Plug and Play For PowerPC^{TM} Reference Platform

Gary Y Tsao
Power Personal Systems Division
IBM Corporation
Austin, Texas 78758
e-mail: tsao@futserv.austin.ibm.com
May 31, 1994

Abstract

Plug and Play is a powerful concept of usability. It is the name of a technology that lets computer hardware and attached devices work together automatically. A customer can simply attach a new device and begin working. This is possible even while the computer is running, without restarting it. Plug and Play is defined in a series of specifications covering the major component pieces. This paper will extract the concepts and apply them to a PowerPC^{TM} platform. The topics will include the hardware, operating systems and firmware.

1. Introduction

Since add-in cards first appeared over a decade ago, they have given PC users a lot of installation headaches. Installation is often a process of trial and error to determine which resources aren't already being used by other cards. The root of the problem is that when the PC was designed, no configuration management capability was defined for ISA bus architecture, nor for integrating the hardware with the operating system software. Alternative bus architectures like MicroChannel™and EISA have hardware and software mechanisms to identify the resources requested by a card and resolve conflicts, however add-in cards for these buses are more expensive and lack the wide range choices of ISA systems. Configuration mechanisms are inherently hard to use and functionally limited because they are not integrated with the operating system. The setup mechanisms require some manual intervention and require users to restart their systems to make any sort of configuration change - a major limitation for mobile computer users.

Plug and Play (PnP) development is being performed by a broad-based group of companies led by Microsoft, Intel, Compaq, and Phoenix Technologies. Even though the architecture is open and independent of specific operating systems and hardware implementations, certain specifications [1], [2], and [3] are PC-oriented. However, the concept and framework design definitedly can be adapted to different platforms and operating systems.

Reference [4] provides the general description of PnP architecture. This architecture defines how each system component - firmware, operating system, device drivers and hardware - performs the auto-configuration process and how these components interact with each other. The basic functions that PnP components must perform are:

- Devices must be uniquely and positively identified
- Their resource requirements must be identified
- They must state the services they provide
- Their drivers must be identified and loaded
- They must allow software to configure or re-configure them

If any hardware change is made to the system configuration, the process must be repeated at run time. There must be a centralized management of this configuration process to ensure complete coordiation. A PnP platform must work with all buses and device types.

There are a variety of changes needed to be addressed when adapting the original PnP architecture to PowerPC™Reference Platform. The PnP architecture consists of new hardware and new software. The new hardware is for the most part defined by new bus technologies such as

PnP ISA, PCI and PCMCIA. With the new technologies, the operating systems must be updated and a new device driver model is required. At the time this paper was written, none of the vendors of operating systems for the PowerPC have released their specification for plug and play, so the discussion here is generic and does not relate to a particular operating system. In the following sections, we will use the term plug and play (pnp) as a generic concept versus the PnP as the original definition.

2. System Resource and Device

System resources include interrupt requests (IRQ), I/O and memory addresses, and direct memory access (DMA) channels. Every computer has a limited number of these resources available. Assigning theses resources to devices without coordination would introduce conflicts between devices. A common problem that prevents ISA systems from booting successfully is the fact that all devices are enabled at power up and the resource requirements of a new device conflicts with those of the primary boot devices.

In a pnp system, each device must be able to uniquely identify itself [5 - 6.1]. PnP ISA has defined a compacted form of necessary information about devices that is suitable for storing in space constrained places like NVRAM. This compacted form, called serial ID, is an EISA Product Identifier. The Serial ID consists of 4-byte vendor ID and 4-byte serial number. The serial number must be unique to differentiate multiple devices of the same type in one pnp system. The vendor ID contains three compressed uppercase ASCII characters and three hexadecimal digits for product number and one digit for revision level. Note that the serial ID is for device or adapter and has nothing to do with the system or cpu ID. The pnp devices need to be registered, possibly within a company first and with some consortium service later. PowerPC systems will reserve four bytes for a device ID. Note that the serial ID is encoded in each PnP ISA adapter hardware [5 - 3.3].

Other buses have similar definitions for device ID but have different formats. For example, PCI defines two-byte Device ID, two-byte vendor ID and one-byte revision ID. PCMCIA defines two-byte manufacturer ID and two-byte PC card ID (in Manufacturer Identification Tuple).

In PnP, a resource is described by a bunch of contiguous tag packets ending with an End tag [5 -6.2]. Two different data types of tags are defined called small items and large items. Currently, there are 11 tags defined for small items:

- Version Number, Tag 0x1
- Logical Device ID, Tag 0x2: To uniquely identify multiple logical devices embedded in a single physical board.

- Compatible Device ID, Tag 0x3: Operating system may use this information to load compatible device driver if necessary.
- IRQ Format, Tag 0x4: Interrupt level and mask description.
- DMA Format, Tag 0x5: DMA channel and mask description.
- Start Dependent Function, Tag 0x6: Group resources that have interdependencies for a logical device.
- End Dependent Function, Tag 0x7:
- I/O Port Descriptor, Tag 0x8:
- Fixed Location I/O Port Descriptor, Tag 0x9: For old ISA cards of 10-bit address decode.
- Vendor Defined, Tag 0xe:
- End Tag, Tag 0xf:

Currently, there are six tags defined for large resource items:

- Memory Range Descriptor, Tag 0x1:
- ANSI Identifier String, Tag 0x2: 8-bit ANSI string.
- Unicode Identifier String, Tag 0x3:
- Vendor Defined, Tag 0x4:
- 32-bit Memory Range Descriptor, Tag 0x5:
- 32-bit Fixed Location Memory Range Descriptor, Tag 0x6:

PowerPC Reference Implementation [11 - 5.6.2] extends the original definition of data items by adding "platform dependent tags". It adds Tag 0xd of small item to describe detailed information about devices, e.g. keyboard, display adapters, displays, diskettes, mouses or modems. It adds Tag 0x16 of large item to describe 32-bit I/O address range. Some examples of the detailed information of devices are: the resolutions of displays or the data rate of a modem.

Reference [11 - 5.6.1] defines per logical device - PPC_DEVICE information as passed by the residual data structure:

DeviceId

BusAccess

AllocateOffset

PossibleOffset

CompatibleOffset

The DeviceId includes Bus ID, four-byte device ID, Serial number, Device Flags, Basetype, subtype, and Interface [2]. BusAccess specifies the device location in a particular bus. AllocateOffset is a pointer to the device heap area of allocated resource description for this logical device. PossibleOffset points to the resource of full range of possibilities before configuration has assigned the final resource. CompatibleOffset points to the compatible device identifier which the operating system may use to load a compatible device driver if necessary.

PPC_DEVICE information can be extracted from NVRAM for legacy ISA devices. The firmware builds PPC_DEVICE information for various systemboard devices, and can also walk through various buses such as PnP ISA, PCI, etc. to build PPC_DEVICE trees and pass them through the residual data structure. The operating system may need to walk through all buses to account for all system's devices.

Some old ISA or even some PnP devices need specific system resources. The configuration manager should grant the system resources to these "static" devices first. Dynamic devices which can be configured to alternative resources are configured later. Configuration or resource managers should follow these procedures to make a final working configuration possible.

3. Boot Time Resource Allocation and Configuration

The primary responsibility of firmware resource allocation is to ensure that the primary boot devices are configured properly to boot the operating system. In general, the primary boot devices are input, output and IPL devices. The firmware may disable any non-bootable devices if they can be disabled to avoid potential resource conflict. For legacy ISA devices which usually can't be disabled, the firmware may need to re-assign the resources of the primary boot (ISA) devices if resource conflict is detected. Besides resolving conflicts with systemboard devices, the firmware may also resolve conflicts with PnP cards during the boot process. Once the system has completed the boot process, the operating system will take over the configuration responsibility.

If systemboard devices can be configured dynamically, this will grant the firmware greater flexibility when alternate configurations are necessary. At run time, this will grant the configuration management software a great deal of flexibility - such as the insertion of a laptop into its docking station - which may need to reconfigure the systemboard devices. Since PowerPC Reference Platform Specification discourages run time call backs between the operating systems and the firmware, the information known to the firmware has to be passed to the operating systems. Residual Data Structures (RDS) [11 5.6.1] serves this purpose for conventional (versus Open

Firmware) firmware. These include at least the systemboard devices and legacy ISA devices plugged into free slots. The firmware can obtain the legacy ISA devices from the NVRAM. There are many ways to put unknown ISA devices into the NVRAM for the first time [6]. For example:

- Run a program like MSD (a configuration utility) that probes the bus for likely cards. If a new device is found, ask the user to verify it and add it to NVRAM.
- Read a .INF file and add to NVRAM.
- Ask the user to fill in the blanks about a device and add to NVRAM.
- Operating system provides a utility, perhaps as part of its device install mechanism, which records the information into NVRAM.
- A user can initiate a session with the configuration utility to input the resource requirements if the system does not detect a new device.

The firmware has to able to finish the boot process even if NVRAM is down (e.g. out of battery power).

Note that the configuration software of individual operating system must have an accurate accounting of the committed resources used in the system. Even if the firmware does a complete job of walking the buses the operating systems are free to do it again. For example, the operating system may want to account for every external SCSI devices but some of them may not power up at boot time. The operating system can walk through the SCSI bus(es) again to get the updated information.

4. Run Time Resource Allocation and Configuration

The Configuration Manager (CM) of an operating system is the system's sole resource manager at run time. The CM must identify every device on the system and its respective resource requirements. Information about each device must be stored by the CM in a central database. The CM then instructs the operating system to load the device drivers for each device.

In the event a resource conflict occurs, the CM resolves this by assigning alternative resources to devices and creates a new working configuration. Once the conflict has been resolved, the CM stores the new configuration information in the central database. If the conflict can not be resolved, the CM instructs the operating system to have an interactive session with the user.

If a change occurs at run time, such as the insertion of a PCMCIA card or a docking station event, the relevant driver notifies the CM of the event so that CM can configure the new device. If reconfiguration is required, CM must store the new configuration information in central database and notify the device drivers of the new resource assignments. If applications register with the CM, CM will signal the configuration changes to applications and the applications may start using the new device or cease using devices that have been removed. This scenario is the typical concept of dynamic event management. One prerequisite of dynamic event management is hot pluggable hardware, i.e. a device can be plugged in or removed while power is on. Hot pluggable hardware includes PCMCIA, docking station or maybe SCSI devices. Dynamic insertion or removal is a pnp system capability which requires special CM software as well as hot pluggable hardware capability.

5. PnP ISA Devices

If the system only uses PnP ISA cards, it is possible to achieve fully automatic configuration. Reference [5] describes a protocol/procedure to accomplish the automatic configuration. The procedure is summarized as follows:

- Each PnP ISA device has a unique device ID even with multiple devices of the same type in one system
- All PnP ISA devices can be put in configuration mode by an "initiation key" sequence.
- The software (bus enumerator) isolates (by using Isolation Protocol) one PnP device at a time and assigns a handle (CSN) to the device. CSN (Card Select Number) is a quick way to select the device.

Note that the bus enumerator has to remember the last assigned CSN.

- The device’s resource requirements are read from the device’s resource data structure.
- Allocate conflict-free resources to each device by resource arbitrator software.
- Enable the PnP devices and remove them from the configuration mode

PnP ISA devices coexist with old ISA devices on the same ISA bus. PnP ISA proposals do not modify the ISA bus. Each PnP ISA device implements extra hardware logics to support the PnP hardware protocols. Each PnP ISA device supports a read-only resource data structure, with the structure supporting multiple functions per ISA device. Each function is defined as a logical device. PnP resource information is provided for each logical device and each logical device is independently configured through the PnP standard registers.

If legacy ISA cards coexist with PnP ISA devices in a system, the resource requirements for the legacy ISA cards (from the NVRAM) have to be granted first. After that, the procedure described above should be followed. In some cases user intervention may be necessary because a resource conflict can not be resolved automatically.

Some bootable PnP ISA devices contain Option ROM. Unfortunately, Option ROM is defined specifically for computing platforms based on the Intel X86 family of microprocessors. Unless a PowerPC based system is able to emulate the X86 code, Option ROM is useless. In such an event, the firmware knows the device is IPL-ABLE (from [5] Logical Device ID tag, byte 5, bit[0]) but where can it get the driver to boot the device? One solution is using the diskette (or any bootable device) as a firmware extension. The card vendor builds a diskette including the code to emulate the firmware plus the driver code to access the device. If a user needs to boot from the ISA device, he/she will insert the diskette and boot from it. The firmware boots the diskette and reads the firmware emulation and driver codes and then boots from the ISA device.

The IEEE P1275 "Open Firmware" defines a neutral F-code which is processor and platform independent. This is the strategic direction of PowerPC Reference Platform. The messy problem of platform dependent boot code as described above can be completely solved by the Open Firmware.

6. Other Buses

The current PCI bus architecture already meets most of the pnp functionality. PCI devices use a standard identification scheme and a mechanism to declare their resource requirements. The PCI Bus Enumerator (BE) or Manager is a new type of driver to perform the isolation and identification of PCI devices and their resource requirements. This BE is a special driver to build the hardware configuration tree for both PCI devices and PCI bridges. The BE is generally registered with the CM so the CM can communicate with the BE. This interface is operating system dependent. The BE performs bus-walking and configures the PCI devices as instructed by CM.

Current PCMCIA cards must include the manufacturer identification and configuration tuples for unique identification and proper configuration. The PCMCIA BE may leverage existing Card Services and Socket Services drivers to perform the walk-through and configuration. The Card Services must be modified to coordinate with the CM instead of a stand-alone configuration component.

EISA and Micro Channel devices already provide a standard identification mechanism and a way to configure their resources through software. To integrate with the pnp architecture, a BE for each bus is required to make configuration information accessible to the CM.

Pnp SCSI devices may require design changes [8]. The SCSI adapter configuration is managed by a BE. The auto-configuration of SCSI bus itself, such as terminating both ends of the SCSI bus and setting SCSI device IDs, is not defined by SCSI-2 specification. The SCSI-3 [7] has voted to include a protocol, SCSI Configured AutoMagically (SCAM) [9], to automatically assign device ID's. The SCAM protocol is designed so that legacy SCSI devices can be detected and used. If more than one legacy device is present, the user must ensure that they do not have SCSI ID conflicts. The new SCSI BE has to communicate with the SCAM masters to obtain the SCSI configuration information.

Other type of devices such as IDE, ECP, serial or infrared can participate in the pnp architecture as long as they provide the necessary mechanisms for identifying and configuring the device.

7. pnp Device Drivers

There are new requirements for device drivers under pnp system. Device drivers must be dynamically loadable and unloadable, to enable reconfiguration and make most efficient use of system memory. Device drivers must register with CM when they are first loaded, remain inactive until they are given their resource assignments, and may be able to communicate with applications to respond to dynamic events. The pnp interface between device driver and CM is operating system dependent.

Device drivers for pnp will need to change their initialization sequence to dynamically determine their configuration. A typical sequence is as follows:

- When a driver is loaded, it issues an operating system call to verify the existance of a CM or configuration utility. If the CM exists, the driver registers with it.
- The driver calls the CM with device ID to receive a valid configuration.
- The driver parses the returned configuration structure, determines the assigned configuration and completes the initialization sequence.

For pnp devices, the BEs are sufficient to identify all the devices. Probing (searching) port addresses as performed by a traditional ISA device driver is unnecessary. The CM is solely responsible for providing system resources to the devices. For static pnp devices which require specific sources, the BE and the CM have little to do other than to verify the requested system resources are valid and available.

8. A Complete pnp System Example

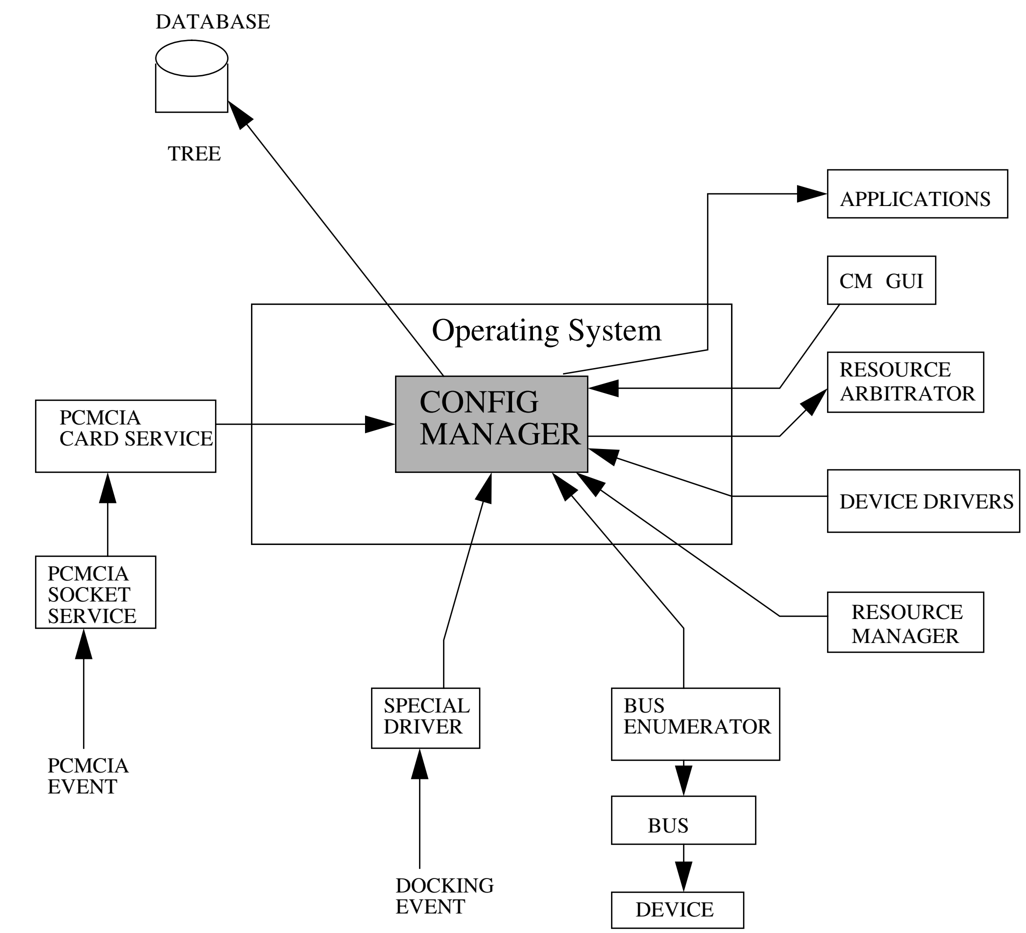

FIGURE 1

Figure 1 is an example of a complete pnp system. The operating system requires several new drivers to support pnp functionality.

The Configuration Manager can be implemented as part of an operating system kernel or as a special driver. It is a piece of software to control the configuration process and coordinate with all other configuring components. The CM is initialized when the operating system receives some device configuration list from the firmware. The CM builds the hardware tree (for current system configuration) by calling the BEs to walk through their specific buses. Some tree information can also be drawn from the central database. For each device, a driver is loaded and waits for assignment of resources by the CM. The Resource Arbitrator allocates resources to each device and , in the event of a conflict, performs an interactive process of reconfiguration through the CM-GUI (Graphics User Interface) until a working configuration is reached. Device drivers are then initialized and get their configuration resources from the CM. The GUI provides user assistance in sorting out potentially complex configuration failures and error message deciphering. It also allows the user to view system resource assignments and specific device configurations, as well as the available system resource pool.

The above process is reinitiated when a "dynamic event driver" informs the CM of an event that requires a change to the system configuration.

In this example, PCMCIA Card Services is registered with CM. Card Services interacts with CM to get configuration information. PCMCIA device drivers then use the Card Services interface to allocate/deallocate resources. The PCMCIA Socket Services serves as a "dynamic event driver" for dynamic insertion or removal of a PC card.

As another example, a special docking station driver is registered with the CM to provide an event handler entry point address to which CM will issue synchronous event message notifications. The CM has an event interface which enables it to recognize an event. The implementation of event notification can be handled through polling or asynchronous (interrupt) notification. When a docking even (such as the removal of a notebook from a docking station) is notified by the driver, the CM recognizes the fact and dynamically releases the resources.

In some operating systems, there are several subsystem resource managers which take autonomous ownership of certain system resources. In a pnp system, these resource managers may need to interact with the CM. For example, with the advent of new devices such as PCMCIA memory cards, system memory may come in as non-traditional packages which require some assistance in setup by CM prior to being usable by the Resource (Memory) Manager.

Application programs can also register with the CM and participate in the dynamic event activities. The API between CM and applications is operating system dependent.

9. Partial System Benefit

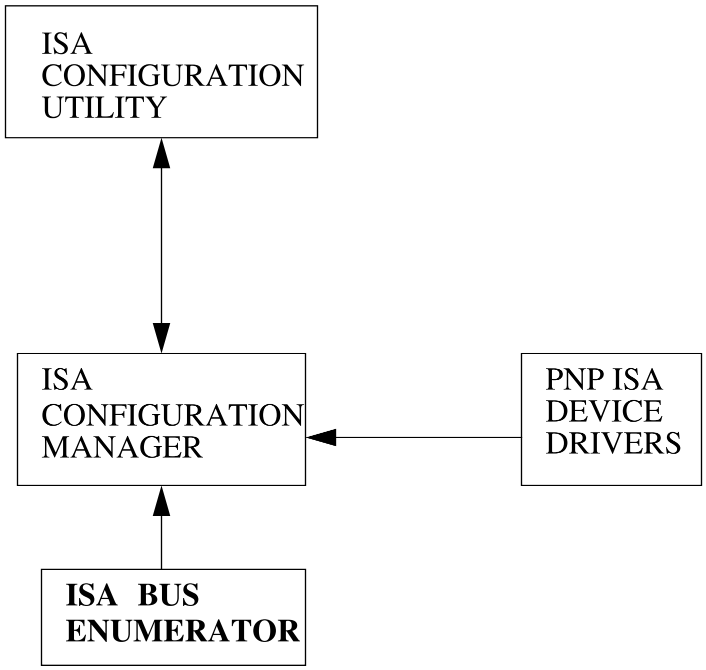

FIGURE 2

Figure 2 shows a non-pnp operating system with a separate run-time configuration utility (CU). This can be useful when a pnp operating system is unavailable. The CU [10] contains configuration information for many legacy ISA device drivers. It isolates the device and the device's resource by a heuristic probing algorithm. If the device matches one in CU's database, the CU identifies the device successfully. If this fails, CU interacts with the user to get the resource requirements. The CM is not a part of the operating system and is implemented as a special driver. In this system, some benefits of Plug and Play can be realized. If a new PnP device is plugged into the system, the CM can automatically configure the new device.

There is one catch to this approach; the interface between PnP device drivers and the CM does not necessarily conform to any operating system. This non-portable approach discourages vendors from writing drivers for this implementation.

10. Conclusion

Plug and Play is an integrated architecture of operating system, device drivers, firmware and hardware. The usuability feature make computer easier to use and helps to reduce customer support costs. For mobile computer users, the pnp dynamic event support is a must. Without that, users must restart their systems to make any sort of configuration change. To prove the importance of ease-of-use to users, we predict the migration of PCMCIA cards from notebook/subnotebook segments to desktop computers.

The capability of configurable devices also carries the promise of power conservation savings by leaving the device powered off until needed. This concept of "power configuration" can be another property of a device. By integrating these power configurations with conventional device configurations, a seamless energy efficient system can be constructed.

Most new buses such as PCI, PCMCIA or PnP ISA have well-defined mechanisms for devices identification and their resource requirements. The pnp has support from major industry players and it costs little (in hardware) to implement. Though no operating system is truly Plug and Play yet, next generation operating systems will definitely include this as a standard feature to improve the ease-of-use of personal computers.

11. Trademarks

PowerPC and MicroChannel are trademarks of International Business Machine Corporation.

12. Reference

[1] Plug and Play BIOS Specification, Version 1.0A, May 5, 1994.

[2] Plug and Play Device Driver Interface for Microsoft Windows 3.1 amd MS-DOS, Version 1.0c, October 5, 1993.

[3] Interfaces Specification for MS DOS and Windows Run-Time Configuration Services, Revision 0.99F, April 1, 1994.

[4] The Plug and Play Framework: Advancing the PC Architecture Backgrounder, Microsoft Corporation, September, 1993.

[5] Plug and Play ISA Specification, Version 1.0a, May 5, 1994.

[6] W. Daniel Private communications.

[7] SCSI-3 Parallel Interface [X3T9.2/855D]

[8] Proposal for Plug and Play SCSI Specification, Version 0.95, October 14,1993

[9] SCSI Configured AutoMagically, X3T9.2/93-109r5, SCSI BBS 719-574-0424

[10] PC Magazine, Plug and Play: Easy Upgrades At Last, March 15, 1994

[11] PowerPC Reference Platform Specification, Version 1.0, June, 1994.