Go-HotSwap

White Paper

Jungo Software Technologies Inc, 555 North Mathilda Avenue, Suite #2 Sunnyvale CA, 94086 U.S.A
Tel: 1-877-514-0537 (USA); 972-9-885-8611 (worldwide). Web: www.jungo.com. Email: info@jungo.com

JUNGO™
Connecting Software to Hardware

Table of Contents

JUNGO™
Connecting Software to Hardware

Architectural Overview

Jungo Go-HotSwap is a software infrastructure that adds the necessary software modules required to enable CompactPCI hot swapping, and provides the tools and development environment to develop hot swap aware drivers.

Go-HotSwap complies with PICMG 2.1 R2.0 requirements for CompactPCI General Use Full Hot Swap Software. Go-HotSwap supports Linux, Solaris and Windows 2000/XP/Server 2003, utilizing the native Plug-and-Play capabilities of these operating systems.

Go-HotSwap also includes driver development tools that enable development of Hot-Swappable[1] drivers from scratch. Go-HotSwap provides the mechanism to allow for CompactPCI Hot Swap - It identifies the insertion/removal of CompactPCI boards; it dynamically allocates the required resources for the newly inserted boards, and de-allocates the resources for those boards that have been removed; finally it notifies the system services and the relevant applications /drivers of the Hot Swap event..

Introduction

Go-HotSwap software is a combination of four components:

Hot Swap Engine – This component serves as a generic hot swap aware driver that resides in the kernel. The Hot Swap Engine communicates directly with the hardware, as well as with the hardware specific driver (directly, or through the Configuration Manager). Its central task is to detect hot swap events, identify the hot swapped board's required resources and (optional) dynamically allocate them or de-allocate them upon removal of the board.

Kernel Mode Messaging Mechanism – This component is responsible for establishing and maintaining an updated log of the Hot Swap events according to the data received from the Hot Swap Engine. It then transfers the relevant messages to the subscribed drivers or applications.

Driver Development Toolkit – This component, if you wish to use it, enables quick and simple development of your driver's hardware access code. The driver development toolkit enables the user to easily produce a user application aware of hot swap events.

[1] Drivers that support the Hot-Swap capability of the device they are driving.

Configuration Manager/hs_activate – This component is an user-mode application which activates and stops the Hot Swap Engine and enables the Hot Swap Engine to respond to hot swap events. The **hs_activate** is a console-mode application suitable for all supported operating systems, while the Configuration Manager is a graphical implementation of **hs_activate**, suitable for Windows only. The user can configure the Configuration Manager/**hs_activate** in order to execute tasks upon hot swap events such as running a batch file or starting/stopping a service, thus achieving hot swap capabilities without modifying the driver's source code.

The following diagram illustrates two possible methods of working with Go-HotSwap:

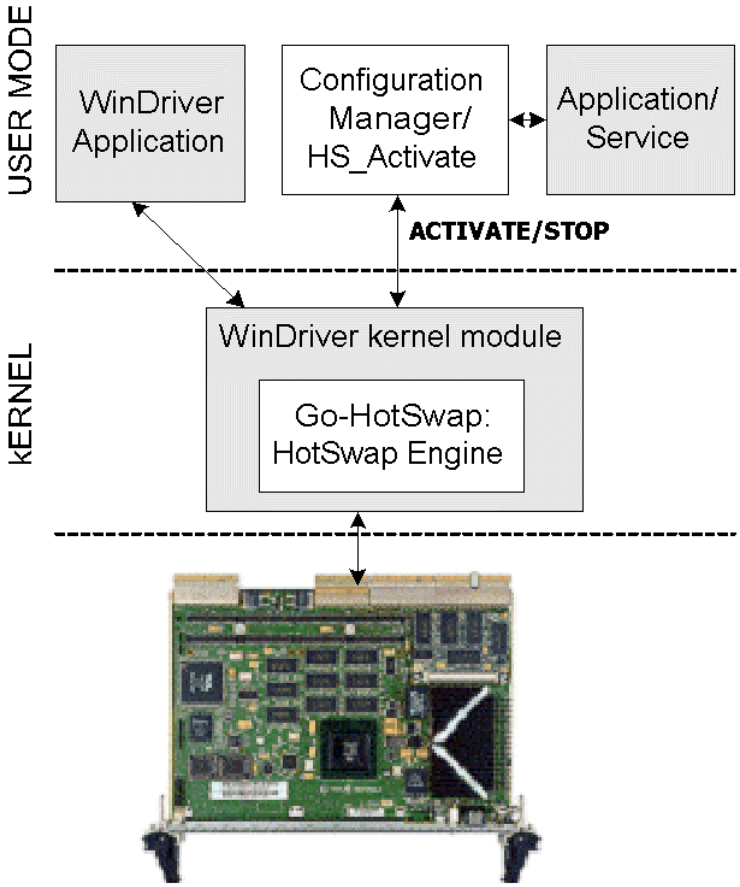


Diagram #1
GO-HotSwap Architecture

In diagram #1, the Configuration Manager/**hs_activate** is responsible for activating the Hot Swap Engine.

The Hot Swap Engine is then responsible for detecting hot swap events and (optional) allocating the boards' required resources. The Hot Swap Engine notifies either the

WinDriver application or the Configuration Manager/**hs_activate** about hot swap events.

The Configuration Manager/**hs_activate** is also responsible for stopping the Hot Swap Engine.

You can use WinDriver to produce the user application, which includes automatic hot swap support. You have the option to use the WinDriver wizard to generate your driver's hardware access code. During this process, you are asked to select your driver's options, one of which is Plug and Play notifications, another is Power Management notifications. Select these options in order to use WinDriver's application with the Hot Swap Engine.

The alternative option is to use a different application/service via the Configuration Manager/**hs_activate**, which notifies/starts/stops the drivers or applications according to the way the user configured the Configuration Manager/**hs_activate**.

Hot Swap Engine

The Hot Swap Engine acts as a generic Hot-Swap/Hot-Plug System Driver (as described in the Hot Swap Specification PICMG 2.1 R2.0). Its main task is to detect hot swap events and identify the hot swapped board's required resources. On Windows platforms you have the option to use the Hot Swap Engine to dynamically allocate the hot swapped board's resources (or de-allocate them upon the board removal).

a. Detecting Hot Swap Events

In order to detect live insertion/removal of CompactPCI devices from the bus, the Hot Swap Engine can use one of the following methods: polling the CompactPCI bus periodically, or poling the ENUM# signal[2].

Polling the CompactPCI Bus

When the polling mechanism is activated the Hot Swap Engine starts polling the bus periodically. This is Go-HotSwap's default mechanism for detection of Hot Swap events, because it allows compliance with all CompactPCI system configurations and board types (including basic and full Hot Swap boards), as the Hot Swap

[2] A signal provided by CompactPCI full Hot Swap boards and by Hot Swap friendly / ready CompactPCI silicone, in order to notify the system host that either a board was freshly inserted or is about to be extracted.

Specification PICMG 2.1 R1.0 allows the ENUM# signal to either drive an interrupt or be polled by the system software at regular intervals, and ENUM# implementation and support requires platform specific support. Moreover, this architecture allows Go-HotSwap to enable Hot Swap with boards that do not implement the ENUM# signal at all.

After initiation, the Hot Swap Engine scans the CompactPCI bus to determine which devices are actually present. To do this, the Hot Swap Engine accesses the PCI Configuration Space and reads the Vendor ID register in each one of the possible 32 CompactPCI slots on each of the CompactPCI buses. (the bus will report an attempt to read the Vendor ID of non-existent devices by returning a value of all 1's). When a board is found, the Hot Swap Engine retrieves additional information, such as:

1. The board's device ID.

2. Whether the board is Hot Swap compatible or not (Basic Hot Swap or Full Hot Swap). If it is, the Hot Swap Engine continues to check:

    a. Whether the status of the Handle Switch (actuated by the lower ejector handle) is closed or open, by querying the Hot Swap Control and Status bits INS and EXT. Any alternations to the handle status detected in the next polling cycle will be used to determine whether the board was inserted (Handle Switch closed), or whether it is about to be removed (Handle Switch open).

    b. Whether the board is a PCI bus master (i.e. a peripheral board containing a bridge that connects the parent bus, on which it resides, to a child bus residing beyond the board, capable of carrying additional boards), by accessing the Class Code Register in the PCI configuration space of the board. If it is, the Hot Swap Engine locates the buses and devices residing beyond that bridge and adds them to the scan list. The same scan scheme will be applied to a child bus as well.

    c. Whether the board is a multifunction device, by querying the Header Type Register in the PCI Configuration Space of the board. If it is, the Hot Swap Engine queries all the board's functions and adds them to the scan list.

At the end of this process the Hot Swap Engine constructs a list of existing buses, boards and functions.

At the next polling cycle the Hot Swap Engine will poll the listed buses, boards and functions, along with the PCI buses, updating the scan list according to changes that result from insertion/extraction of boards, into or out of the system.

From this point onward, the PCI buses will be polled constantly at regular intervals, thus updating and constructing a new scan list whenever a change is made.

Once a new scan list has been constructed, the engine compares the new list with the old one. It sends a message concerning any modification it detected to the Kernel Mode Messaging Mechanism, from where it can be extracted by the relevant processes (user mode applications or device drivers). The Hot Swap Engine will detect one of the following modifications:

The ejector handle of an existing board was opened - the Hot Swap Engine sends a message asserting that the board is about to be removed and adds this data to its database. However, the engine will not allocate this board's resources to a newly inserted board until it has been physically removed from the chassis.

An existing board was physically extracted from the chassis – The Hot Swap Engine sends a message to the Kernel Mode Messaging Mechanism, asserting that the board was removed, and adds this data to its database. The message is sent whether this board was extracted after all subscribed applications and drivers had terminated their access to the extracted board, or before (surprise extraction). The Hot Swap Engine also frees the board's resources to be reallocated when a new board is inserted into the chassis.

A new board was inserted – If this board was found to be Hot Swappable, but its ejector handle is still open, no resources are allocated to it. If the ejector handle of a Hot Swappable board is closed, the engine allocates its required resources (see below) and sends the appropriate message to the Kernel Mode Messaging Mechanism, including vendor/device ID, Bus, Slot and Function.

ENUM# driven

Go-HotSwap enables one to hook into the ENUM# signal in order to detect Hot Swap events. In this case one should not activate the polling mechanism, but instead instruct one's application to listen to the ENUM# interrupt in order to detect the Hot Swap events. Listening to the ENUM# interrupt is done with WinDriver's interrupt handling API. Please refer to the WinDriver user's guide for detailed information.

b. Allocating Required Resources (Enumeration)

In non-hot-swap systems, enumeration only takes place as the system boots. The Go-HotSwap software enables live enumeration under a running system. When a Hot Swap board is inserted into the system, the Hot Swap Engine performs the process of Dynamic Configuration, whereby it allocates system resources to the board[3] while the system is running.

When using the polling method, the Hot Swap Engine will automatically allocate (or de-allocate) the required resources. If using the ENUM# signal to detect Hot Swap events, then the Go-HotSwap function WD_HsEventSend() should be called in order to allocate (or de-allocate) the required resources. Please refer to the Go-HotSwap

[3] see PICMG Hot Swap specification 2.1 R2.0

Function Reference (in Go-HotSwap user's guide) for more information regarding this function.

The allocated system resources are some or all of the following:

I/O, Memory and Prefetchable Memory

The resource allocation process differs in the case of a simple board and a PCI Bus Master.

(a) Allocating I/O, Memory and Prefetchable Memory for a simple board

When a simple board is inserted into one of the slots, the Hot Swap Engine first queries its required resources by accessing the PCI configuration space registers. The Hot Swap Engine queries all 6 BARs (Base Address Registers) of the board, and identifies the required resources of each BAR. These resources could be I/O, memory and prefetchable memory. It then allocates the board's required resources accordingly.

(b) Allocating I/O, Memory and Prefetchable Memory for PCI bus master

When a PCI bus master is inserted into one of the slots, resources should be allocated for the PCI bus master itself, as well as for the child bus and for any board residing on the child bus beyond the bridge. Again, these resources could be I/O, Memory and Prefetchable Memory. The Hot Swap Engine first queries the 2 BARs of the PCI bus master and identifies the resources required by each BAR; it then allocates the board's required resources accordingly. The Hot Swap Engine also allocates resources to the child bus, according to a predefined algorithm. The Hot Swap Engine then scans the child bus to determine what resources are to be allocated for any board located on the child bus. At this stage the engine also sets a range of bus/slot numbers available at the child bus.

Interrupts

If the Hot Swap Engine detects a board that requires interrupts, it assigns them as follows:

1. On x86 machines, if the board was inserted into a slot that is listed on the interrupt routing table located in the BIOS, the engine finds the appropriate register in the APIC and writes the interrupt number to the board.

2. On PPC machines, or if the board was inserted into a slot that is not listed in the interrupt routing table, the engine recursively calculates the interrupt line and then finds the appropriate register in the APIC and writes the interrupt number to the board.

Command register

In order to access the board, the engine sets the command register in the PCI configuration space to a predefined value, as defined in the PCI specification R2.2.

When the allocation process is complete, the relevant drivers/applications can be notified by the Hot Swap Engine of the boards'/ devices' newly allocated resources. This message could be transferred directly to the driver (if a Hot Swap aware driver had been developed), or via the Go-HotSwap Configuration Manager (if a legacy PCI driver had been used without any driver code programming or modifications).

Kernel Mode Messaging Mechanism

Any application or driver that needs to be notified upon a Hot Swap event, detected by the Hot Swap Engine, registers itself to the Kernel Mode Messaging Mechanism. The registration process also includes the set of event criteria that are relevant to this application or device. For example, an application can be registered to receive notifications whenever a CompactPCI board with a vendor ID of 0x1234 is removed. Each registration call will create a separate linked list of events.

As of this point, every Hot Swap event that occurs and meets the predefined set of criteria, will be added to the linked list and sent to the relevant subscriber/s.

After receiving a notification, the subscriber will retrieve additional information from the linked list that was not included in the original notification. In the example above, after the application received a notification that a CompactPCI board with a vendor ID of 0x1234 was removed, it retrieves additional information regarding its device ID and physical location.

Driver Development Toolkit

Jungo offers two types of driver development tools that can be used in order to simplify the difficult task of developing a device driver:

- WinDriver
- KernelDriver

WinDriver is a toolkit for developing monolithic device drivers and KernelDriver is a toolkit for developing standard operating system internal drivers that require hardware

access and that must communicate with the operating system or must be implemented in the kernel.

In most cases a monolithic device driver is required (i.e. a driver that drives a hardware by directly accessing the hardware). This is why WinDriver is an integral part of the Go-HotSwap package.

KernelDriver is not an integral part of the Go-HotSwap package but as a Go-HotSwap user you are entitled to receive Jungo's KernelDriver toolkit. If needed, please contact Jungo and KernelDriver will be provided to you immediately.

WinDriver provides a hardware-access API that is available to any application. The WinDriver API enables the application to access the CompactPCI card's registers, memory ranges, I/O ranges and handle interrupts, via WinDriver's kernel module. The kernel module is a generic device driver that can access any device, as instructed by theWinDriver API. When developing a hot swap aware driver using theWinDriver toolkit, WinDriver's kernel module includes the Hot Swap Engine.

WinDriver features the DriverWizard, a GUI-based diagnostics and driver generation tool environment, which can automatically generate the driver code that is specific to the particular hardware. Another WinDriver feature included is the Kernel PlugIn, which enables performance critical parts of the user-mode code to run in the kernel mode, thereby achieving optimal performance, limited only by system capabilities.

When using theWinDriver wizard to generate your driver's hardware access code, you are asked to select your driver's options, one of which is Plug and Play notifications, another is Power Management notifications. Select these options in order to create a hot swap aware application. Use the Configuration Manager/**hs_activate** to start/stop the Hot Swap Engine, which will then notify the WinDriver application about hot swap events.

NOTE: It is not a requirement to use Jungo's driver development tools in order to utilize the GO-HotSwap infrastructure, but by using WinDriver you will gain the following advantages:

- **Easy Development -** WinDriver enables programmers to create PCI / ISA / EISA / ISA PnP/ PCMCIA / USB based device drivers in an extremely short time. WinDriver allows you to create your driver in "User Mode", in a familiar environment - using MSDEV, Visual C/C++, Borland, Delphi, Visual Basic or any other Win32 compiler. WinDriver eliminates the need for you to be familiar with operating system internals, kernel programming, the DDK or have any device driver knowledge.

- **Multi OS Support -** The driver created with WinDriver will run on Windows 98 / ME / NT / 2000 / XP / Server 2003 / CE, Linux, Solaris and VxWorks - i.e. write once - run on any of these platforms.

- **Cross OS capabilities -** The code generated with WinDriver will run on all the supported operating systems without any modifications.

- **Friendly Wizards -** The DriverWizard (included) is a Graphical diagnostics tool that lets you write to, and read from the hardware, before writing a single line of code. With a few clicks of the mouse, the hardware is diagnosed - memory ranges are read, registers are toggled and interrupts are checked. Once the device is operating to your satisfaction, the DriverWizard creates the skeletal driver source code, giving access functions to all of the resources on the hardware.

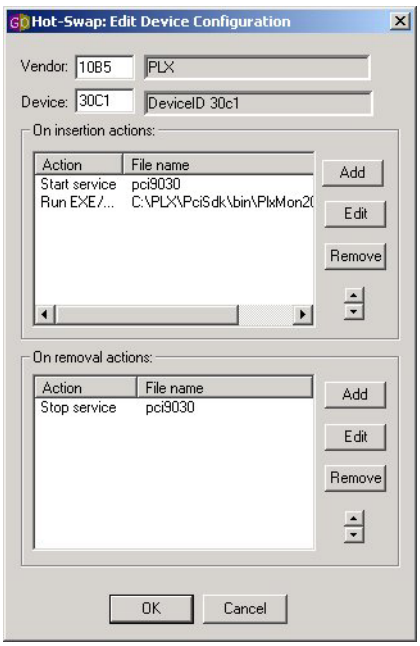
- **Kernel Mode Performance -** WinDriver's API is optimized for performance. For drivers that need kernel mode performance, WinDriver offers the "Kernel PlugIn". This powerful feature enables you to create and debug your code in user mode, and run the performance critical parts of your code, (such as the interrupt handler, or access to I/O mapped memory ranges), in kernel mode, thereby achieving kernel mode performance (zero performance degradation). This unique feature allows the developer to run the user mode code in the OS kernel without having to learn how the kernel works. When working on Windows CE, there is no need to use the Kernel PlugIn since CE has no separation between user mode and kernel mode, thus enabling you to easily achieve optimal performance from the user mode code.

Configuration Manager / HS_ACTIVATE

The **Configuration Manager** and **hs_activate** are user-mode applications, which enable the Go-HotSwap engine to respond to hot swap events. The **hs_activate** is a console-mode application suitable for all supported operating systems, while the Configuration Manager is a graphical implementation of **hs_activate**, suitable for Windows only.

When developing a hot swap aware driver (using WinDriver or any other toolkit), the engine is able to interface directly with the driver within the kernel mode. However, Go-HotSwap enables you to use the Configuration Manager/hs_activate to initiate the engine and interface with the user to receive further instructions.

The Configuration Manager/hs_activate can be pre-configured to do any of the following upon insertion or extraction of a board: start/stop a PCI driver or a service or run any user-mode application or script. These instructions are kept in the Configuration Manager's database, as a configuration file.



Configuration Manager - device configuration screen

The Configuration Manager is not needed whenever a CompactPCI hot swap aware driver is developed with theWinDriver API, other than for activating and stopping the Hot Swap engine. In this case, the Hot Swap Engine data is transferred directly to the newly developed CompactPCI driver.

Summary

The combination of these four components enables Go-HotSwap to add hot swap capabilities for CompactPCI hardware to all major operating systems, and with any available system board.

Go-HotSwap enables users to immediately add hot swap capabilities to their existing PCI drivers or to easily develop hot swap aware drivers from scratch. The resulting code will compile and run on all supported operating systems.

Go-HotSwap also transparently supports any CompactPCI hardware, i.e. basic or full hot swap, as well as silicon friendly and ready.

About Jungo Ltd.

Founded in 1998, Jungo Software Technologies Inc. is a privately held company with corporate offices in San Jose CA and an R&D center in Israel. Jungo's hardware access product line, featuring WinDriver, enables developers to quickly create drivers for custom devices that can run on a multitude of operating systems without modification. Jungo's high-availability product line, including Go-HotSwap, provides a complete solution for the CompactPCI high-availability market.

For additional information please visit Jungo at: http://www.jungo.com

For Go-HotSwap pricing information, please contact sales@jungo.com, or call 1-877-514-0537 ext 3 (USA), +972-9-8850619 ext 3 (Int'l).