Distributed Hot Swap management in an embedded system

By Chris Brand and Geoff Holt

Many modern embedded systems have a need for minimal downtime due to failures, maintenance, or upgrades, that are addressed by permitting insertion and removal of boards while the system is active. Both CompactPCI and VME systems support live insertion and/or removal of boards called "Hot Swap". It is important that notification of such a change in configuration is communicated appropriately throughout the system so that it can react properly, perhaps by redistributing work between the available resources or shutting down any direct communication between boards before the boards involved are physically removed from the system. In a CompactPCI system, the notification takes the form of an interrupt to a System Host that takes the appropriate action. However, not only does the System Host constitute a single point of failure, but in many systems where there is no other traffic over the CompactPCI bus, handling this interrupt may be the only reason for the System Host to be present at all. Elimination of this centralized control function would therefore lead to a cheaper, more robust and optimal system architecture.

In an embedded multiprocessor system, entities that need to know about insertion or removal events may be distributed throughout the system. Such entities may care about Hot Swap events in one of two ways. They may be interested in knowing about a Hot Swap event within some defined time of it having occurred, or they may need to know about an impending Hot Swap event and take some action before it occurs. In the former case, asynchronous notification of completion of the event is sufficient, however the latter case demands synchronous action on the part of the interested entity. The mechanism used for both cases should ideally provide complete decoupling of interested entities and sources of Hot Swap events and achieve an acceptable response time so that completion and notification of events is not unduly delayed.

This article discusses how a modified version of the Real-Time CORBA Notification Service may be used to meet these requirements.

CompactPCI Hot Swap

CompactPCI Hot Swap, as defined in the Hot Swap Specification[1], introduces the ability to insert or remove a peripheral board (any board other than the System Host) from a running system. Removal is a two-phase process. The first phase occurs when the lower latch on the board is opened, this generates an interrupt notifying the System Host that removal of the board is imminent. The Host then has an opportunity to prepare the board before initiating the second phase by lighting a blue LED on the board, which indicates that it is now safe to fully extract the board. Hot Swap thus increases the availability of CompactPCI systems by allowing the removal and replacement of peripheral boards without the need to power down the entire chassis. Unfortunately, the System Host is still a single point of failure. If the System Host fails or needs to be upgraded, the entire chassis must still be powered down in order for it to be removed and replaced.

The Hot Swap specification imposes certain requirements on the software running on the System Host. In particular, when detecting that a peripheral board is about to be removed, the System Host is required to ensure that the board being extracted no longer accesses the PCI bus, and that the board is not accessed from the PCI bus. This is necessary to ensure the integrity of signals on the bus as the board is removed. This requirement is reasonably easy to meet when all communication is between the peripheral board and the System Host, but significantly harder if there is direct communication between peripheral boards.

An alternative approach

In many CompactPCI systems, peripheral boards have a reasonable amount of intelligence. They may have fully functional CPUs and even run Operating Systems. For these boards, it is natural that they communicate directly with one another.

In systems with a large amount of data to process in a short period of time, the bandwidth of the CompactPCI bus may not be sufficient. This is exacerbated by the fact that the available bandwidth is shared between all the peripheral boards and the System Host.

In such a system, it may make sense for peripheral boards to communicate directly via some other medium, quite possibly in a point-to-point manner. This is the impetus behind PICMG 2.16, the forthcoming PICMG 2.18, and other related standards that introduce alternative communication links between peripheral boards. In systems such as these, the CompactPCI backplane may be little more than a source of power and a clock. A clock signal can easily be generated on the peripheral boards if it is not present on the backplane but is needed for a local PCI bus on the board. This leaves the non-redundant System Host with just one job, supporting Hot Swap of the peripheral boards.

Removing the system host

If the work involved in handling Hot Swap events could be distributed to the peripheral boards, the System Host could be removed completely, eliminating the single point of failure.

©Copyright 2003

In a CompactPCI Hot Swap system, peripheral boards have a micro-switch attached to the lower board latch. When the latch changes state between open and closed, the System Host is interrupted and can deal with the event. A Hot Swap Control and Status Register in configuration space of the device are used to communicate the details of the event. With a CPU on the peripheral board, it is relatively simple to implement similar functionality entirely on the board itself. Opening or closing the latch generates an interrupt to the CPU on the peripheral board. That CPU can query the hardware to determine exactly what happened and how to respond. It is even possible to implement a register that behaves exactly like a Hot Swap Control and Status Register, in which case the software that runs on the local CPU to handle Hot Swap events may be exactly the same software that usually runs on the System Host. This allows each peripheral board to independently detect that it is about to be removed.

In a system with no System Host, there can be no communication over the CompactPCI backplane, so there is nothing there that needs stopping. There may of course be other communication links that need to be moved to a quiescent state however. There is a need to handle insertion as well as extraction. An insertion may look exactly like a power-up, the board moves from an un-powered state to one where it has power, but there is also the possibility that the latch was opened, causing the extraction processing to be performed, and then re-closed without actually extracting the board and causing it to power-cycle. This change in state needs to be handled in much the same way as when the board first gets powered up.

It is possible to design a board that detects if there is a clock on the CompactPCI bus. If it detects such a clock, it should meet all the normal CompactPCI requirements. If it does not detect such a clock, it can assume that the CompactPCI backplane is not being used for communication and that there is no System Host present. In this case, the local CPU can take over the handling of Hot Swap management tasks as described previously.

Types of interaction with Hot Swap events

In an embedded system with no centralized control, entities that care about insertion or removal events (Hot Swap event consumers) may be distributed on different heterogeneous processors throughout the system.

Such consumers may care about Hot Swap events in one of two ways. They may merely need to react to a Hot Swap event after it has happened. For example different consumers may need to:

- Initialize hardware that has been inserted.
- Instantiate proxies for hardware that has been inserted.
- Update the system state on a user interface.

Although an asynchronous mechanism is sufficient, in fact desirable for these types of notifications, it still needs to be efficient enough to achieve acceptable response times.

Alternatively they may need to be directly involved in an impending Hot Swap extraction event in order to perform some processing before the extraction actually happens. For example different consumers may need to:

- Prepare them and save their state if they are physically located on the hardware to be removed.
- Shutdown communications if they have a direct connection to the hardware to be removed.
- Reconfigure the application if they are acting in some type of system supervisory role.

Clearly a synchronous mechanism is required to notify all directly involved entities in a timely manner and ensure that they have completed their processing before extraction is allowed to proceed.

General requirements for decentralized Hot Swap event notification

In order to support decentralized Hot Swap in a heterogeneous computing environment, an event notification mechanism is required that:

- Provides both efficient synchronous and asynchronous notification services (while not required, it is desirable that these services are accessed via a common interface).
- Supports diverse target processors.
- Supports location transparency so that events are distributed identically irrespective of whether they are local to the hardware in question, within the same chassis, or somewhere else in an arbitrarily large system.
- Provides loose (i.e. determined at run-time) coupling between sources and consumers of Hot Swap events (since system configuration is by definition changing on the fly).
- Is standards-based in order to maximize re-use and portability.
- Offers redundancy to remove sensitivity to individual failures.

The chosen solution

A solution based on the Object Management Group's (OMG) CORBA Notification Service[2] is chosen for the following reasons:

- It is based upon the Common Object Request Broker Architecture (CORBA) specification[3] that is a well-supported middleware industry standard that offers an architecture, language, transport, and vendor independent communication in a location transparent manner.
- It provides an efficient asynchronous event distribution mechanism that offers advanced quality of service (QoS) and filtering properties (that were not present in its predecessor the CORBA Event Service). The QoS properties are used to control reliability, priority, and timeliness of event distribution, while the filtering mechanism makes efficient implementation possible by suppressing unnecessary distribution of all events to all consumers.
- It provides the required loose coupling by means of a publish/ subscribe model that sources and consumes registers independently with a well-known Hot Swap Event Channel. Therefore requiring no direct reference to each other.
- Provided the CORBA implementation supports it, the Hot Swap Event Channel may be implemented as a redundant object in accordance with the Fault Tolerant Specification[3].

The Notification Service does not provide a synchronous event distribution but its interfaces and semantics may be extended to provide such a mechanism as discussed next.

Implementing a Hot Swap Event Channel

During system initialization a single Notification Service event channel is instantiated. This Hot Swap Event Channel registers itself at a well know location within the CORBA Naming Service.

Any consumer that wishes to be notified of any kind of Hot Swap event locates the Hot Swap Event Channel by means of the Naming Service and registers it to receive Hot Swap Events (Figure 1). The filtering mechanism provided by the Notification Service allows a consumer to specify only the exact kinds (Insertion, Insertion_Error, Pre_Extraction, Post_Extraction, and Channel_Shutdown) and sources of Hot Swap events that it cares about. This eliminates the overhead of the service distributing and the consumer ignoring events that are of no interest to it.

For example, a consumer that needs to perform some processing before a particular piece of hardware is extracted, needs to register for the Pre_Extraction event from that particular source, while a consumer that needs to be notified of system configuration changes would register for the Insertion and Post_Extraction events from all sources. The Insertion_Error event is generated when hardware is inserted but is unusable for some reason (for example a self-test failure). Some consumers might register for all events, and then the Hot Swap Event Channel will call the consumer back when an event that it is interested in (i.e. matching its filter) occurs.

Sources of Hot Swap events are typically service routines responding to the Hot Swap interrupt on processors residing on removable hardware, however the System Host in a CompactPCI chassis that contains only dumb peripherals may also be a source distributing events to a wider super-system. It is also possible for a device (such as a system health monitor) in the system to inject Hot Swap events for boards on which it detects a failure. When a Hot Swap source has an event to distribute, it locates the Hot Swap Event Channel in the Naming Service and calls the Channel with the event.

Thus a push model for event distribution is employed where sources push events to the Channel that in turn pushes the events to interested consumers. Structured events as defined by the Notification Service are used since they provide filtering support.

Asynchronous event distribution

Asynchronous distribution is used whenever a source has an Insertion, Insertion_Error, Post_Extraction, or Channel_Shutdown to report. This is the normal fire and forget mode of operation of the Notification Service that the push call from the source returns immediately. The Channel thereafter passes the event through the applicable filters and distributes it to all interested, registered consumers (Figure 2). In an efficient implementation, multiple threads are used to notify interested consumers simultaneously[4].

Synchronous event distribution

In order to support synchronous participation in Pre_Extraction events, an extension to the Notification Service is needed to provide synchronous behavior. The semantics of this extension otherwise match the standard Notification Service specification.

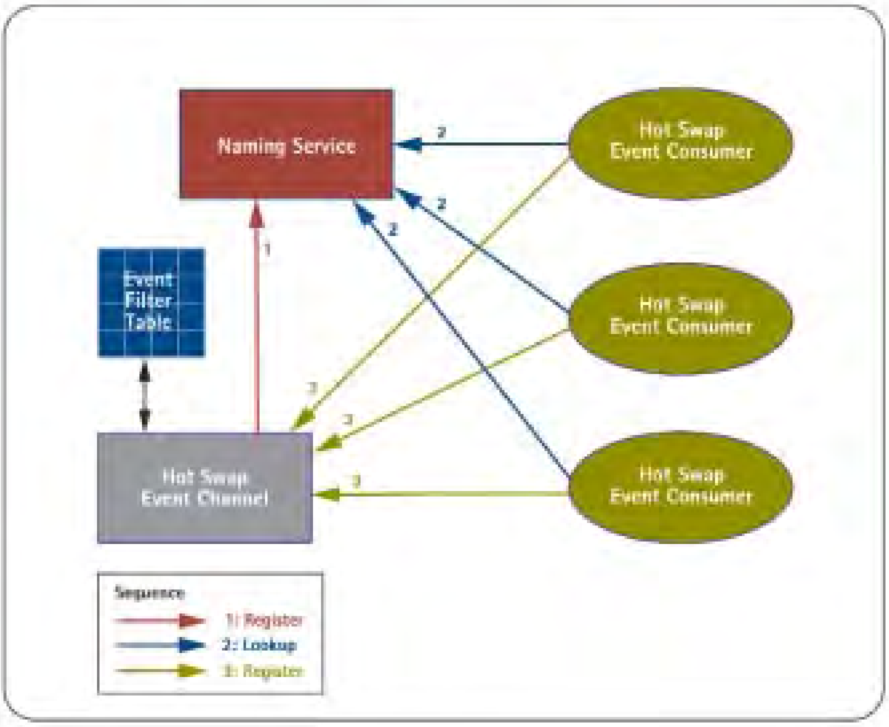


Figure 1

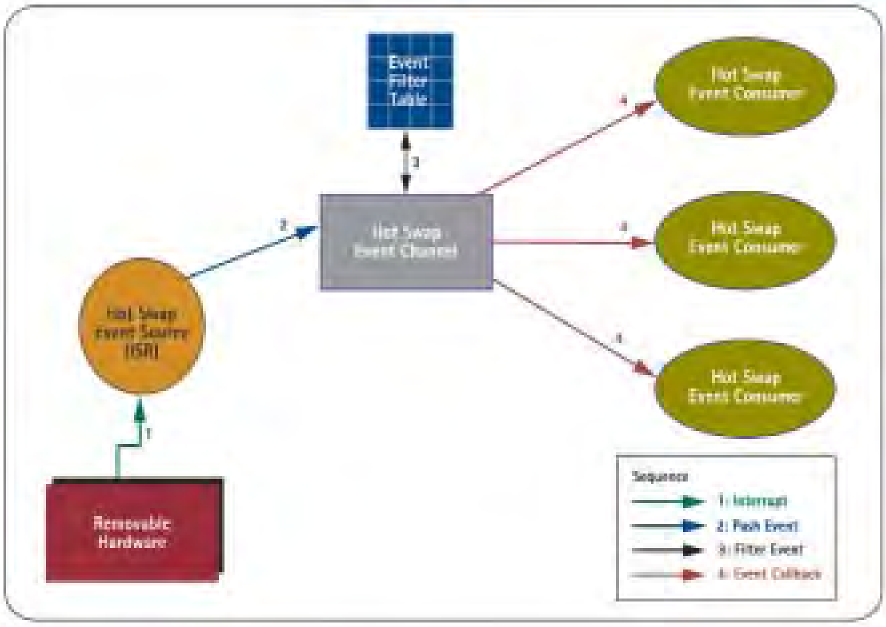


Figure 2

©Copyright 2003

A synchronous event is generated when a source calls the push operation on a newly defined synchronous interface on the extended Notification Service. This event is distributed in exactly the same way as an asynchronous event, except that the source is called back when all consumers have completed their processing (or otherwise failed or timed-out) with an indication of the success or reason for failure of the total operation (Figure 3). This permits the source to wait for completion of the event distribution before continuing or alternatively implement its own timeout if no response is received.

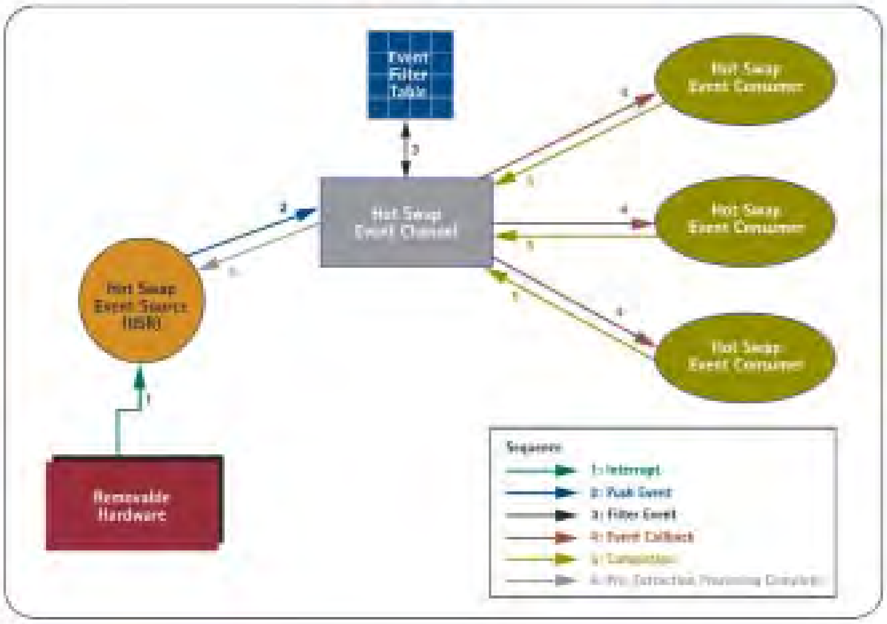


Figure 3

Extraction processing

Thus each extraction that is initiated first generates a synchronous Pre_Extraction event notification to ensure that all required pre-extraction processing is complete, followed by an asynchronous Post_Extraction event notification prior to lighting the blue LED for extraction to complete.

Fault tolerance

One of the stated advantages of eliminating the need for a System Host from a CompactPCI bus was the removal of a single point of failure from the system. However, when using the extended Notification Service, all events must pass through a Hot Swap Event Channel that must reside on a single processor that then again becomes a single point of failure.

This problem is addressed by the Fault Tolerant CORBA specification. By providing for distributed redundant replicas of CORBA server objects and mechanisms to switch between these replicas to deal with failures, the problem is solved. This is completely transparent to users of the service. Therefore, if required, the Hot Swap Event Channel can be implemented as a Fault Tolerant object to provide additional system reliability without impacting Hot Swap event sources and consumers. The Hot Swap Event Channel meets all the above requirements for a decentralized Hot Swap control mechanism.

Summary

With the aid of appropriately configured middleware, it is possible to distribute Hot Swap control among intelligent computing devices in a CompactPCI chassis, thereby eliminating the need to centralize this function on a System Host. Moreover, events may be distributed throughout a super-system consisting of multiple chassis both with and without System Hosts. Providing both reliability and system management benefits.

An enhanced implementation of the CORBA Notification Service provides a suitable middleware platform to support distributed Hot Swap control.

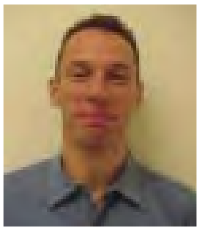
References

1 CompactPCI Hot Swap Specification PICMG 2.1 R2.0 January 17, 2001

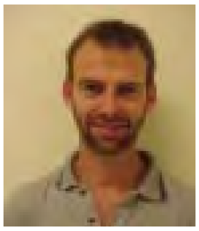
2 Object Management Group, Notification Service Specification, Version 1.0.1, OMG Document formal/2002-08-04

3 Object Management Group, Common Object Request Broker Architecture (CORBA/IIOP), Version 3.0.2, OMG Document formal/2002-12-06

4 Pradeep Gore, Douglas C. Schmidt, Carlos O'Ryan, and Ron Cytron, "Designing and Optimizing a Scalable CORBA Notification Service", Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001



***Chris Brand** is a software technical lead at Spectrum Signal Processing. He also teaches Real-time Systems, part-time at the University of British Columbia and Computer Systems Technology Foundation at the British Columbia Institute of Technology. He received a B.Sc. (Hons) from the University of Reading, England and a Post-Graduate Diploma in Advanced Computing from Bournemouth University, England. He is a senior member of the IEEE, a member of the IEEE Communications Society, IEEE Computer Society, and a member of the Association of Computing Machinery (ACM).*



***Geoff Holt** is a senior software engineer at Spectrum Signal Processing Inc. With ten years of experience in the field of real-time systems, his areas of expertise are in embedded communications infrastructure and middleware for distributed systems. Geoff has a Bachelors and Masters in Engineering from the University of the Witwatersrand, South Africa.*

For further information, contact Chris or Geoff at:

Spectrum Signal Processing
200 – 2700 Production Way
Burnaby, B.C., V5A 4X1
Tel: 604-421-5422
Fax: 604-421-1764
E-mail: chris_brand@spectrumsignal.com
E-mail: geoff_holt@spectrumsignal.com
Web site: www.spectrumsignal.com

©Copyright 2003