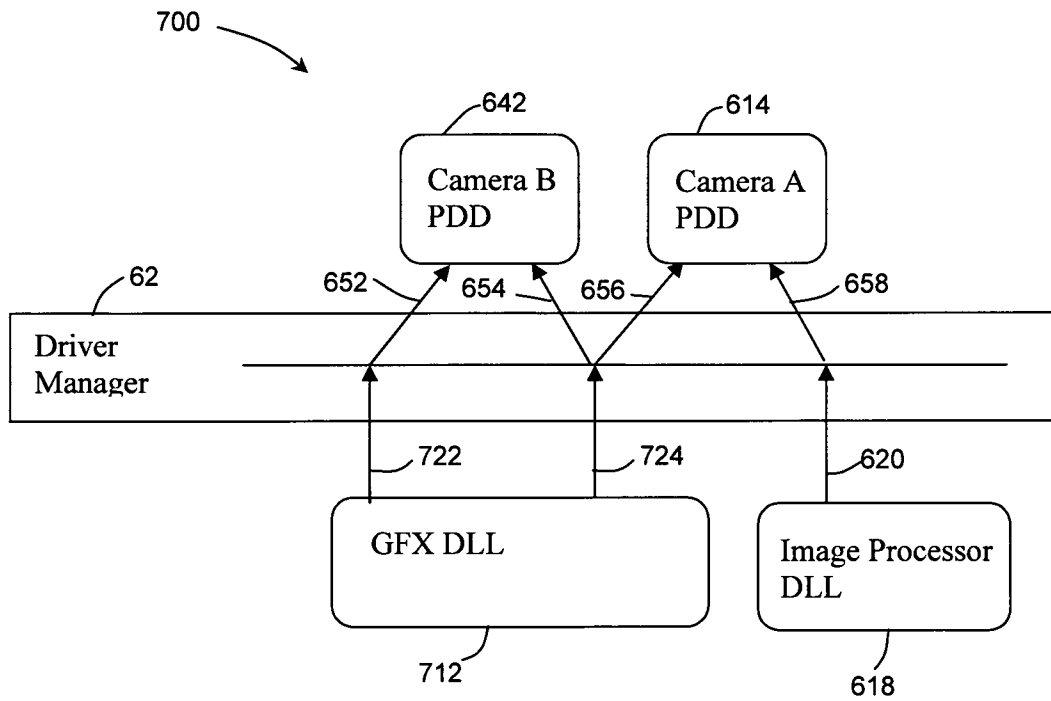


Fig. 7

INTERNATIONAL SEARCH REPORT

International application No

PCT/GB2009/001088

A. CLASSIFICATION OF SUBJECT MATTER
 INV. G06F9/445 G06F13/10

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 practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2004/261084 A1 (ROSENBLOOM OREN [US] ET AL) 23 December 2004 (2004-12-23) abstract paragraph [0004] paragraph [0012] paragraphs [0040] - [0043] paragraphs [0055], [0056] figure 4 claims 1,3,6,14,20	1-26
X	US 2006/026317 A1 (NIIKURA YASUHITO [JP]) 2 February 2006 (2006-02-02) abstract paragraphs [0011], [0012] paragraphs [0036], [0037] paragraph [0047] figure 3	1-26



Further documents are listed in the continuation of Box C.



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 document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *&* document member of the same patent family

Date of the actual completion of the international search

19 August 2009

Date of mailing of the international search report

31/08/2009

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
 NL - 2280 HV Rijswijk
 Tel. (+31-70) 340-2040.
 Fax: (+31-70) 340-3016

Authorized officer

Tomàs Blanch, F

INTERNATIONAL SEARCH REPORT

International application No

PCT/GB2009/001088

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2002/026536 A1 (HYDER JAMEEL [US] ET AL) 28 February 2002 (2002-02-28) abstract paragraph [0002] paragraphs [0013] - [0017] paragraphs [0023], [0024] paragraphs [0030] - [0032] paragraph [0057] paragraphs [0067], [0068] paragraphs [0071] - [0073] claims 1-3,6 -----	1-26

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/GB2009/001088

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2004261084 A1	23-12-2004	NONE	
US 2006026317 A1	02-02-2006	JP 2006039982 A	09-02-2006
US 2002026536 A1	28-02-2002	NONE	