The Service Availability™ Forum Platform Interface

The Service Availability™ Forum develops standards to enable the delivery of continuously available carrier-grade systems with off-the-shelf hardware platforms and middleware.

This paper explains the context and need for widespread adoption of one of these standards – the Service Availability™ Forum Platform Interface. It introduces the interface and provides guidance in its use.
Contents

Introduction ........................................................................................................................................... 3
Need for a Standard Carrier-grade Platform Interface ................................................................. 4
The Role of the Platform Interface in a Service Availability™ Solution ........................................ 5
The Service Availability™ Hardware Platform Interface Model ....................................................... 6
   Physical View ................................................................................................................................. 6
   Management View ......................................................................................................................... 7
Management Instruments ..................................................................................................................... 9
   Entity Inventory Repositories ........................................................................................................ 9
   Sensors ............................................................................................................................................ 9
   Controls ......................................................................................................................................... 10
   Watchdog Timers ............................................................................................................................ 10
Event Generation and Logging ........................................................................................................... 10
Discovery ........................................................................................................................................... 11
Hot Swap Capabilities ......................................................................................................................... 12
Using the Service Availability™ Forum Platform Interface .............................................................. 14
Conclusion .......................................................................................................................................... 16

Figures

Figure 1 : The Service Availability™ Forum Interfaces ................................................................. 3
Figure 2 : HPI Physical View ............................................................................................................ 7
Figure 3 : HPI Management View Model ......................................................................................... 8
Figure 4 : Association of management instruments with physical entities .................................. 9
Figure 5 : Domain event processing ............................................................................................... 11
Figure 6 : Simplified hot swap model ............................................................................................. 13
Figure 7 : Full hot swap model .......................................................................................................... 14
Introduction

The Service Availability™ Forum is a consortium of communications industry leaders and innovative startups dedicated to producing standards aimed at enabling the development of carrier-grade communications systems from off-the-shelf hardware platforms and middleware. The key characteristic of a “carrier-grade” system is its ability to provide uninterrupted user access to the services the system is designed to deliver, with no loss to the continuity of those services.

To meet this goal, the Service Availability™ Forum is developing two layers of standard carrier-grade interfaces: an Application Interface, and a Platform Interface, as shown in Figure 1.

The Service Availability™ Forum Application Interface provides access to a standard set of tools for application software to use in order to distribute its processing over multiple computing elements, and to respond to failures of those elements without loss of service delivery or continuity to any user. These tools are provided through management middleware that conforms to the Service Availability™ Forum Application Interface specification. Middleware which conforms to this specification is called “Service Availability™ middleware.”

Service Availability™ middleware provides these tools to application software in part by monitoring and controlling the physical components of a high-availability computing platform via the Service Availability™ Forum Platform Interface. By using a standard interface to manage the physical platform, Service Availability™ middleware can be written that is independent of any particular hardware. This, in turn, allows application developers to choose the best hardware platform and the best Service Availability™ middleware to fit their needs.

This white paper discusses the need for a standard carrier-grade platform management interface, and identifies benefits derived from the widespread adoption of the Service Availability™ Forum Platform Interface. It then provides an overview of the interface, and guidance on using it to monitor and control a high-availability system platform as part of a complete carrier-grade solution.
Need for a Standard Carrier-grade Platform Interface

While there is much diversity in the architecture of system platforms that can be used to build a carrier-grade solution, a common characteristic of any suitable platform is manageable redundancy. In order to provide continuous service, application software must be able to execute continuously. Even with the most reliable hardware available today, this requires a strategy that permits continued operation in the presence of hardware failures. To meet this requirement, system platforms must provide redundant components, and methods for allocating application processing to those components.

Furthermore, a carrier-grade solution must include a strategy that permits continued operation in the presence of software failures. Typically, this is also achieved with the aid of redundant hardware. When software running on one computing element fails, processing can continue on other computing elements that are either already operating in parallel, or at least are in a “hot standby” mode.

Finally, the ability to repair the hardware platform while it continues to operate is required in systems that are to provide continuous service. This implies a need to have “hot-swappable” components in the system. Often, system platforms include the capability to actually change the hardware configuration while the system continues to run, in addition to just replacing failed components.

Because of these needs, system platforms used for carrier-grade solutions tend to be complex, with significant platform management capabilities. Typical capabilities include:

- Monitoring the “environmental health” of system components, including measurement of ambient temperatures, component temperatures, and input voltage levels.
- Monitoring the “performance health” of system components such as fan speeds, power supply output voltages and currents, or memory ECC errors.
- Setting various configuration or operational characteristics of hardware components in the system.
- Reporting “inventory data” including component model numbers, serial numbers, revision levels, etc., especially for field replaceable components.
- Detecting and managing the hot-swap actions of components in the system, including controlling power to individual components.
- Operating watchdog timers, which can automatically reset or power cycle a board if the software running on that board stops.

In order to correctly control processing of the continuously available application, carrier-grade middleware must understand how to interact with the hardware platform, and utilize its “built-in” platform management capabilities. While various approaches to standardization of management capabilities have been developed, no existing platform management standards provide a strong, common model of a high availability system with redundant, hot-swappable components.

The Service Availability™ Forum Platform Interface provides such a common model, presenting an “abstract view” of the hardware platform and its management capabilities. By
using this common model and abstract view to monitor and control the actual hardware platform, Service Availability™ middleware packages can be developed that run equally well on any platform that supports the Service Availability™ Forum Platform Interface specification. This produces several significant benefits:

- Service Availability™ middleware packages can be developed more quickly, and more economically, because there is no need for the middleware developers to devote significant resources to “porting” their product to various hardware platforms.

- Platform vendors may leverage more Service Availability™ middleware options by developing a single standard platform interface that is immediately usable by different middleware implementations.

- Platform vendors are more free to innovate, because new hardware features may be quickly made available through the standard interface to a variety of Service Availability™ middleware packages.

- Users will see a more consistent model of the hardware platform for management as it is “filtered” through the standard interface and Service Availability™ middleware.

The Role of the Platform Interface in a Service Availability™ Solution

In order to maximize the benefits of an open architecture, all access and control of the hardware platform in a Service Availability™ solution should be made exclusively through a platform interface that conforms to the Service Availability™ Forum Platform Interface specification. There is adequate flexibility in the interface specification so that a platform vendor should be able to model all management capabilities of the hardware through the interface.

The primary user of the interface is the Service Availability™ middleware software itself. In the Service Availability™ model, as shown in Figure 1 on page 3, Service Availability™ middleware has the responsibility of monitoring and controlling the hardware platform, and providing services to the application software that are independent of any particular hardware. Service Availability™ middleware does this by using the standard platform interface to discover what capabilities the hardware platform provides, then mapping these capabilities into a standard system model, which is maintained by the Service Availability™ middleware and presented to application software.

The platform interface is not available exclusively to Service Availability™ middleware, though. It is an independent interface usable by any software package that needs to monitor and control the hardware platform. By providing platform management via this standard interface, vendors make their hardware manageable by any software written to use the interface.

In particular, the adoption of the Service Availability™ Forum Platform Interface specification by platform vendors provides an important migration path from purely proprietary systems in use today to completely open Service Availability™ solutions in the future. Today, application software often includes availability management functionality embedded within itself, and the application is then adapted to a specific hardware platform.
By adapting an existing application to the virtual “Service Availability™ platform,” it then becomes usable on any hardware platform that supports the standard interface.

When Service Availability™ middleware solutions are developed that provide services the application developer wishes to use, direct platform management can be removed from application software as a middleware package assumes those responsibilities in the system.

**The Service Availability™ Hardware Platform Interface Model**

The Service Availability™ Forum Platform Interface is called the “Hardware Platform Interface” or “HPI” in the specifications. This designation is intended to emphasize the purpose of the interface, which is to provide monitoring and control of the actual platform hardware.

As stated in the specification itself, “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, then providing standard methods of monitoring or controlling that model. 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.

Because the model is designed to represent high-availability system platforms with built-in redundancy and extensive management capabilities, it includes the ability to represent relatively complex system architectures.

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 management infrastructure in a high-availability platform is likely to consist of multiple components, which themselves may be hot-swappable. As a result, the management capabilities of the system may change over time as the management infrastructure itself changes. Furthermore, the interface is 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.

**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 the combination of an “Entity Type” and an “Entity Instance.” The entity type describes the type of hardware component, and the entity instance allows multiple occurrences of a particular hardware component to be distinguished. For example, a particular power supply in the system may be identified as “Power supply number 3,” or a particular compute blade as “SBC blade number 4.” A selection of entity type codes is included in the specification.

Due to the complexity of high-availability computer systems, though, a single level description of the physical component is unlikely to be sufficient. Rather, high-availability systems tend to have subsystems nested inside of each other, with redundancy at various levels in the hierarchy. So, the HPI model uses a list of entity type/entity instance pairs called an **Entity Path** to fully identify each component. Each subsequent entity type/entity instance pair in the list describes the next higher level of physical containment.
For example, if a system consists of two racks, each of which contains up to four subracks, and each subrack may house multiple power supplies, then a particular power supply in the system may be identified with the entity path: “Power supply number 2 in subrack number 3 in rack number 2.” Figure 2 shows an example of a system platform with example entity paths for a few components.

Each “container” entity like a subrack may have an arbitrary configuration. Whatever sorts and numbers of components that are actually contained in the subrack can be described by using the appropriate entity paths. However, the HPI model includes specific system structures for CompactPCI® and AdvancedTCA™ systems. The user of the interface may inquire of the interface as to whether or not one of these pre-defined physical platform models are used, and if so, user software may then tailor its management of the platform accordingly.

Management View
The physical entities in the system have basic management capabilities that are modeled in the interface as “management instruments,” such as sensors or controls. These are discussed more fully in the next section. Management instruments throughout the system are addressed using a hierarchical 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. Management instruments are modeled as belonging to a single resource when they share common accessibility in the system. For example, if the management infrastructure in a platform contains several management controllers, each of which provides management services for different sets of boards, fans, power supplies, etc., then each of these management controllers may be modeled as a separate resource containing the sensors and controls that correspond to the actual management services provided by that controller. If one of the management controllers fails or is removed from the system, the interface can report that the corresponding resource is no longer available, and all of the specific sensors, controls, etc., hosted by that management controller are therefore not accessible.
A resource does not have to correspond to a physical management controller, however. There are other reasons the platform interface may make parts of the platform management infrastructure available or unavailable at different times, or to different users. One important reason is in support of hot-swappable components, described in more detail below. Another reason is to give certain users management control over just part of the overall system. This can be particularly useful in “multi-tenant” systems where parts of a high-availability system platform are leased by a service provider to independent customers.

A resource is a member of one or more **Domains** in the HPI model. Access to all of the management instruments contained in a resource is permitted 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, permitting different sets of users simultaneous access. For example, in a multi-tenant system, a particular resource may be available to a tenant, and also to a system administrator through different domains.

Figure 3 shows an example of the management view of a system. A user with access to Domain “A” would have visibility and access to the management instruments in resources 1, 2, 3, 4, and 5. A user with access to Domain “B” would have visibility and access to the management instruments in resources 4, 5, 6, 7, and 8. Although it is not shown in this diagram, it is possible for resources to have different Resource IDs in each domain.

![Figure 3: HPI Management View Model](image)

The physical view of the system is overlaid on the management view by associating entity paths with each management instrument, identifying the specific hardware component in the system the management instrument monitors or acts upon. Each resource is also associated with an entity path, which identifies the “primary” hardware component associated with that resource. This is of particular significance when a resource is associated with a hot-swappable hardware component.
Management Instruments

Management capabilities of system entities are modeled in the HPI with “management instruments.” Four types of management instruments are defined: sensors, controls, entity inventory repositories, and watchdog timers. Management instruments are contained in HPI resources, and are associated with specific physical entities, as shown by example in Figure 4.

![Figure 4: Association of management instruments with physical entities](image)

**Entity Inventory Repositories**

Each physical entity in the system may have inventory data associated with it, readable via the HPI interface. This inventory data, such as manufacturer, model number, revision level, serial number, and static configuration information, is accessible by reading records from a management instrument called an entity inventory repository. The HPI user may read the inventory data from any repository, and may also update the data in a repository.

**Sensors**

A sensor is used to monitor any physical characteristic of an entity. Generally, sensors are used to model values that may change over time, like temperatures, voltages, latch positions, etc. However, they can also be used to report static configuration information such as the setting of straps or switches on a component.

Sensors may report analog or discrete values. Analog values are reported in terms of a raw reading, or in the appropriate engineering units for whatever is being measured. Discrete values are reported as just a raw reading, which is often just a single bit, but may be multiple bits.
Sensors also have associated event states. 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.

For analog sensors, event states usually correspond to the reading crossing certain threshold values, which may be set by the HPI user. For discrete sensors, the HPI implementation will pre-define conditions that result in sensor event states being asserted or de-asserted. For many discrete sensors, the actual raw reading may be meaningless, and the only useful information will be the sensor event state.

**Controls**

While a sensor is used to monitor physical entities, a management instrument called a control is used to send a command to an entity. Controls are abstract management instruments that are tied to actual capabilities by the HPI implementation. Six types of controls are defined to handle different sorts of data that may need to be sent to the entity. The defined control types are:

- Digital – send on/off type settings
- Discrete – send multiple-bit settings where the meanings of different bits or patterns is implementation specific
- Analog – send an analog value
- Stream – send a repeating pattern of on/off bits (e.g., to make an LED flash, or an audible alarm beep in a fixed pattern)
- Text – send text to a display device
- OEM Defined – catchall for any sort of data a specialized control may need

**Watchdog Timers**

A watchdog timer management instrument is used to control physical watchdog timers that may be implemented on physical entities. Specialized function calls are available to configure and start a watchdog timer, to send “keep-alive” heartbeats to it, and to define what actions are taken when it expires.

**Event Generation and Logging**

As discussed above, sensors may generate event messages when state changes occur or thresholds are crossed. This is one example of a more general capability within the HPI of event generation and logging when significant things happen in the system.

Users of the HPI may “subscribe” to receive events from the HPI on a domain-by-domain basis. 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 via a blocking or non-blocking call. If a user process makes a blocking call to receive an event, that process will wait until an event is received, and will then be “woken up” to process the received event.

Because event messages are generated for all “significant occurrences” in the system platform, a user can conceivably 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 provides a special feature that 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 subscribes to receive events from a domain, a flag may be set that requests the HPI place events on the user’s event queue for all “active alarms” in the system. This effectively recreates events that occurred prior to the subscription request, but that the user “needs to know about” if it is not going to poll all sensors for their current state.

In addition to forwarding events to subscribing users, the HPI also logs events in a non-volatile “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 5 shows how events are processed by the HPI for a particular domain.

![Figure 5: Domain event processing](image)

**Discovery**

The Service Availability™ Forum Platform Interface is designed to be used with a wide variety of system platforms. Rather than require platforms to offer pre-defined management capabilities, or to provide an interface that meets a “lowest common denominator” level of service, the interface provides a method for the platform to describe to the user what platform management capabilities it offers. As a by-product of this feature, the user can also discover the general makeup of the system.

Discovery of the system platform and its management capabilities is accomplished by accessing two data structures maintained by the interface: The Resource Presence Table
The Service Availability™ Forum Platform Interface

(RPT) associated with a domain, and the Resource Data Records (RDRs) associated with a resource.

There is an RPT maintained for each domain defined in the interface. Within the RPT, there is an entry for each resource that is a member of that domain. By reading the RPT in a particular domain, the user may gain visibility to the part of the overall system that is manageable through that domain. If a domain is set up to permit a user access to only part of the system, that user will only “see” the part of the system that it may access.

An RPT may also contain entries that reference additional domains available through the interface. These entries are used so that a user may discover all the resources available in the platform, even if they are not all included in a single domain. 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 special “default domain” defined in the specification, 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.

By reading RPT entries, a user will learn of all the resources accessible through a domain. After discovering each resource, the user should then access the Resource Data Records (RDRs) maintained by that resource. These records describe each of the management instruments available in the resource. There are differently formatted RDRs for sensors, controls, entity inventory repositories and watchdog timers. Each RDR includes:

- Type of management instrument – sensor, control, entity inventory repository, watchdog
- Management instrument ID used to address the management instrument in the resource
- 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 monitors or controls.

While reading RDRs, the user software will encounter entity paths for each management instrument. Often, Service Availability™ middleware or other carrier-grade software will need to create an internal model of the entire system; that is, the “physical view” of the system. This may be accomplished by doing a complete discovery of domains, resources, and management instruments, noting the entity paths of each management instrument as it is discovered. By collecting these entity paths, discarding duplicates, the physical view of the system may be constructed. If a platform includes physical entities that contain no management capabilities, but nonetheless need to be discovered and mapped by middleware, the HPI implementation may create empty, “dummy” entity inventory repositories for these entities and place them resources.

Hot Swap Capabilities

One of the key things that set the platforms used for carrier-grade solutions apart is their inclusion of “hot swappable” components. At a minimum, platforms generally have the ability to remove and replace failed components while the rest of the system remains
operational. More commonly, they include a general capability to have components inserted or removed while the system runs – for reconfiguration or repair purposes.

Managing hot swappable devices and a changing platform configuration is an important feature of the Service Availability™ Forum Platform Interface. A component that may be added or removed from the system as it operates is called a “Field Replaceable Unit” or “FRU” in the specification. FRUs are always modeled as separate resources in the management view of the system platform. This allows the interface to dynamically adjust as the hardware platform configuration changes.

When a FRU is inserted in the system, the HPI implementation will add a corresponding resource to at least one domain, and will generate an appropriate event messages. User software can then access the RDRs in that newly added resource to discover the capabilities of the FRU. Thus, even if a new type of FRU is added to the system, Service Availability™ middleware can dynamically incorporate it into its system model.

Similarly, when a FRU is removed from the system, the corresponding resource is removed from any domains in which it had membership, and appropriate events are sent. User software thus is aware that all the management instruments hosted by that resource are no longer available in the system, and the entities they are associated with have been removed.

The HPI supports two models of the actual hot-swap activity. A simplified hot-swap model is used for FRUs that do not require any special processing as they are inserted or removed. The state diagram for this model is shown in Figure 6.

For FRUs that require special processing related to their insertion or extraction, the full hot swap model may be used, as shown in Figure 7. This model includes three additional states: “Insertion Pending,” “Extraction Pending,” and “Inactive.”

The Insertion Pending and Extraction Pending states are used to do special processing as a FRU is being inserted or extracted, respectively. The HPI can be configured to do automatic, pre-defined processing on insertions or extractions, or to simply inform the user that an insertion or extraction is taking place, allowing the user software to do any required processing. This may include integrating new hardware into the application, gracefully ending the use of hardware that is being removed, or anything else needed to support this action.

The Inactive state is used when a FRU has been logically “hot-swapped” out of a system, but it is still physically present in the platform. From this state, user software can cause a “re-insertion” action to take place by issuing a special command to the HPI. With the full hot swap model, user software can also initiate an “extraction” operation via a command to the
HPI. By using these commands, a FRU may be logically “removed” and “re-inserted” under program control, in order to attempt to clear a fault condition.

**Figure 7: Full hot swap model**

**Using the Service Availability™ Forum Platform Interface**

The platform interface 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 Service Availability™ Forum 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 is used on the platform.

Service Availability™ 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.
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.

Other than these two functions, all accesses to the HPI by users are processed in relation to a context called a “session.” A user initiates interaction with the HPI by calling a function to “open” a session. The caller passes a DomainID to the open function, and a session “handle” is returned. All subsequent calls to the HPI include the session handle in order to provide context for the call.

A particular session is said to be opened “to a domain” because all calls using that session handle can only access resources and other capabilities related to the domain that was identified in the session open call. For example,

- Function calls that read RPT entries will read the RPT entries associated with that domain.
- Function calls that subscribe to events or read event queues will only deal with events for resources that are members of that domain.
- Function calls that address resources or management instruments hosted by resources may only access resources that are members of that domain.

After establishing a session to a domain, user software can use function calls provided in the interface to discover all the resources that are members of the domain by reading the domain RPT, and may discover all the management instruments and entities visible and accessible via that domain by reading the RDRs in each discovered resource, as well as any entity inventory repositories, as they are found.

If the domain RPT contains references to other domains, user software may open additional sessions to those domains to further discover other parts of the system. During this process, it is possible to encounter the same resource in more than one domain. The user software should detect this, and adjust its discovery routine appropriately.

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

In a polled operation, the user software can periodically scan all sensors to get current readings and check for alarm conditions, set and send heartbeats to watchdog timers, and operate any controls as needed. The user should also periodically read the domain’s RPT information record to determine whether any resources have been added to or removed from the domain due to hot-swap activities. If a resource is removed, further polling of that resource should be stopped. If a resource is added, the RDRs for that new resource should be read, and the newly discovered sensors added to the list of sensors to be scanned.

In an event driven operation, user software will issue an event subscription request to the domain. The user can then issue a blocking call to get the next event on the queue. If there are no current events, the process will suspend until one is available, which will then be returned. Along with the event, the user may request a copy of the appropriate RPT entry and RDR for the resource and management instrument generating the event. In this way, the user may be given all relevant information needed to process the event without having to have done any earlier discovery. After taking whatever action is appropriate in response to the received
event, the user may loop back to receive the next event, suspending again if there is no event available to process.

Combinations of polled and event driven operations are allowed, and are expected to be common. A user application may complete a discovery operation, then spawn a sub-process or thread to handle an event queue, while actively interacting with management instruments in other processes or threads. Or, the event queue may be polled periodically with a non-blocking call, rather than causing suspension of a process.

Note, however, that any call to the HPI may cause a context switch from the calling process, and may thus be interrupted by normal OS means (for example, due to signal processing in UNIX). A particular HPI may implement most or all calls as remote procedure calls, or may suspend the calling process while it fetches requested information from the management infrastructure.

**Conclusion**

The Service Availability™ Forum Platform Interface provides a standard interface for the rich management capabilities found in the carrier-grade hardware platforms used with applications that must 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™ middleware or other carrier-grade software can be developed that is adaptable to the underlying system capabilities, thus avoiding a “least common denominator” view of the system.

There are strong motivators for platform vendors to support this interface; even without the Service Availability™ middleware layer envisioned by the Service Availability™ Forum, the platform interface provides tremendous value to users by enabling application software to be written that is portable to different high-availability platforms.