# Hardware Platform Interface Data Type Definitions

Juhana Mattila

Helsinki 10th April 2003

Seminar on High Availability and Timeliness in Linux

University of Helsinki
Department of Computer Science

# Contents

# 1 Introduction

The Service Availability™ Forum (SAF) Hardware Platform Interface (HPI) provides an industry standard interface to monitor and control highly available telecommunications system platforms. HPI provides a common model, presenting an "abstract view" of the hardware platform and its management capabilities. The ability to monitor and control these platforms is provided through a consistent and standard set of programmatic interfaces that are targeted for adoption by the telecom building block industry to significantly reduce product time-to-market and development costs while retaining or enhancing total system/network availability.

The HPI provides the interface between the middleware software solution stack and the hardware solution stack, allowing portability of middleware software building blocks across many different middleware software building blocks. The HPI is shown in Figure 1 as Platform Interface between the middleware and operating system.

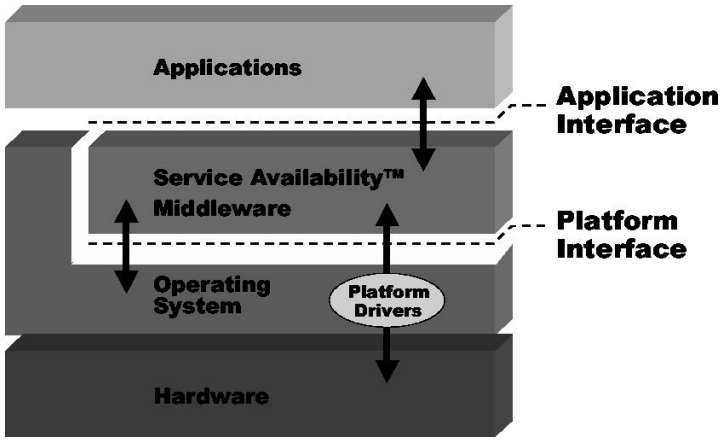


Figure 1: The Service Availability™ Forum Interfaces

The primary user of the HPI is the Service Availability™ middleware software itself. The "user" means any software component that is making use of the HPI. Any software package that needs to monitor and control the

hardware platform can use the HPI.

# 2 Hardware Platform Interface Overview

The HPI provides a platform-independent interface to platform-specific management services. It does this by representing the platform-specific characteristics of the system in an abstract model called "platform management view". The components of the system are identified and described to the extent that they may be monitored, controlled, or detected by the platform management infrastructure.The vendor-supplied implementation of the HPI is responsible for representing the physical hardware in terms of this abstract model, and for translating the library function calls that address the model into appropriate actions on the physical hardware.

The model provides two different representations of the system – a "physical view" and a "management view". The reason for the two views is that the system is likely to consist of multiple components, which may be hot-swappable. As a result, the management capabilities of the system may change over time. The interface is also designed to allow different access to management capabilities by different users, if desired. To achieve this, different users may have different management views of the system, while seeing the same physical view.

## 2.1 Physical View

The physical view of the system is achieved by identifying each component in the system with an **Entity Identifier**. An entity identifier consists of combination of an "Entity Type" and an "Entity Instance". The entity type describes the type of hardware component, and the entity instance identifies which component, in case there is more than one hardware component of the same type within the containing entity. For example a particular power

supply in the system may be identified as "Power supply number 2" or a particular subrack as "Subrack 4".

A single level description of the physical component is unlikely to be enough because the high-availability computer systems tend to be complex. Instead the HPI model uses a hierarchial model, where subsystems can be nested inside each other, with redundancy at various levels. Every entity is uniquelly named by an **entity path** that identifies the component in terms of its containment within the system. An entity path consists of an ordered set of {Entity Type, Entity Instance} pairs. The path defines the physical location of the entity in the system. The path is ordered from the entity itself, to the "root" of the system hierarchy. Each subsequent {Entity Type, Entity Instance} pair in the list describes the next higher level of physical containment.

For example, consider a system that contains multiple racks, each of which contain multiple subracks, with each subrack containing power supplies that serve that subrack. Figure 2 shows an example of system platform with example entity paths for a few components. A full Entity Path for an individual power supply may be:

| **Entity type** | **Entity Instance** |
|---|---|
| Power Supply | 2 |
| Subrack | 3 |
| Rack | 1 |

Figure 3 shows the Entity Containment Tree from the example system shown in Figure 2 with additional entities not shown in Figure 2. Note that there may also be more than one Entity Type at the top level of the system.

## 2.2 Management View

HPI presents a "platform management view" of the components in the system. The physical entities in the system have basic management capabilities

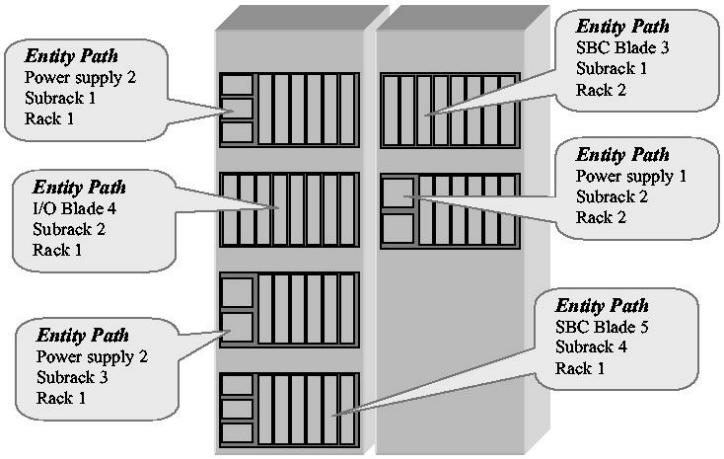


Figure 2: HPI Physical View

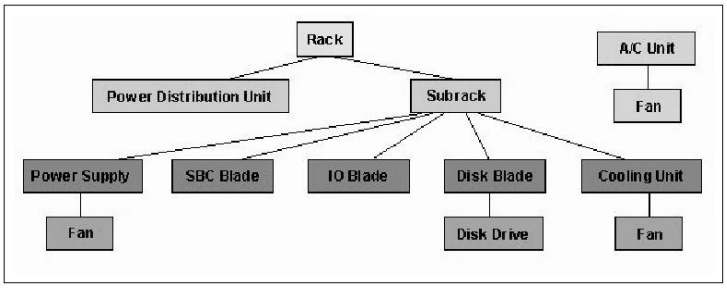


Figure 3: Enity Containment Tree

that are modeled in the HPI as "management instruments", such as sensors, controls or watchdog timers. Management instruments throughout the system are addressed using a hierarchial address consisting of a "Domain ID", "Resource ID" and "Instrument ID".

A **Resource** is simply a collection of management instruments associated with one or more entities in the system. Each resource is responsible for managing and presenting to the HPI user the entities that it has management control over.

The HPI view of a system is divided into one or more **domains**, where a

domain provides access to some set of the resources within the system. A resource is a member of one or more domains. Access to all of the management instruments contained in a resource is permited to all users who are able to access a domain that contains the resource. Resources may be members of more than one domain at the same time, permiting different sets of users simultaneous access. Figure 4 shows an example of a system with two domains. Many systems may have a single domain, whereas systems that have areas dedicated to separate tasks may manage these through separate domains.

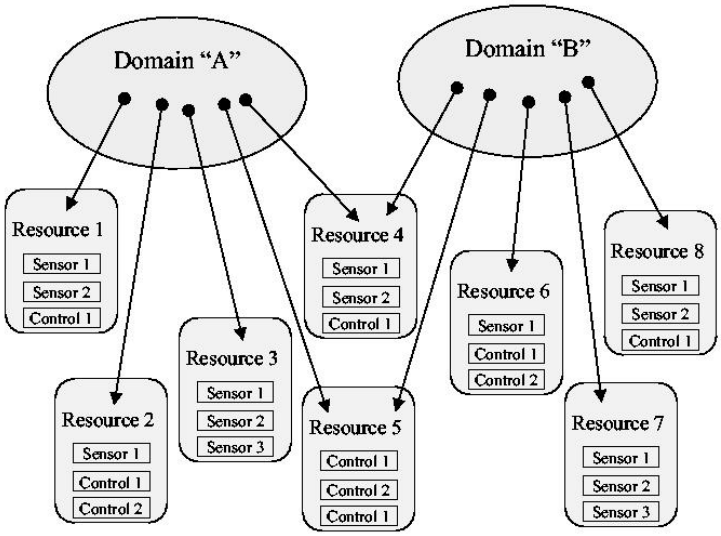


Figure 4: HPI Management View Model

A user of the HPI accesses the system through **sessions**, where each session is opened on a domain. A session provides access only to resources that are visible in the domain upon which the session is opened. One HPI user may have multiple sessions open at once, and there may be multiple sessions open on any given domain at once. It is inteded that, in future releases of HPI, access control to the HPI will be performed at the session level.

HPI uses the management view to represent the components in the system. The entity paths from the physical view are used to identify the hardware

components that management instruments and resources are associated with. Each management instrument is associated with entity path, which identifies the specific hardware component in the system the management instrument monitors or acts upon. Each resource is associated with an entity path, which identifies the "primary" hardware component associated with a hot-swappable hardware component.

## 2.3 Management Instruments

Management capabilities of system entities are modeled in the HPI with "management instruments". Management instruments are contained in HPI resources, and are associated with specific physical entities. Four types of management instruments are defined.

**Entity Inventory Records** reports inventory and static configuration data of the entity. The inventory data can include for example manufacturer, model number, revision level, serial number.

**Sensors** are used to read values related to the operation or health of a component. Generally, sensors are used to model values that may change over time, like temperatures, voltages, latch positions, etc. When changes in the reading occur, the sensor can assert or de-assert one or more of its event states. When a sensor asserts or de-asserts an event state, an HPI event message may be created and sent to an HPI user or put in a log.

**Controls** are the controlling aspects of the operation of a component. Controls are used to send commands to an entity.

**Watchdog Timers** are used to physical watchdog timers that may be implemented on physical entities. Watchdog timers may cause implementation-defined actions to occur when the timers expire.

# 3 Using the Hardware Platform Interface

The HPI is made available in a series of C language library calls and a header file provided by a platform vendor. The header file is taken directly from the SAF specification, and except for assigning a few basic data types to the appropriate types for the processor family, no changes are required by the vendor. The library functions, however, must be written specifically for each system platform to map the HPI model and functionality to the actual hardware platform capabilities. The platform vendor must provide libraries appropriate for whatever operating system and compiler used on the platform.

The Service Availability$^{TM}$ middleware, or other user software, invokes the library by including the header file in its software modules, and making calls to the appropriate library functions.

## 3.1 Initialization and Cleanup

There are two HPI function calls that should be used just once each for system initialization and shutdown, respectively. These calls, which take no parameters, are designed to allow the HPI implementation to schedule any startup or close-down activities that may be needed on a global basis.

Before the user can use the HPI user must call the function `saHpiInitialize()` to initialize the the HPI. The function returns the version of the HPI implementation. Initialization is run only once for each HPI instance. Though many implementations may simply return from initialization function without any further processing other implementations may use this as an opportunity to allocate memory, to open files, to set up default configurations, etc.

After the user stops using the HPI user must call the function `saHpiFinalize()` to clean-up the HPI. The function is provided to allow for freeing of memory,

closing of files, etc., as needed by a particular implementation.

## 3.2 Sessions

A user of the HPI accesses the system through sessions, where each session is opened on a domain. A user initiates interaction with the HPI by calling function `saHpiSessionOpen()`. The caller passes a domain identifier `DomainId` to the open function, and a session identifier `SessionId` is returned. There is also a parameter for security characteristics `SecurityParams` but it is reserved for future versions of HPI. At this point this parameter must be set to `NULL`.

Once a session is opened on a domain all subsequent function calls that reference that session (via session identifier), implicitly address a specific domain. All subsequent function calls have a session identifier `SessionId` as a parameter. A user may have multiple session open at once. A user is able to access resources and retrieve events via a session.

The function `saHpiSessionClose()` closes a session. A session identifier `SessionId` is passed to the function and the identifier will no longer be valid.

## 3.3 Domains

An HPI system is organized into one or more domains. Regardeless of how domains are implemented by specific HPI service, to HPI user, all domains have a common structure. A domain consists of a domain controller, and it may contain zero or more resources. Figure 5 shows a typical domain. A domain controller is a abstraction of set of services that provide information about the resources in the domain. These services include management of a Resource Precense Table (RPT), and management of Events.

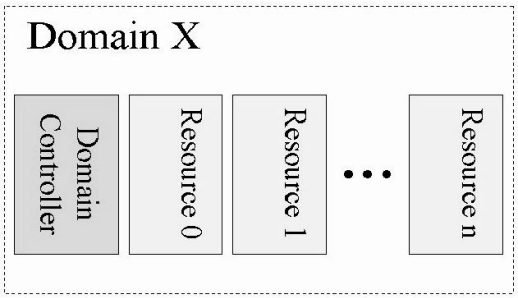


Figure 5: Example Domain

The RPT contains an entry for each resource currently present in the domain, and users of the HPI may read these entries. The resources can be used to discover which entities are present. The contents of RPT entry describing a resource is covered in more detail in section 3.4 Resources. The RPT is automatically built and maintained by the HPI implementation. Resource entries will be dynamically added to or removed from the RPT as Field Replaceable Units (FRUs) are physically added to or removed from a platfrom. If a resource is contained in multiple domains, it will be recorded in the RPT of each of the respective domain controllers.

The event management service in domain controller, forwards events to users of the HPI and maintains a historical log of events. The event management service collects events in a domain controller and then distributes those events to the System Event Log (SEL) and to users who have subscribed for events. If the HPI implementation presents multiple domains, the domain controller in each domain will manage its own set of events. The event management service is covered in more detail in section 3.7 Events.

The first important aspect to consider in domain architecture is the default domain (`DomainID = SAHPI_DEFAULT_DOMAIN_ID`). The default domain allows the user of the HPI to open a domain and interact with the HPI implementation, even when it has no prior knowledge of the platform. The use of the default domain is coverd in more detail in section 3.6 Discovery.

## 3.4 Resources

Resources represent the management access to the components of the system. Each resource provides access to information about some of the components of the system called entities. Each resource currently accessible in a domain must be represented by an entry in the RPT in the domain controller of that domain. Each RPT entry includes:

- RPT entry ID.
- Resource ID of the resource with which the RTP entry is associated.
- Resource info that contains static configuration data concerning the management controller associated with the resource, or the resource itself.
- Entity path of the primary hardware component with which the resource is associated.
- Cababilities of the resource. This identifies if the resource is a domain or a resource (or both). Capabilities also include information of what types of management instruments are associated with this resource.
- Domain ID if the RPT entry is associated with a domain.
- Informational value that supplies the caller with naming information for the resource.

## 3.5 Entities

There is no single element of the HPI that represents the entity. Rather, each entity is modeled as a collection of Resource Data Records (RDRs), which contain information about the controls, sensors, watchdog timers, and inventory data associated with the entity. Every RDR contains the entity path of the entity to which it relates; hence, the user can determine the type

of a given entity, and the set of the RDRs in the system, and correlating RDRs with the same entity path. Each RDR includes:

- Management instrument ID used to address the management instrument in the resource.
- Type of management instrument – sensor, control, entity inventory repository, watchdog.
- Entity path for the physical hardware component with which the management instrument is associated.
- In the case of sensors and controls, descriptive information about the particular management instrument, including information on what it actually monitor or controls.

Every entity in the system must have at least one RDR associated with it. If the entity contains no controls, sensors, or watchdog timers, then the HPI implementation should supply an empty entity inventory RDR for that entity.

## 3.6 Discovery

After establishing a session to a domain, user software can use function calls provided in the HPI to discover all the resources and management capabilities of the system platform. Since domains are the largest aggregation of resources that can be addressed, the first task in discovering the content of a system is to determine the domains that are present in the system. Every HPI implementation must have one domain with a `DomainId` equal to `SAHPI_DEFAULT_DOMAIN_ID`. This domain allows the HPI user to open an initial session to the HPI, and serves as the starting point for a full discovery procedure.

In order to support this full system discovery, the specification requires that the HPI implementation include domain reference entries in RPTs so that if a user accesses a default domain, and then accesses all the domains referenced in the RPT of that domain, and then continues this process for all those domains, and so on, then all domains available in the system will be discovered.

To discover the resources in the domain, the user reads the RPT entries of the domain by calling function `saHpiRptEntryGet()`. The caller passes a session identification `SessionId` and an RPT entry identification `EntryId` to the function. The function returns the RPT entry `RptEntry` for the specified entry and a entry identification `NextEntryId` of the next entry in RPT. To retrieve an entire list of entries, function `saHpiRptEntryGet()` is first called with the RPT entry identification `EntryId` set to `SAHPI_FIRST_ENTRY`. Then the function returns the first entry in RPT and the identification of the next entry. Then caller uses the returned `NextEntryId` in the next call. This is repeated until the `NextEntryId` returned is `SAHPI_LAST_ENTRY`.

For each RPT entry the user extracts the capability flags `ResourceCapabilities` for that resource to find out if the RPT entry identifies a resource or a domain. The user also extracts the resource ID or domain ID, which will be used in the following function calls.

After discovering each resource, the user should then access the RDRs maintained by that resource to find out the entities managed by the resource. The user reads the RDRs from the RDR repository by calling function `saHpiRdrGet()`. The caller passes a session identification `SessionId`, an resource identification `ResourceId` and an RDR identification `EntryId` to the function. The function returns the RDR `Rdr` for the specified entry and a entry identification `NextEntryId` of the next entry in RDR repository. To retrieve an entire list of entries, the function `saHpiRdrGet()` works the same way as the function `saHpiRptEntryGet()`. The user can use the entity path `Entity` in the RDR to determine which RDRs refer to same entities.

If a RPT entry is also identifying a domain with unique domain ID, user

should repeat the same process for this subdomain. This will allow all domains to be discovered.

After the process described above user has information of all domains, resources and entities in the system. The physical view of the system can be constructed during this process by using the entity paths contained in each RDR.

## 3.7 Events

After discovery is complete, user software may operate in polled mode, an event driven mode, or a combination of the two. The event driven mode is described below.

To get events on current session, the user must first subscribe to receive events by calling function `saHpiSubscribe()`. The caller passes a session identifier `SessionId` and a boolean value `ProvideActiveAlarms` to the function. (The parameter `ProvideActiveAlarms` is described below.) When a user subscribes to events from a particular domain, the HPI sets up an event queue for the user, and places a copy of all events related to any resource that is a member of that domain on the queue as they are generated.

After subscribing, a user may read events off the queue by calling function `saHpiEventGet()`. The caller passes a session identifier `SessionId` and a timeout parameter `Timeout` to the function. The function returns next available event `Event`, the resource data `Rdr` associated with the event and the RPT entry `RptEntry` associated with the resource that generated the event. The returned event includes the identification of the source which generated the event, the type of the event, a timestamp when the event was generated and information about the severity of the event.

If the timeout parameter `Timeout` is set to `SAHPI_TIMEOUT_IMMEDIATE` the events are read with a non-blocking call which returns immediatly if there are no events in the event queue. If the timeout parameter is set to `SAHPI_TIMEOUT_BLOCK`

the events are read with a blocking call which will block until there are new events in the event queue.

A user can operate simply by receiving and responding to events, without having to poll the various management instruments in the system on a regular basis. However, to operate this way, the user must first be aware of the "initial state" of the system. To assist with this, the HPI allows the user to learn this initial state via processing event messages rather than having to poll all the sensors in the domain. When the user calls the function `saHpiSubscribe()` with the parameter `ProvideActiveAlarms` set to `SAHPI_TRUE`, the HPI places events on the user's event queue for all active alarms in the system.

In addition to forwarding events to subscribing users, the HPI also logs events in a nonvolatile "event log" associated with each domain. This event log is accessible by users through the HPI at any time, to retrieve a historical record of events that have been generated for a domain. Users may also add records to the event log, if desired.

Figure 6 shows how events are processed by the HPI for a particular domain.

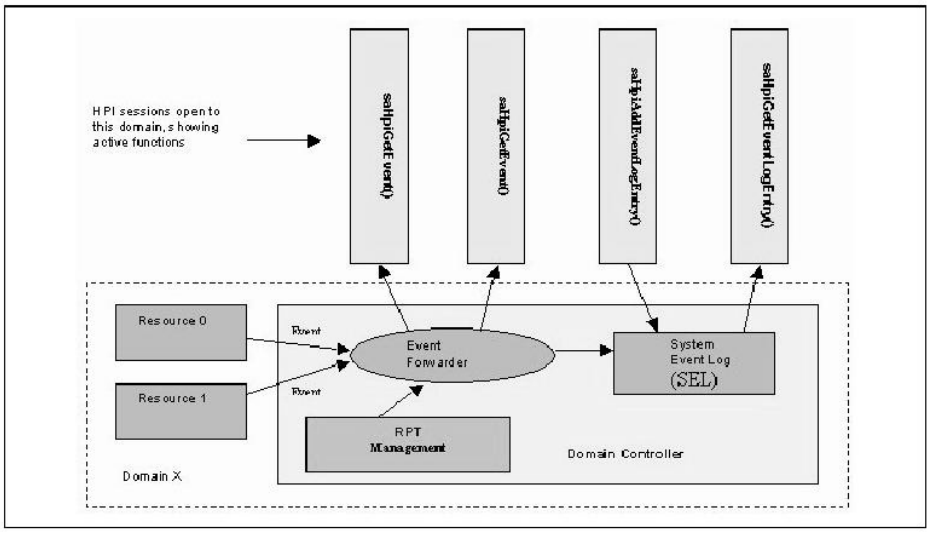


Figure 6: Domain event processing

# 4 Conclusion

The HPI provides a standard interface for the rich management capabilities found in the carrier-grade hardware platforms used with applications that msut provide continuous availability. The interface is built on a model that provides an abstract view of the system, while still allowing full access to whatever management capabilities it may contain. A "discover what is there" approach is used so that Service Availability$^{TM}$ middleware or the carrier-grade software can be developed that is adaptable to the underlying system capabilities, thus avoiding a "least common denominator" view of the system.

# References

[SAF02a] The Service Availability$^{TM}$ Forum, Platform Interface. White paper, 2002. Available at: http://www.saforum.org/specification/specification_white_paper.pdf

[SAF02b] Service Availability$^{TM}$ Forum, Hardware Platform Interface Specification. 2002. Available at: http://www.saforum.org/