ADVANTECH White Paper

An Integrator's Guide to MicroTCA
Part III – System management software

By Peter Marek, Advantech

In the previous article of this series we explained the basic processes of the start up of a Field Replaceable Unit (FRU) using the example of the insertion of an AMC module into a MicroTCA chassis.
In this third part we will discuss system management software and the various implementation options available.

What is this carrier, shelf and system manager stuff, anyway?

Throughout this series of articles we have used the above terms quite frequently starting with the µTCA management hierarchy summarized in Part 1. Here we will clarify the relationship between management entities in order to describe how management software is implemented.

Carrier Manager

This is a firmware component executed on the MCH. It provides all the low level control of everything that is used in the carrier. So what is the carrier? Basically think of the carrier as the backplane and everything plugged into it: AMCs, MCHs and power modules. The best way to understand these items is to recall that the µTCA standard was derived from ATCA. The carrier in µTCA represents the same entities that are present on an ATCA AMC carrier. Table 1 provides a helpful summary of how those entities compare.

µTCA Carrier	ATCA AMC Carrier
MCMC on MCH	Carrier Management Controller (CMC)
Power module	Local DC/DC converter and hot swap power switching for AMCs
Backplane with AMC slots	Onboard AMC sites
Backplane connectivity	Point to point connectivity on carrier board
Carrier Manager (firmware)	CMC firmware

Table 1: µTCA Carrier and ATCA AMC Carrier Board

The carrier manager handles the following tasks:

- Represent the carrier to the shelf manager
- IPMB communication
- Basic hot swap and FRU state management
- Carrier Inventory management
- Power budgeting and management
- E-keying
- Carrier event log

Shelf Manager

The Shelf Manager does what its name says: it manages the shelf. If there is just one backplane in a chassis, the shelf, in terms of management, consists of one carrier and the cooling units. However, µTCA supports chassis which utilize more than one backplane (carriers) and one or more cooling units shared or not between the carriers. In these multi-carrier environments, the shelf management is quite challenging.

The shelf manager handles the following tasks:

- Represents the shelf to the system manager over a RMCP(+) connection
- Shelf inventory and
- Inventory management
- System event log and platform event filtering
- Thermal management

Shelf managers usually also support additional, proprietary interfaces to the system manager such as CLI (command line interface) or SNMP.
Most implementations of the shelf manager preserve as much similarity with ATCA as possible in the way they present the system to the system manager. The shelf manager basically is a logical component. It either resides on the MCH as a piece of firmware, on a processor AMC within the µTCA chassis or it can be implemented outside the µTCA shelf. If the shelf manager is not integrated on the MCH, it uses an IP connection and RMCP protocol to communicate with the Carrier Manager. If both Carrier Manager and System Manager reside on the MCH, usually a proprietary, implementation-specific interface is used between Shelf and Carrier Manager.
As described, the Carrier and Shelf Manager are quite straight forward and there are several different ways to implement them. This leads to the key question: What is a system manager and where does it reside? Depending on the application and target market, the answers from system engineers will range from "Why do I need a system manager?" at the low-end to "We use OpenHPI with commercial high availability middleware in a 1+1 redundant configuration which is running our proprietary system manger application GUI on top as a distributed application" at the high-end.
Let's take a look at both of these extremes:

Operator = System manager

In applications where there is a low requirement of redundancy and automatic failover mechanisms, the basic features to be deployed are (1) hot swap management in case an operator needs to replace a faulty FRU (managed by the Carrier Manager) and (2) overall shelf management in terms of power and cooling (which are managed by the Shelf and Carrier Manager).

In this scenario, the system manager is usually the operator who services the shelf and who may be using a simple terminal program to talk to the command line interface of the Shelf Manager. Even this operator may not be around all the time and the system will work autonomously. Just in case there is a failure, the Shelf Manager will send an Alarm message so that a service engineer can attend the system. During this period

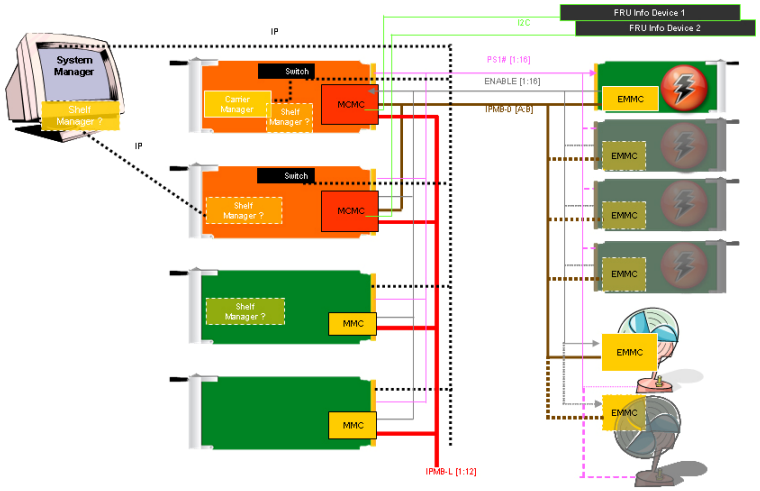

Figure 1: Shelf Manager Options

the system may have limited availability and some services may be degraded. This is acceptable in many situations where availability is less important than system cost and redundancy.

Redundancy and Availability

If there are more stringent requirements on the reliability and availability of a system, redundancy comes into play. In the telecom world, redundancy at the core of the network is implemented at component level (i.e. redundant building blocks including blades, power modules, etc.) However, redundancy is directly proportional to cost so the further one moves from the core of the network to the edge, the higher the price pressure grows on the equipment provider. At the core one finds highly available, complex ATCA systems whilst at the edge, µTCA comes into play with lower system cost and finer modularity. As a comparison, one of the key elements in µTCA is the MCH which effectively integrates an ATCA Shelf Management Module and an ATCA Hub board into a single component to drive down cost points. Even further out on the subscriber side of the network, downtime is still viewed as less critical to users and so there is generally no redundancy at all.

Outside the traditional telecom space there are different redundancy concepts. On the IT and enterprise side of the converging network it is more common to find shelves without redundant building blocks. In such environments, redundancy and availability is implemented at the system level. If a system fails, it fails completely and system management will provide a failover mechanism to one or more systems of the same type using some sort of load balancing algorithm. The simplest example is a server farm or server cluster.

SAF based management

As the networks are converging and the clear separation of an IT data center and a telecom central office is fading away, it has become evident that there needs to be a common way to manage different hardware platforms such as proprietary server architectures like IBM's Blade Center and open standards based platforms such as ATCA or µTCA with one common interface. This is where the Service Availability Forum (SAF) comes into play.

SAF defines standard interfaces for system management:

The Hardware Platform Interface (HPI) is a standard that abstracts the properties of the underlying platform and hardware and presents a standard interface to high availability middleware. Using HPI, middleware can talk to an IBM Blade Centre the same way it talks to ATCA. On the layer between middleware and applications, SAF provides the Application Interface Specification (AIS) to provide a common interface between the HA middleware and application software. In between AIS and HPI there is room for different implementations of middleware. The different applications per se do neither affect the underlying platform design nor the application software given the standard APIs.

The most popular HA middleware vendors such as Enea, GoAhead and OpenClovis all support HPI and AIS.

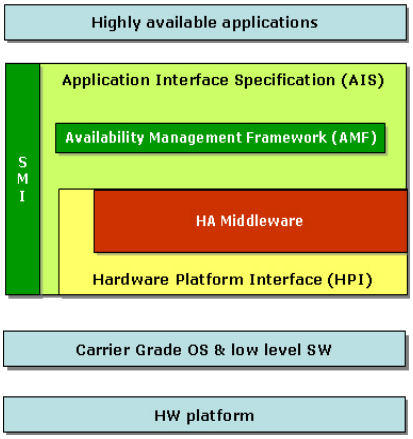

Figure 2: SAF Management standards

HPI basics

The choice of middleware is of course a matter of both technical and commercial rationale which are out of the scope of this article, but HPI is a common layer to be understood by system engineers. Let's take a look at the basics (re fig 3) HPI uses *domains* as representation of the systems to be managed and allows users to establish *sessions* with these domains. Sessions comprise user authentication and access rights. HPI automatically detects the *resources* present in the system (*domain*) by maintaining a *resource presence table* (*RPT*).
As FRUs are inserted or extracted from the system (hot plug), HPI will automatically add and remove the *resources* to and from the *RPT*. *Resources* actually represent the managed elements, called *entities*, and the *management instruments* associated with these *entities*.
In other words, *resources* are a logical representation of a piece of managed hardware and the management capabilities and methods for that hardware. For each *resource*, HPI maintains a repository of *Resource Data Records* (*RDRs*). Each *RDR* in the repository describes one *management instrument* associated with that *resource*, these include:

- Sensors (such as Hot Swap, voltage, temperature and other sensors)
- Controls (for managing LEDs, fan speeds, etc.)
- Watchdog Timers (for supervision tasks)
- Inventory Data Repositories (FRU EEPROM data as presented over IPMI containing serial numbers, production date and other information)
- Firmware Upgrade Management Instruments (e.g. for BIOS Upgrade)
- Diagnostic Initiator Management Instrument (e.g. for self tests run in foreground or background)

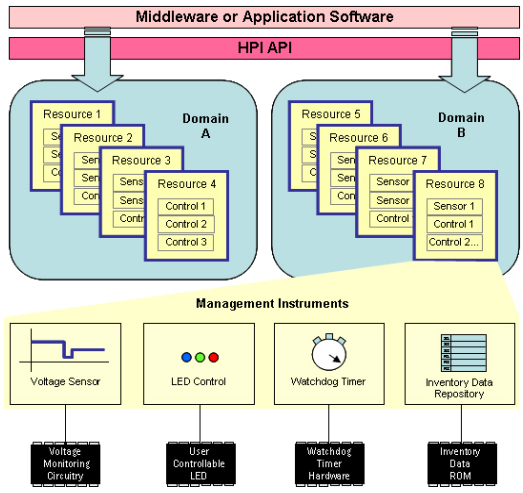

Figure 3: HPI model

HPI also manages the system event log and system alarms as well as supporting multiple domains in multiple configurations, however details are omitted here for the sake of simplicity. For further information on HPI and other SAF standards, please refer to the SAF home page at www.saforum.org.

While there are commercial implementations of HPI available, OpenHPI, an open source implementation of HPI, seems to be the most common in use. Like most HPI implementations, OpenHPI consists of two components: the OpenHPI library and the OpenHPI daemon (re fig 4). These components execute in different locations, usually (means they can also reside on the same machine), and communicate over a socket based interface on an IP link (Ethernet connection). The OpenHPI library is linked with the management application, usually a commercial high availability middleware package and performs the high level tasks. The OpenHPI daemon executes on a remote machine and is the real work horse. It uses the concept of PlugIns to talk to the underlying platform hardware. IPMI and IPMIDirect PlugIns are the most common ones to support an IPMI management infrastructure and both support xTCA system architectures. The difference between the two PlugIns is mainly that the IPMI PlugIn uses an underlying OpenIPMI library to talk to the IPMI infrastructure whereas IPMIDirect talks directly to the IPMI infrastructure.

Middleware

Middleware is the binding element between HPI and highly available applications as well as the system manager application. Middleware itself is a quite complex topic and it would require an entire article to explain this layer. However, here's a short list of the main tasks performed by HA middleware:

- high availability management
- resource and redundancy management
- alarm and fault management
- distributed messaging services
- remote procedure calls
- check pointing and supervision
- component lifecycle management

Middleware also executes in a distributed environment and is failsafe and highly available of course.
One important item to understand is that in 1 +1 redundant configurations, where one node is considered "active" and the other one "hot standby", two HPI daemons will be running on the two redundant nodes. As HPI does not require the daemons to synchronize with each other, it is up to the middleware to keep the two HPI daemon instances in sync and manage the failover between them.

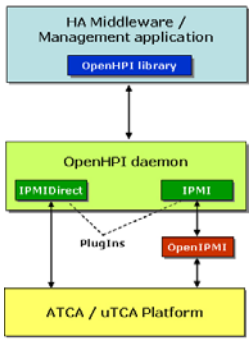

Figure 4: OpenHPI components

How the pieces fit together: Middle-where ?
After this short but deep dive into redundancy, availability, HPI and middleware let's discuss how the components could fit together in an µTCA architecture using the common 1+1 redundancy model.

Case 1
Redundant MCHs – HPI & Middleware outside the uTCAµTCA Shelf (fig. 5)

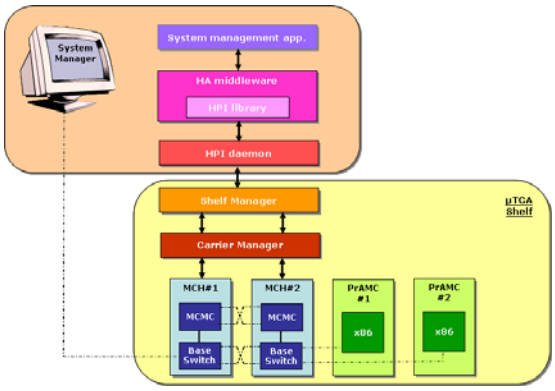

Figure 5

In this configuration, there are two redundant MCHs in the system executing the Carrier and Shelf manager functionality with integrated failover mechanisms.

The MCHs connect to an external system manager through an RMCP communication link. The external system manager implements the HPI and HA middleware and the system manager application.

Case 2
Same as Case 1 + HPI daemons on processor AMCs inside the shelf (fig. 6)

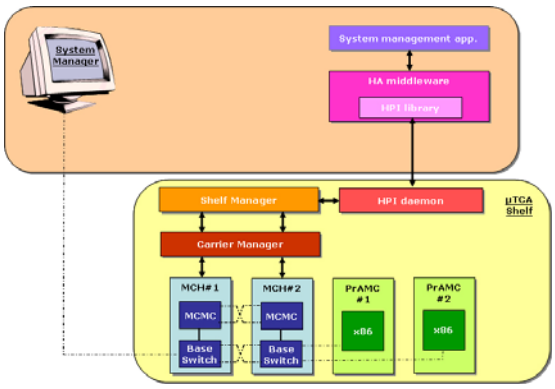

Figure 6

In this case there are two redundant processor AMCs inside the shelf that are linked to the MCHs through the base fabric interface.

The processor AMCs are running the HPI daemons and communicate with an external system manager that is executing the HA middleware and the system management application.

Note, that usually the processor AMCs are used to implement more control plane functionality in the system in parallel and are not just required for basic HA management.

Case 3
Same as Case 2 + middleware on the processor AMCs (fig.7)

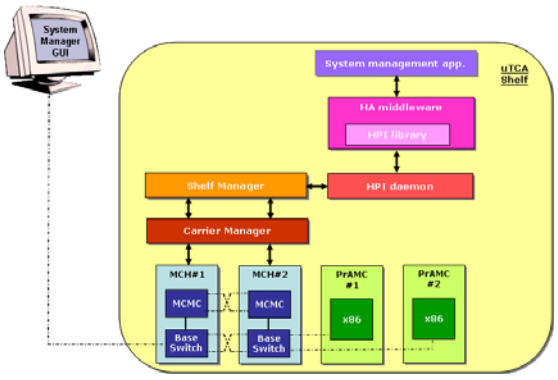

Figure 7

In this scenario, the processor AMCs are also used to run the HA middleware and system management application.

More options:
Obviously further options exist, especially if an MCH can integrate the functionality of a processor AMC on PCB3. In this way, the MCHs can supervise the complete HA and system management stack from the Carrier Manager up to the HA middleware. This can result in a highly integrated and compact system solution at a very attractive price point

Summary
This article has discussed the basic system management framework for µTCA spanning a wide range of applications in terms of availability levels and supported system management. The µTCA MCH is a key element in this architecture: At the low-end, the MCH will be the element which executes both Carrier and Shelf Manager firmware as a basis for simple but efficient system management. Component cost is a key factor for successful deployment of µTCA in such applications. At the high-end, an MCH which can integrate a control plane processor will hit the sweet spot for a highly integrated, cost efficient, highly available system using a state-of-the-art SAF-based management framework.

Figure 8: Example MCH - Advantech UTCA-5503

Advantech's µTCA-5503 MCH (figure 8) has been designed with these requirements in mind. It's modularity was architected for application-specific cost optimization across a wide range of low to high-end system topologies.