NPF HA architecture model and framework Implementation Agreement

July 6, 2004
Revision 1.0

Editor(s):

Sanjeev Verma, Nokia, sanjeev.verma@nokia.com

T.Sridhar, FutureSoft, sridhar@futsoft.com

For additional information contact:
The Network Processing Forum, 39355 California Street,
Suite 307, Fremont, CA 94538
+1 510 608-5990 phone ✦ info@npforum.org

Table of Contents

1 Revision History

Revision	Date	Reason for Changes
1.0	07/06/2004	Created Rev 1.0 of the implementation agreement

2 Scope and Purpose

This document describes NPF SW HA architecture framework and NPF SW APIs that will be used by applications. The objective of the HA framework is to

- Leverage existing work in supporting high availability.
- Provide service continuity in addition to high availability.
- Allow implementation of use cases identified by the HA group.
- Permit extensions for supporting use cases that are not covered currently.
- Minimize the changes to existing NPF SW APIs.

3 Normative References

The following documents contain provisions, which through reference in this text constitute provisions of this specification. At the time of publication, the editions indicated were valid. All referenced documents are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

1. NPF.2003.296. NPF HA use case and requirement,
2. Software API Framework Implementation agreement, IA, NPF, Version 1.0
3. NPF.2003.012. Proposal for High availability architecture model.
4. NPF.2002.240.27. NPF Packet handler API.
5. SAI-AIS-A.01.01. Service Availability Forum Application Interface Specification.
6. NPF2004.036. NPFHA Service API.

4 Acronyms and Abbreviations

The following acronyms and abbreviations are used in this specification:

- Card - Line card or control card is referred to a Card.
- Resource - A resource is a logical or physical entity that is managed by HA middleware. Resource may be either HA aware or Non-HA aware. Resource registers with a HA middleware using a component Name.
- HA aware Resource - A HA aware resource is a resource that uses HA middleware APIs and implements functions that need to be invoked by the HA middleware and reports its states and availability information to the other HA aware resources through the HA middleware. Resources cooperate among themselves and provide periodic status to the HA middleware in the form of events. The HA Task Group restricts HA-aware resources to software applications that run on the line and control cards.

  There may be situations, for example, when an HA aware resource first registers with the HA middleware and then forks several child processes (or child resources). In this situation, only the parent process is registered with the HA middleware – i.e. the HA middleware manages the parent process only. . Child processes that need HA services should explicitly register with the HA middleware either under the same component name as parent or may use different component name.
- Non HA aware Resource - Resources that neither register nor use the HA middleware functions and services are classified as non-HA aware resources. A non-HA middleware resource is one that does not provide redundancy. The HA middleware running on line and control card can only perform basic operations like Start and Stop on these resources. The required level of management and monitoring of these resources depends upon the underlying operating system and defining such requirements is beyond the scope of this task group.
- Resource pool - A resource pool is a collection of one or more processes that are registered under the same component name. Resource pool can either reside within a single card or be distributed across several cards.

  A line card or control card is considered as a single unit. One or more applications running under a control or line card may use HA middleware and its services. For example, two control units, say CE1 and CE2, might be running two HA aware applications, say BGP routing daemon and OSPF routing daemon. Each of these HA applications is uniquely identified under a CE and the redundancy is considered across CEs. For instance, CE1 may be in active state and CE2 may be in standby (or hot standby) state. Also, HA resources should be same in both the CEs.
- HA Server (HAS) - Each line card or control card runs an instance of HAS. An HAS is a server that implements the HA API. An HA Server running in line or control cards discovers each other, periodically synchronizes their states and provides a notion of HA middleware to the applications.
- HA-API - The HA framework provides a set of HA APIs to build highly available systems that provide continuous service. It consists of two sets of APIs namely Availability Management Function API(AMF) and Service API (SE). Each HA resource must implement HA AMF API and SE API's.
- HA-FAPI - FAPI implementation that can invoke the HA Service or HA application management API is called HA-FAPI.
- HA-SAPI - SAPI implementation that can invoke HA Service or HA application management API is called HA-SAPI.
- API - Applications Programming Interface
- CE - Control Element also referred as control card
- FAPI - NPF Functional API

- FE - Forwarding Element also referred as line card
- HA - High Availability
- NE - Network Element
- NP - Network Processor
- NPE - Network Processing Element
- NPF - Network Processing Forum
- NPU - Network Processing Unit (same as NPE)
- SAPI - NPF Service API
- FAPI - NPF Functional API
- ForCES - Forwarding and Control Element Separation
- HAS - HA Server
- HA-SAPI - HA aware SAPI
- HA-FAPI - HA aware FAPI
- RHAS - Root HA Server
- BHAS - Backup HA Server
- HAS SET-ID - Unique HA set identifier
- HA-ID - Unique identifier for HAS within a HA SET

5 HA Overview

Highly available systems are designed to protect against network and operational failures. This is usually achieved via redundancy within each network element. Also, network elements are moving from a monolithic software entity to a more distributed function. The high availability functionality should support this distributed architecture. To support high availability in telephony networks, redundancy was built into each network element. However network elements such as routers have evolved from a monolithic software piece to a distributed software and hardware entity. Network elements may have to maintain per-user or aggregate states to satisfy the service requirements of emerging real-time services. Hence, network elements need to provide high-availability features such as fail-over, load-balancing, state replication and resource redundancy in order to avoid disruption in service. This implies that network elements should support high availability features such as fail-over, load-balancing, state replication etc.

This document describes an HA architectural model for existing NPF SW framework to build highly available systems in the NPF software environment. Next, this document describes how the Service Availability (SA) Forum API can be used within the NPF HA API and describes how the SA Forum services can be seamlessly integrated into the NPF HA middleware.

6 HA Framework

Figure 1 describes the NPF SW framework. The HA API can be logically classified as an operational API, within the NPF SW framework. 1. NPF SAPI, FAPI and other operational API 2 can invoke the HA API. The HA API enables checkpoint and event services and supports full resource management functions to provide seamless redundancy/fail-over mechanisms for applications. The API does not provide direct interfaces to the NPE.

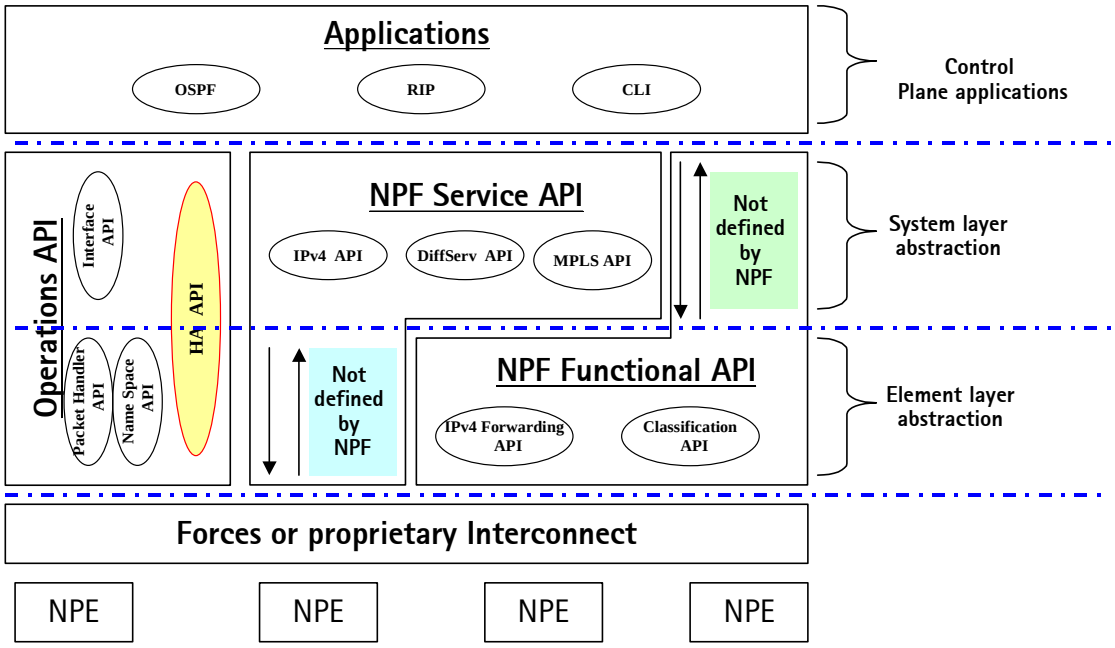


Figure 1 NPF SW HA framework

6.1 HA architectural elements

Figure 2 shows the architectural entities within the HA framework along with their interfaces. The card shown in the figure could be a control or line card. An application may be HA aware or Non-HA aware. An HA aware application/resource interfaces to the HA middleware and reports its state and availability information to the other resources through the HA middleware. The HA middleware periodically keeps track of the HA aware applications by performing health checks.

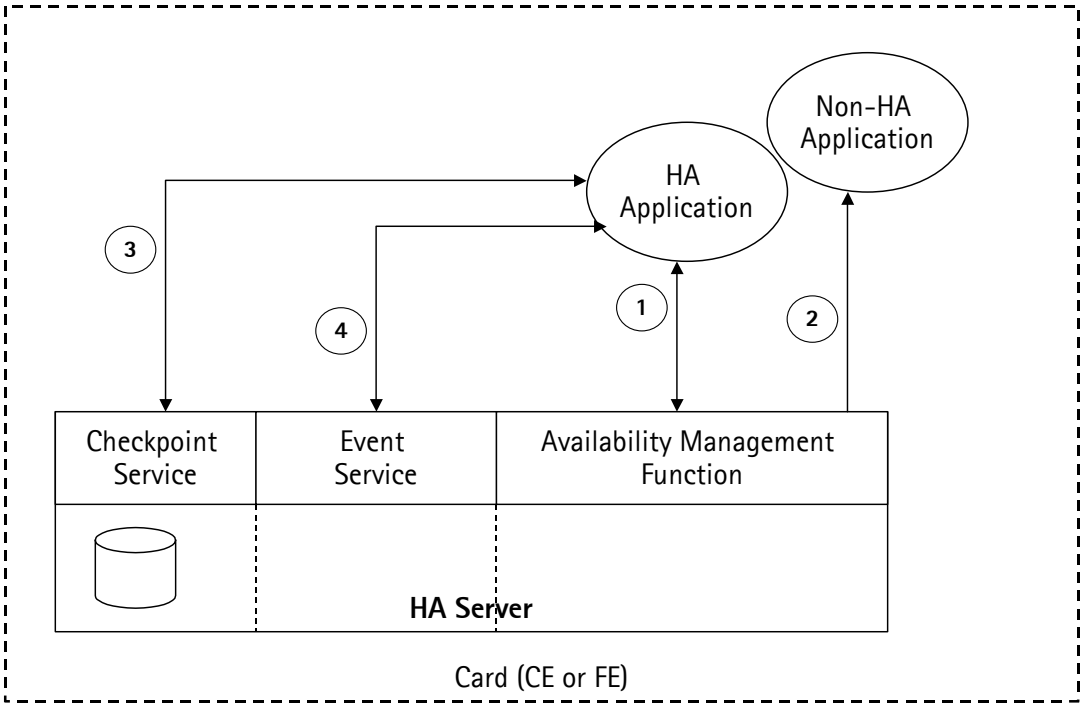


Figure 2 HA architectural entities

Non-HA aware applications do not register with the middleware. Depending upon the system implementation, the HA middleware can only perform basic operations like Start and Stop on the non-HA aware applications.

The HA Server (HAS) is a server that implements HA API and/or HA service API. Only one instance of HAS is allowed to run in a Card. The HAS can be started either manually or at boot time depending upon operator configuration. HA Servers in various Cards discover each other and synchronize the states of resources across Card. The HAS to HAS interaction is beyond the scope of this work. The HAS entities together provide the notion of the HA middleware to the applications on the various cards.

The HAS provides two sets of APIs for HA aware applications – the Availability Management Function (AMF) API and the HA Services API (SE – not to be confused with the SAPI). Each HA resource must implement HA AMF API and provide following functions:

- Registration and deregistration of the HA resource.
- Health monitoring and HA state maintenance.
- Resource and resource pool management.
- Events and error reporting.

The Services API is used to access the services provided by the HA function. The two identified services are checkpoint (for replication management) and event services (for notification of status change). Each HA aware application/resource uses these Services API, as shown in Interfaces 3 and 4 of Figure 3.

6.2 HA API reference point

In order to provide continuous service to applications, the NPF SW HA framework and applications need to provide a set of services and functions. Figure 3 depicts interface reference points namely Hm, Hs, and Hp.

- Hm provides a set of APIs that are part of HA Availability management functions. Hm interface functions are implemented in two parts. One part of Hm interface functions is implemented by the Availability Management Functions of the NPF HA framework. (described earlier). The other part of the Hm interface functions are implemented by each HA aware resource. These functions are called resource state management functions. The NPF HA framework will invoke these APIs for resource management and health monitoring.

- Hs are sets of API functions that are part of HA Services API- the currently defined ones being checkpointing and event services.

- Hp is a set of API functions that are needed to support multi-vendor HA operation in a given NE. This interface is out of the scope of the NPF HA framework and is not discussed further.

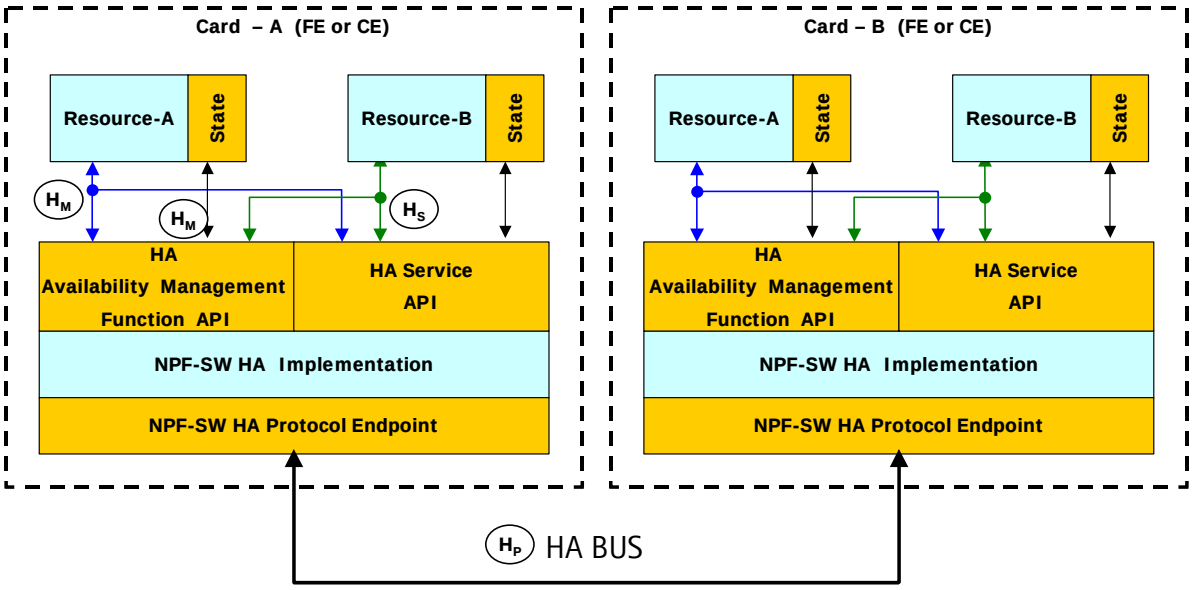


Figure 3 HA reference points

6.3 HA Interaction with NPF API's

The fundamental premise of adding the HA functionality to existing APIs is that the existing SAPI and FAPI interfaces are not to be changed. However, in order to make the FAPI and SAPI implementation HA aware, we introduce additional APIs that are to be implemented by the SAPI and FAPI. An HA application needs to initialize FAPI/SAPI by invoking HA API functions calls and passing information pertaining to the HA resource. FAPI and SAPI implementations that implement these additional API calls are called HA aware FAPI (HA-FAPI) and HA aware SAPI (HA-SAPI) respectively.

We identify common deployment scenarios and provide the interactions between HA APIs with NPF APIs in the rest of this section. This solution, however, necessitates that applications and FAPI/SAPI share the same address space.

6.3.1 HA interaction with NPF Service API

Figure 4 describes the interactions between a HA-aware application and an HA middleware implementation that uses the HA-SAPI.

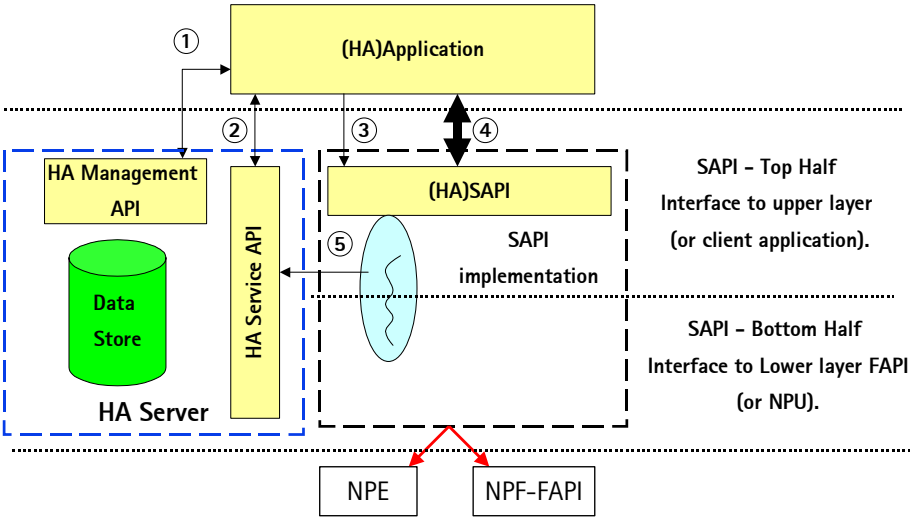


Figure 4 NPF HA and SAPI interaction

Preconditions:

- An application that uses SAPI can be made HA compliant by linking it with the HA SAPI.
- Applications and HA-SAPI that maintain states may use the HA Services through the HA Services API. service through HA-APIs.
- HA aware NPF-SAPI first initializes and registers itself with the HAS (HA middleware). Subsequently, the HAS will keep track of the HA aware SAPI applications.

  Migration from NPF-SAPI to NPF HA-SAPI can be accomplished without making any changes in the existing APIs. However, each NPF-SAPI implementations must implement additional HA specific API calls and appropriate functions to call HA middleware services.

The following describes the various operations performed by the individual entities for realization of the HA functionality.

By HA aware application: (This is performed only once)

1. Initialize the HA context by invoking an HA API call in HA-SAPI .

By HA API extensions in NPF-SAPI: (This is performed only once)

1. Initialize the HA context and internal data structures with reference to the application.

HA implementation inside HA-SAPI:

1. Parse the arguments and process as normal SAPI call.
2. Determine if any of the states needs to updated and appropriately invoke HA service API on behalf of the HA aware application.

Function calls:

```
NPF_error_t  NPF_XXX_HAinit(…);
/* Initialize the HA context for FAPI or SAPI.    This needs to provided by each HA aware SAPI
and     FAPI  */

NPF_error_t  NPF_XXX_HADeregister(…);
/*  De register HA application from the      HA environment */
```

For detailed functional API description refer HA Service API documentation 6.

Figure 4 describes the sequence of interactions between an HA aware application that uses the HA-SPI and the HA middleware.

1. The HA aware application registers with the HA middleware under a unique component name. It also passes the instance identifier (e.g., process identifier) during the registration process.

2. If the HA aware application maintains states and wishes to use the HA service API, it can register with one or more HA services. It does so by performing registration for each HA service separately.

3. After successful registration and initialization, the HA aware application calls HA aware SAPI HA initialization APIs and passes HA context information to SAPI. This information will be later used by the HA-SAPI implementation to invoke HA services on behalf of the application. .

4. HA aware application invokes SAPI function calls.

5. SAPI after completing the normal processing invokes HA services if needed (this operation is SAPI specific)

a. It is possible that SAPI implementation may like to maintain SAPI specific states and replicate these states to standby systems. This is done by making use of the services of the HA middleware via checkpoints and events.

6.3.2 HA interaction with NPF Functional API

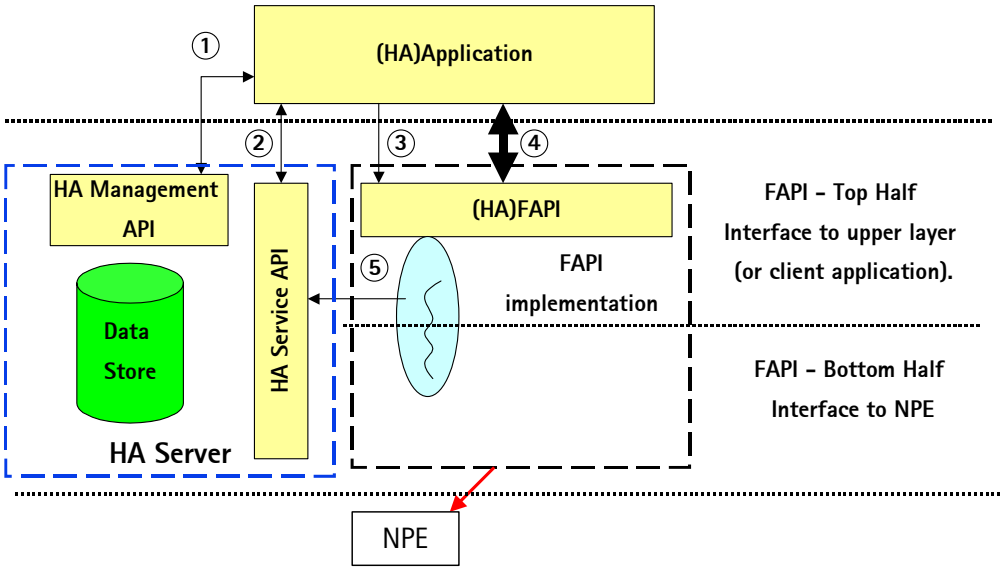


Figure 5 NPF HA and FAPI interaction

Preconditions:

An application that uses FAPI can be made HA compliant by linking it with HA FAPI.

- Applications and HA FAPI that maintain states can use HA service through HA APIs.
- HA aware NPF-FAPI first initializes with HA middleware and thereafter the HA middleware will keep track of the HA aware FAPI applications.
- Migration from NPF-FAPI to NPF HA-FAPI can be accomplished without making any changes to the existing APIs. However, each NPF-FAPI implementation must implement additional HA specific API calls and appropriate functions to call HA middleware services.

The following describes the various operations performed by the individual entities for realization of the HA functionality.

By HA aware application: (This is performed once during initialization)

1. Initialize the HA context by calling HA APIs corresponding to each FAPI call.

By HA API extensions in NPF-FAPI:

1. Initialize the HA context and internal data structure with reference to the application.

HA implementation inside HA-FAPI:

1. Parse the argument and process as normal FAPI call.
2. Determine if any of the states needs to be updated and appropriately invoke HA service API on behalf of the HA aware application.

Function calls:

```
NPF_error_t  NPF_XXX_HAInit(…);
/* Initialize the HA context for FAPI or SAPI.    This needs to provided by    each HA aware
SAPI and     FAPI  */

NPF_error_t  NPF_XXX_HADeregister(…);
/*  De register HA application from the      HA environment */
```

Figure 5 describes the sequence of interactions between an HA aware application that uses HA-FAPI and the HA middleware.

1. The HA aware application registers with the HA middleware under a unique component name. It also passes the instance identifier (e.g., process identifier) during the registration process.

2. If HA aware application maintains states and wishes to use HA service API, it can register with one or more HA services. It does so by performing registration for each HA service separately.

3. After completing successful registration and initialization, the HA aware application calls HA aware FAPI HA initialization APIs and passes HA context information to FAPI. This will be later used by the HA-FAPI implementation to invoke HA services on behalf of the application. .

4. The HA aware application invokes FAPI function calls.

5. FAPI after completing the normal processing invokes HA services if needed. (this operation is FAPI specific)
    a. It is possible that FAPI implementation may like to maintain FAPI specific states and replicate these states to standby systems. Such HA FAPI implementations can create checkpoints or generate events using their associated application context.

Figure 6 describes the sequence of interactions between a HA aware application that uses both HA-FAPI and HA-SAPI.

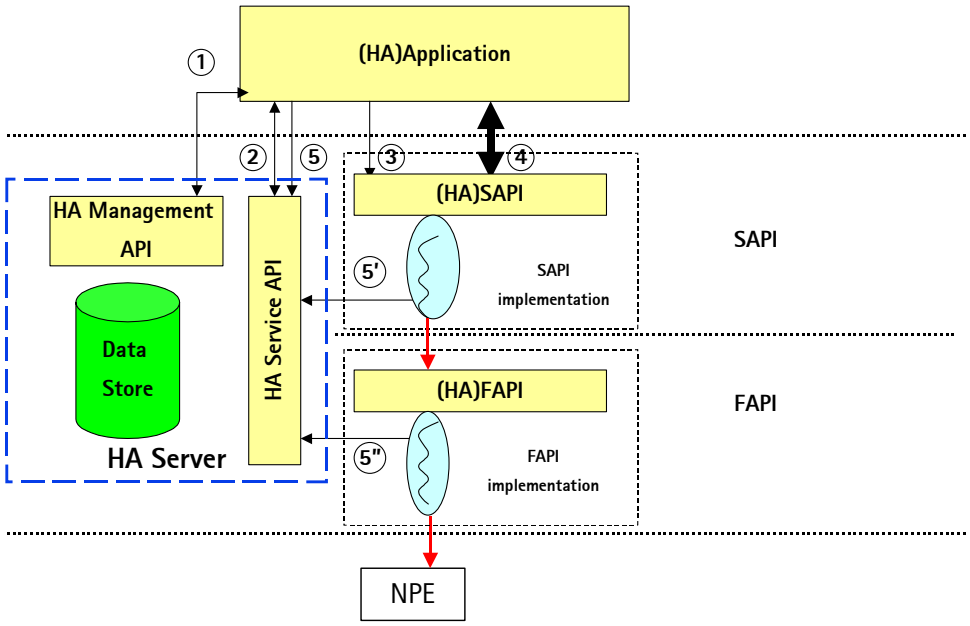


Figure 6 NPF HA and FAPI and SAPI interaction

Level of conformance:

We can have three levels of HA conformance namely:

- Level 1: code compiles and links, so if an app consists of multiple components some of which are HA aware and others not, ensuring level 1 compliance will allow all components to co-exist in a single load.
- Level 2: compliance would be the case where some components are HA aware and others implement just the stop, start interfaces.
- Level 3: compliance will be where components are HA aware and utilize various NPF HA API to implement HA.

7 NPF/SA Forum conceptual model

Here we describe the SA Forum APIs 5 and map them to NPF specific components.

7.1 Mapping SA Forum entities to NPF

7.1.1 System description

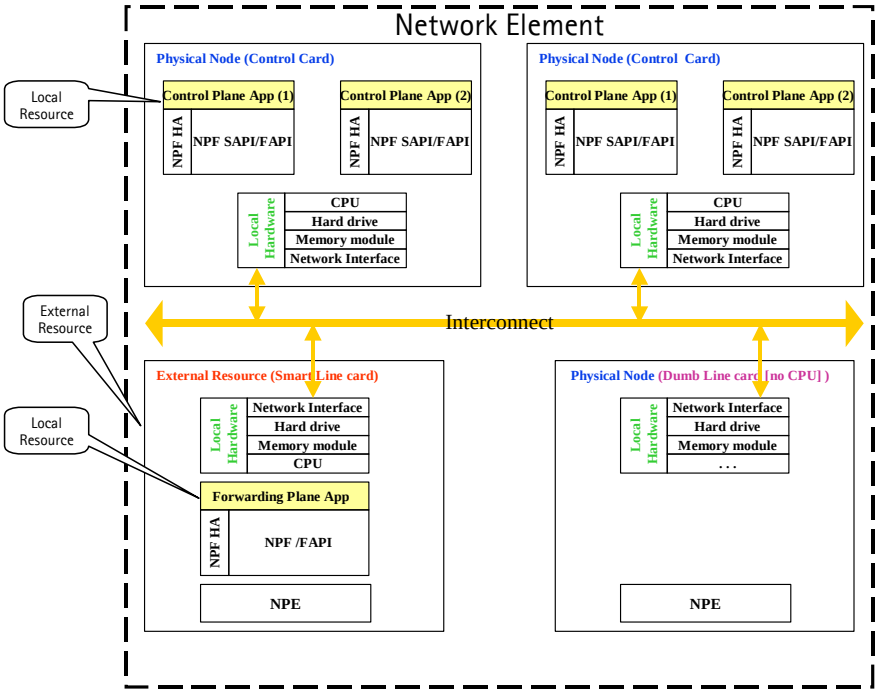


Figure 7 SA Forum/NPF Physical entities mapping

The resource is a basic entity that is managed by the HA middleware. Resource can either refer to a physical entity like card or refer to an application process running in the hardware. For easier management, the SA Forum divides a given system in to various Cards like node, cluster etc. Figure 7 illustrates the mapping of these entities to the NPF physical entities.

The physical node is a resource that is intelligent enough to manage its own local resources. The line card or control card in a given network element is an example of physical node. Some sort of communications mechanism interconnects the physical nodes present in a given network element. Resources that are contained in a single physical node are called local resources. This includes both local hardware and local software resources.

Example of local hardware resource units are hard disk drive unit, flash memory etc. Examples of local software resources are software applications like an interior gateway protocol (IGP), exterior

gateway protocol (EGP) or IPsec application. In the NPF software context, only the software resources are relevant.

The external resources are not local resource but are managed and controlled remotely. For example, consider a case where network element contains a dumb line card, which does not have intelligent software to monitor the local resources. In this situation, the control card HAS may periodically monitor the status of the dumb line card. The dumb line card in this case is an external resource and the HAS in the control card monitors the line card and its resources.. The HA middleware encompasses the proxy process functions.

7.1.2 Logical entities

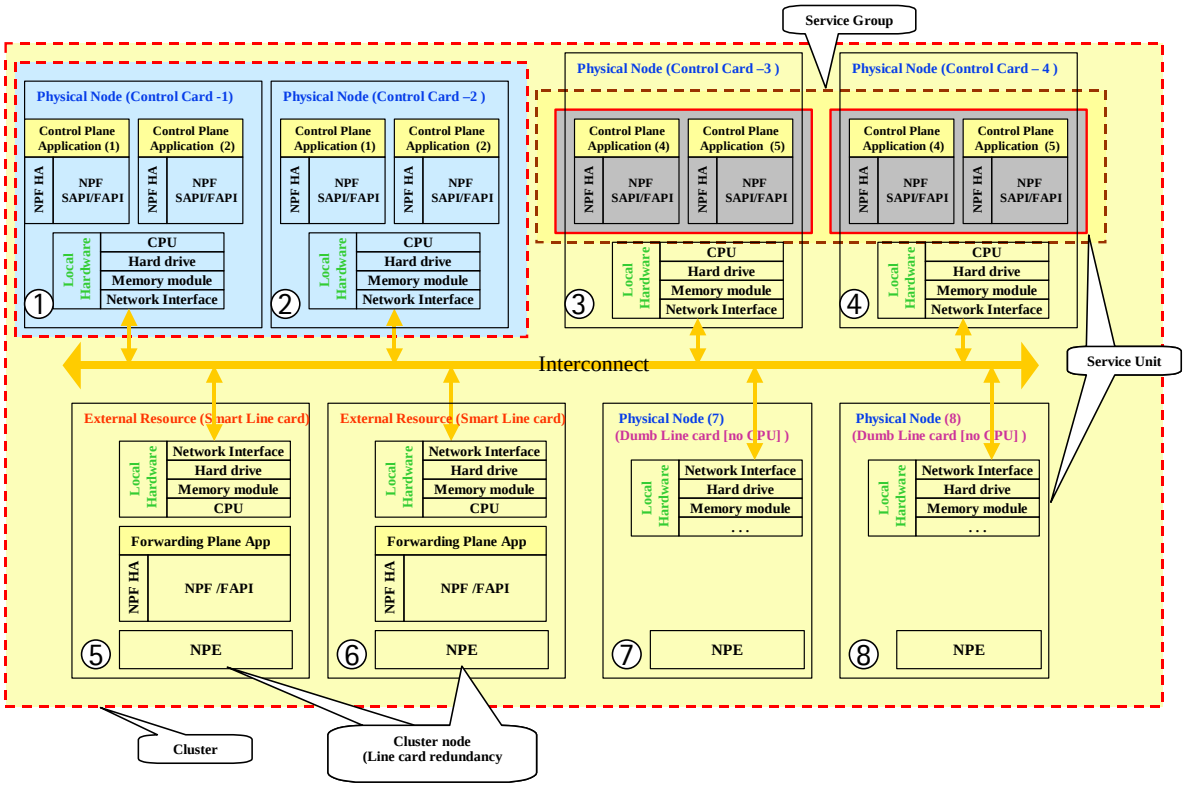


Figure 8 SA Forum/NPF logical entities mapping

For continuous service, finer control and monitoring, the SA forum has defined logical entities like cluster, service unit, service group etc.

Cluster:

Cluster is a collection of physical units that provides same set of services managed by a single HA middleware.

Control cards 1 and 2 in Figure 8 describe a cluster formed by physical nodes to provide control card redundancy mechanism. Physical nodes that are part of the cluster are called cluster nodes.

Figure 8 shows another cluster formed by the Line cards 5 and 6 to provide line card level redundancy.

A Cluster should have the following characteristics:

- Nodes that are configured to be part of a cluster should be able to communicate with each other.
- All the physical nodes in a cluster need to be managed by a single HA middleware
- All the physical nodes that are part of a cluster should have adequate system level resources to run the supported applications.

In the NPF context, most of the control functions reside in a single network element, so the partitioning of the physical network element is out of the scope of this discussion.

Service Unit:

A Service unit is a collection of one or more hardware and/or software components. Service unit is a unit of redundancy.. In the NPF context, a service unit is identified as follows: Service unit may be used as follows:

- For example a service unit may refer to a control or line card. In this case the resources that are contained in the card needs to be made redundant.

- In another example, we consider a router that provides complete routing service by supporting both EGP and IGP protocols. In this case, one process (software component) will run EGP protocol and one or more processes (OSPF, IS-IS or RIP) will run IGP protocols. The collection of processes providing routing service form a service unit Each process (or resource) of a service unit is assigned a service unit type

- Another example is multiple instances of same resources running in a single card.

- In yet another example, a Service Unit refers to a group of resources that are part of a single service. The examples of services are routing service, QoS service and Security service. Each service may be composed of one or more software applications that implement NPF SW (FAPI or SAPI) APIs.

Service Group:

A Service Group is a logical entity that consists of one or more identical service units. Figure 8 shows a service group consisting of identical service units (card 3 and card 4). A Service group has an associated redundancy model, to describe service availability. We restrict ourselves to the 1:1 redundancy model as it is the most common model.

Service and Component Instance:

Depending upon the existing configuration and HA state, the appropriate card (physical node) will be made active or inactive following which the service unit will be instantiated with appropriate HA state. For example, in Figure 8 control card 1 contains two-control applications 1 and 2 that belong to a single service unit. (Think of this as a router providing routing service, where one component runs an EGP protocol and another one runs an IGP protocol). Control

card 2 provides the identical service; both control card 1 and 2 together form a service group to support 1:1 redundancy model.

If the control card 1 is active, the HA middleware will instantiate the service unit S1 and make service instance for the corresponding Service. Service unit will initiate two application instances—control applications 1 and 2—to provide the service. Each application under the service unit will register with the HA middleware under a unique component name. This is also referred as component service instance (CSI). In NPF the use of component service instance is same as that of component name.

There is another possibility of configuration since each process provides a different protocol functionality within the overall service functionality. Each process registers with a different name under a routing service and the HA middleware manages each process independently. If a BGP process server forks a process, then the child process needs to perform explicit initialization. The component name and component instance are treated equally.

In NPF each application process that implements the NPF SW API and belongs to a service unit is referred to as a component Instance. Each application process is simple referred to as a component, when more than one such application instances are allowed to run then each component is differentiated by the process Id or some other identifier within the service unit or card. That identifier is referred as component instance identifier.

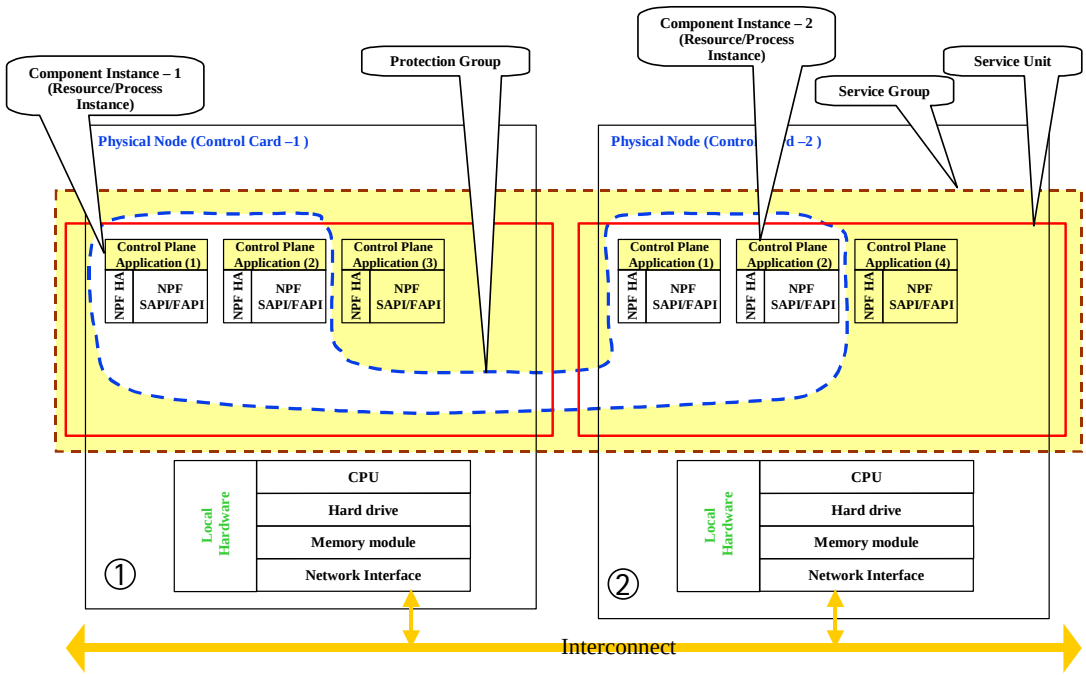


Figure 9 SA Forum/NPF mapping (Service instance and Protection group)

Protection group:

Figure 9 illustrates a protection group consisting of control application 1 in control card 1 and 2 respectively. The control card application running in control card 1 and 2 synchronize their states and work in tandem to provide a HA service. Similarly components running in various service units but within the same service group form protection groups. In NPF the resource pool is same as the protection group.

Relationship between SA Forum and NPF entities:

SA Forum Logical entity	NPF specific term
Node	Control card or line card
Component	Represents either a software or hardware unit. For example, Control application that implements either NPF FAPI or NPF SAPI running as process in card is a software resource. We restrict our HA management function to NPF SW functions

	only.
Local component	HA server running on a card manages the resources that are registered with it and the resource is running locally, then those resources are called local resources. In NPF, the resources are referenced in the HA middleware environment by concatenating the location information, component name and component instance.
External component	In a distributed network element if some of the line cards have intelligence to manage its own resources, then those line cards are modeled as external components. t. Each card maintains resources states about the other cards and resources contained in the other card (external component resources).. In NPF, the resources are referenced in the HA middleware environment by concatenating the location information, component name and component instance.
Physical Unit	Control card or line card
Cluster	Not used in NPF.
Cluster Node	Not used in NPF. Same as line card or control card.
Service Unit	It refers to either a group of applications providing same service and contained in a single control or line card. For example, one or more processes providing routing service, security service or charging service, etc. Or It refers to the hardware unit itself. E.g. non-intelligent line card. In NPF, one or more NPF SW API implementation processes may provide Routing service, QoS Service, and Security Service. The collection of such application processes providing

	a single service can be treated as a Service Unit.
Service Group	Group of control card or line that provides same set of services. For example, for line card redundancy two cards might be placed adjacent to each other and provide same functions. OR If same type of service is being run in one or more cards then they can be configured as a Service Group. In NPF, if service units are available across the cards then they form a Service Group.
Service Instance	Not used in NPF.
Component Service Instance	Each Service unit may be composed of one or more NPF SW implementation processes. Each such application instance belonging to a service unit is referred to as Component Service Instance. NPF focuses mainly on 1:1: redundancy model and component Service Instance refers to Component Name.

7.2 SA Forum/NPF Programming model

7.2.1 SA Forum /NPF deployment view and Interface

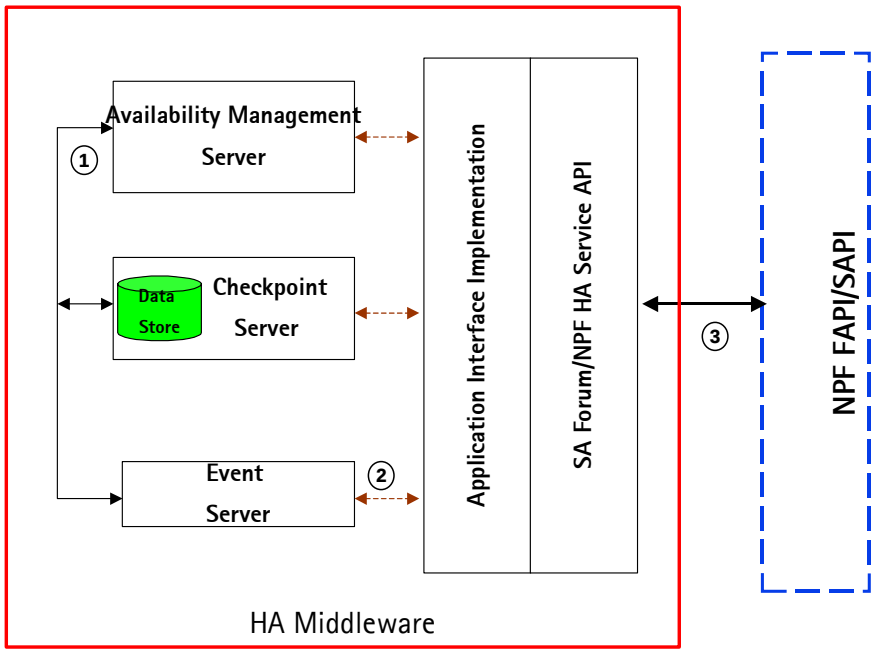


Figure 10 SA Forum interface and NPF SAPI/FAPI deployment view

Figure 10 illustrates a deployment scenario of NPF HA middleware providing SA Forum AIS interface to the applications that implement NPF SW APIs. We have identified Checkpoint services, and Event Services as two basic HA Services that need to be implemented by all NPF-HA middleware implementations. These two services can be implemented as a single process or as two separate entities. These two services depend upon the availability of the availability management server. The communications mechanism between the HA middleware and Application Interface Specification (AIS) is proprietary and it can be either local or remote as illustrated by message 2 in Figure 10 . In order to support multi-vendor operations, the communications mechanisms between three HA services can be either local or remote. NPF SW API compliant applications invoke the AIS APIs as illustrated in message number 3 in Figure 10.

7.2.2 Programming Model

Each component or application that implements the NPF-SW API is a client to the HA middleware, which acts as a server. Components register with the HA middleware and indicate the desired HA middleware service at the time of registration. . We have identified two services namely event and checkpoint services and apart from this the HA middleware provides Availability management framework (Amf) functions to manage and maintain resources running in the HA middleware. The Event and Checkpoint services utilize the Amf services.

The following sections detail the SA forum API. Each API function is self-explanatory. The naming convention for each function is as follows:

type sa<Area> <Object> <Action> <Tag> (<arguments>);

<Area> = Interface area tag

Possible area values that are relevant to NPF are

- Amf = Availability management framework
- Clm = Cluster membership Service (not used in the NPF)
- Ckpt = Checkpoint Service
- Evt = Event Service

<Object> = name or abbreviation of object or service

<Action> = name or abbreviation of action

Some common operations are Request, Response, Set, Get, Open, Close.

<Tag> = tag for the function such as Async or Callback

An example of the SAForum API is:

```
SaErrorT
saCkptCheckpointOpen(const SaNameT *ckeckpointName,
                     const SaCkptCheckpointCreationAttributesT
                        *checkpointCreationAttributes,
                     SaCkptCheckpointOpenFlagsT checkpointOpenFlags,
                     SaTimeT timeout,
                     SaCkptCheckpointHandleT *checkpointHandle);
```

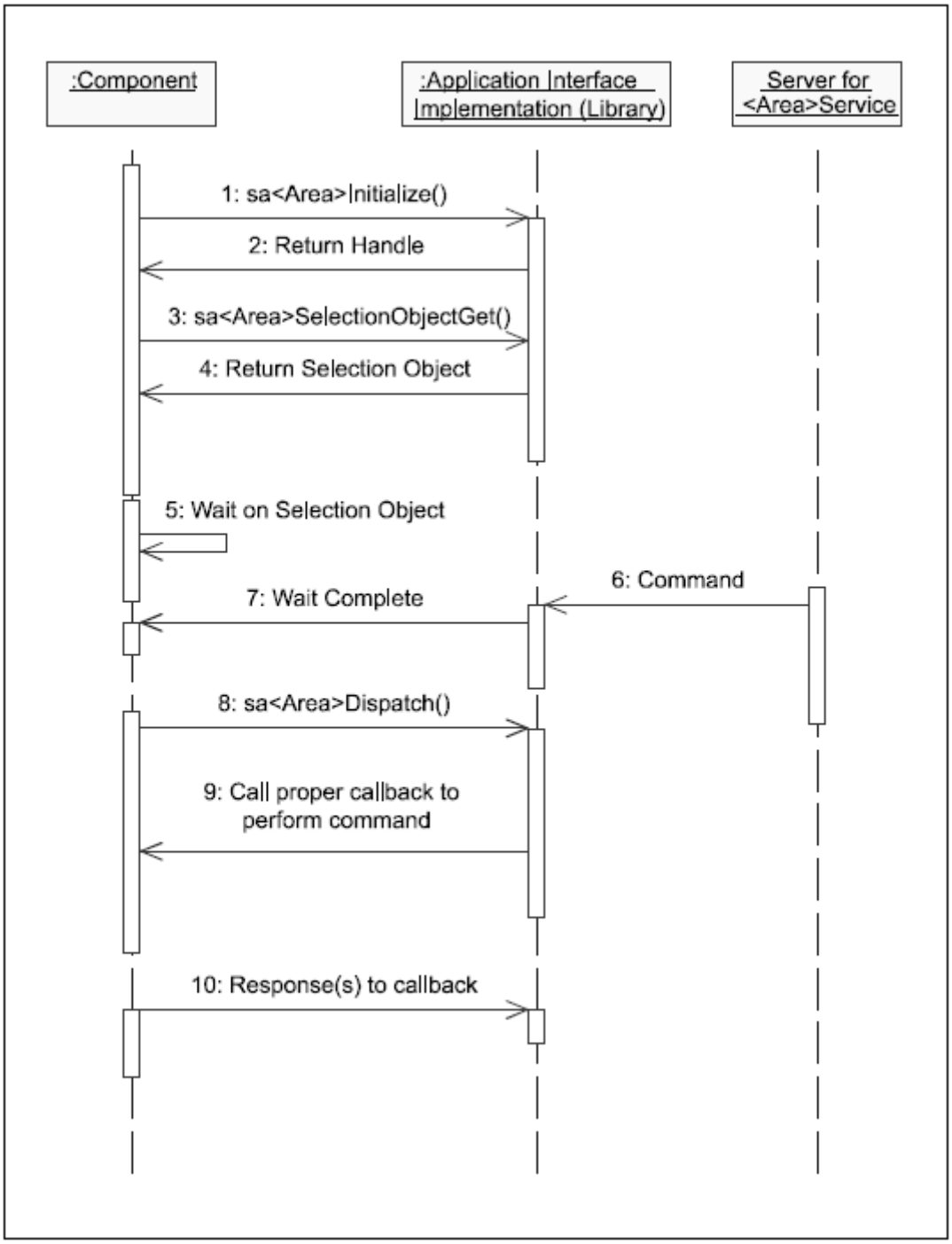


Figure 11 SA Forum API usage model

An example usage of the APIs, which involves a callback mechanism, is as follows:

1. The process within the component invokes the *sa<Area>Initialize()* function to provide a set of callbacks for use by the library in calling back the component.
2. The *sa<Area>Initialize*() function returns an interface handle to the invoking process.
3. The process invokes the *sa<Area>SelectionObjectGet()* function to obtain a selection object, which is an operating system dependent object (e.g., a file descriptor suitable for use in *select()* for Unix/Linux).
4. The interface returns a selection object to the process. This operating system dependent object allows the process to wait until an invocation of a callback function is pending for it.
5. The process waits on the selection object.
6. The area server sends a command over its "private" interface to the library.
7. The library "awakes" the selection object, thereby awaking the process.
8. The process invokes the *sa<Area>Dispatch()* function.
9. The library invokes the appropriate callback function of the process corresponding to the command received from the area server. The callback function parameters inform the process of the specific details of the command issued by the area server or the information provided by the area server.
10. Once the process completes processing the callback, it responds by invoking a function of the area interface. In some cases, more than one response invocation, or no response, may be necessary.

In addition to the callback mechanism, certain functions that the component may invoke are asynchronous, for example, for obtaining information from the area server, via the library, or for reporting errors.

Figure 12 SA Forum API usage sequence

7.2.3 HA State Model

The HA middleware keeps track of all the resources managed by it. At any given point application can change HA state by operator sending commands either through HA middleware or directly to management interface (out side the scope of HA task). The HA middleware state needs to know HA state information to make fail-over decisions. HA state model describes the various states of the resource.

The NPF HA state model depends upon the capabilities of both the line card and the control card. Depending upon the protection group, the HA middleware creates and maintains states across either line cards or control cards. If either a local resource or a control card/ line card itself fails then we recommend switching over to an alternate or standby system. The main reason for using this approach is its simplicity and minimal inter process communications overhead.

Based on the HA state requirements identified so far we need to do two kind of state management in HA framework: state management of line card/control card as a single unit and the state management

of local resources contained in either the line card or the control card. For the purpose of understanding and to be inline with SA Forum state machine terminology, we refer to the card status as service unit administrative status and local resource application status as component resource status.

Administrative Status:

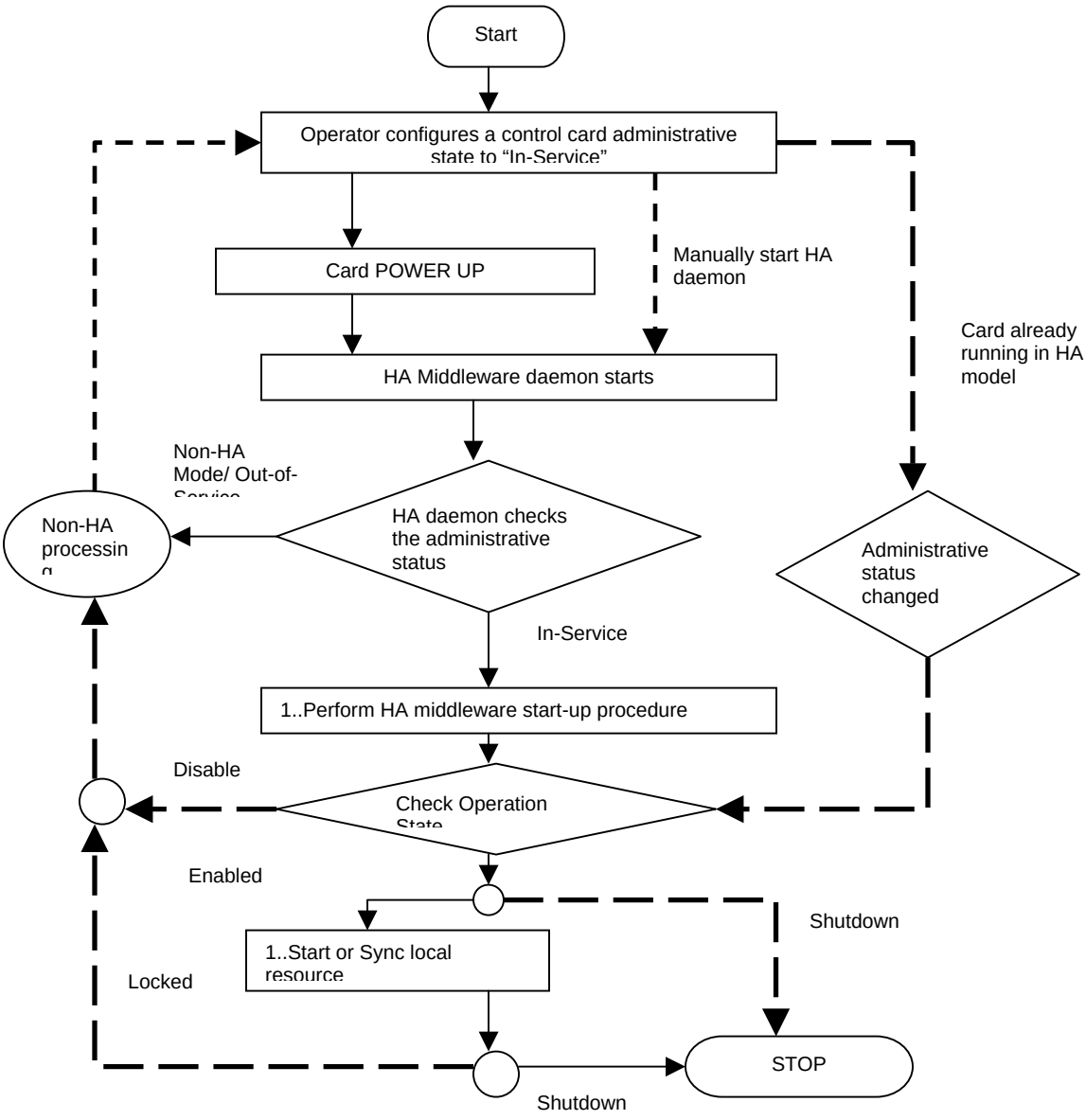


Figure 13 Administrative operations in card

The administrative status represents the status of the complete card or the service unit itself. In NPF HA framework, it refers to the status of the control or the line card. Cards can be in any one of the following administrative states:

- Cards may be physically present in the chassis but all its functions are shutdown or not activated (that is logically disconnected). We refer to this state as “In-Service - disabled”. This is also called as out-of-service state.
- Card may be physically present and all of its functions are activated. We refer this state as “In-Service - Enabled”
- Card may not be physically present or physically present but not powered up.
- Card has been in In-Service and enabled state but the operator wants to perform some software or hardware upgrades and temporarily makes the service state to “In Service - Locked state”. This state is also known as out-of-service state.
- Card or Service unit that was quiesced for the service instance, HA middleware performs switchover operations by making a standby unit to active. HA middleware assigns states to the card or service unit when the switchover is needed it does so by making state changes to the quiesced card or service unit.

Figure 13 describes the complete administrative state transitions. For example, an operator may configure the control card administrative status to one of the above-mentioned states. Depending upon the state transition the HA middleware activates the local resources and reflects the card status on the resources.

The administrative and operational status of a card indicates whether the card is taking part in HA operations or not. It is possible that in a system two control cards may be plugged in and operator wants to make one card as active and another card as standby. When a new card is plugged into the system, it first uses discovery mechanisms to find whether there is any other exiting card in the system with similar configurations. If there is one such existing card in the system, then depending upon the service unit status, the card can start performing HA state management. It is implementation dependent how the administrative or operational states of the cards and/or instances status are initially configured.

For illustration, let us consider a scenario, where we have two control cards, CC-1 and CC-2, with similar hardware and software configurations. Operator wants one of the cards to be in the active state and another card to be in the standby state at any given time. Operator assigns same component name to both the cards, and makes their administrative and operational status as In-Service and Enabled. For card service unit status, operator will configure the card as 1_active_or _1_standby.

- Under this situation when CC1 is powered-up it first checks whether CC2 is already present. If CC2 is not present then CC1 assumes that there is no CC2 and makes its service unit instance status as ACTIVE and activate its local resources accordingly.

- At later time, when CC2 is powered-up, it first checks whether CC1 is present. Since CC1 is present it responds to CC2 with its service unit instance status as ACTIVE. Since the service unit is either configured as “1_active_”or”_1_standby” ; there can only be one active and one standby system at any given instance. CC2 automatically configures its instance state to standby and activate its local resources accordingly.

- During normal operations, if CC1fails, CC2 automatically detects this and changes its state to CC2.

- If an operator by mistake inserts a third control card CC3 with same configurations, CC1 and CC2 will respond to CC3 and ask the CC3 not to start because the system is configured to provide "1_active_or_1_standby" mode only. But if CC2 fails and the operator starts CC3 then the CC1 will make changes in its internal states and treat CC3 as standby.

SA Forum has provided several combinations of redundancy model. NPF will use

- 1_active_or_1_standby: The component or service unit supports either one active or one standby component service instance at any time for control card and line card.

Resource state:

The state of software components running in control or line card is referred as either the resource or the component state in the NPF HA framework. Control card activates each resource depending upon the administrative, operational and instance states.

For example, if a control card state is in service, enabled and active states then all the software processes in the control card will be activated to be in the active state. . If the control card state is in-service, enabled and standby then the HA middleware will activate all the software processes to come up in the standby mode states.

The HA middleware allows control each software process state separately. It is possible to have more than one software application running in the control card. But we recommend that if the control card is configured in active state and the applications are running in active state, operator should launch another application in the same service unit as "active" only. The card is the unit of redundancy in NPF HA.

In NPF, a resource can be in active, standby or busy states. A software application that is performing computation intensive operations and doesn't want to execute any external service request, can make its state as busy. We recommend that this state should be temporary in nature, otherwise it may not be able to perform smooth HA transition if failure is detected in other parts of the network element. Typical example for busy state is when the operator configures a huge number of static routes in the routing process, or when the routing processor performs routing consistency check to ensure the consistency across MPLS, OSPF and other routing processes.

Each resource will have its own HA configuration parameters and we recommend that these parameters be configured in one configuration file to enable interoperability. NPF uses following component state model to represent the Software resource state.

- **1_active_or_1_standby:** This software resource supports either one active or one standby instance at any given time in a control or line card.
- **1_active:** The software resource can have only one active component running at any given time in the control or line card. Typical examples are routing process running in control card.
- **No_State:** For Non-HA application.

APPENDIX A INFORMATIVE ANNEXES

A.1. ANNEX: HA TOPOLOGY

This section describes a mechanism how the different HA servers in various cards can communicate and form the HA middleware.

A.1.1. HA HIERARCHY

The HAS running in a Card needs to maintain the state information of each resource distributed across several Cards. When the number of Card is large say, for example 100's of line cards or control card, fail-over operations may get very complicated. Any resource status change needs to be propagated across the Cards so that HAS running in each Card has the same system-wide view. This might require unnecessary message exchanges among Cards. To minimize unnecessary message exchanges, we can group a set of Cards providing same or similar services into a SET. The HA SET is a collection of HASs that are running in different Cards and manages the same set of resources. The HAS SET is unique under a network element and refers to group of HAS. HA SET is referenced by HA-SET-ID, a unique 16-bit integer value, and manually configured. The HAS that is contained within a HA SET needs to update and synchronize its states with other HASs in the same group. This mechanism reduces the number of messages, increases the manageability and scalability of the system. Figure 14 describes two HA-SET inside a network element. To further reduce the number of message exchanges and assure updates between the HAS in a given set, we introduce the notion of the Root HAS (RHAS). One HAS within a HA set is dynamically assigned the role of RHAS. The remaining HASs within a HA set send resource state updates to RHAS only as depicted in Figure 14.

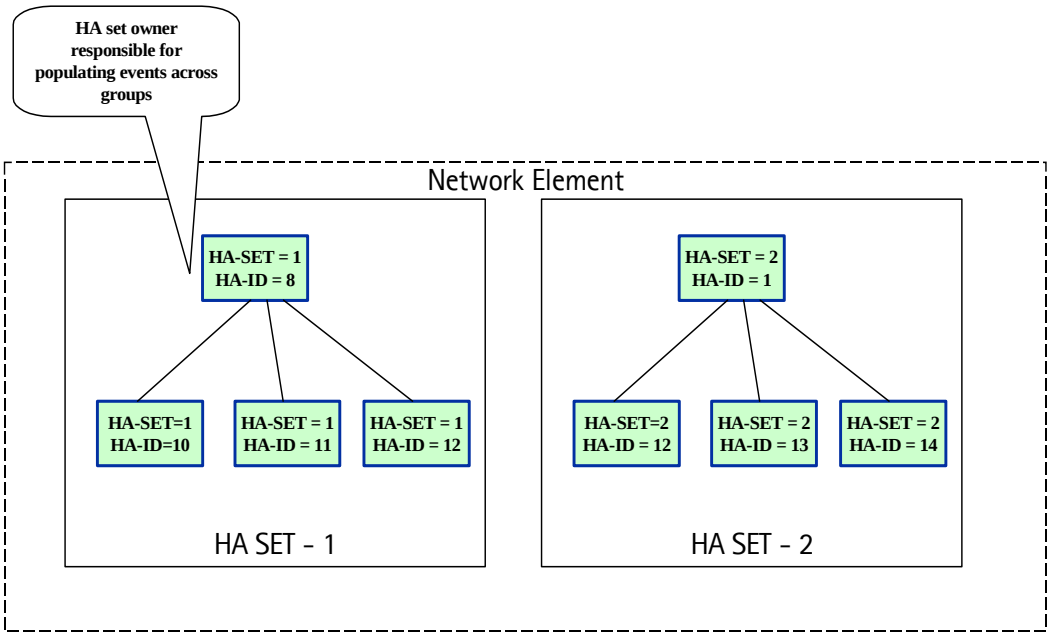


Figure 14 HA group and hierarchy

A.1.2. HA IDENTIFIER

The HAS within a HA SET is uniquely identified by an HA identifier (HA-ID)and its value is computed dynamically by RHAS during startup. Each HAS belongs to the HA set and the HA-SET is unique in the network element. The HA set is identified by a separate identifier, namely HA-SET-ID. The operator configures the HA-SET-ID manually.

In Figure 14 we have shown 8 Cards and each Card runs one instance of HAS and is uniquely identified by an HA-ID namely 8,10,11,12,1,12,13, and 14. Furthermore, HASs with HA-ID 8,10,11,12 belong to HA set 1 and 1,12,13,14 belong to HA set 2. This two HA set within a NE forms a two-tree structure. We have two logical groups or sets in Figure 14. First group consists of HASs with logical ids 8,10,11,12 and the second group consists of HASs with ids 1,12,13,14. The HASs with id 8 and 1 are dynamically assigned as RHASs for their respective logical groups. If there is any state change in an HAS belonging to a HA-Set then it is communicated to the corresponding RHAS. The RHAS then communicates the updates to other members of the HA set through a reliable unicast connection.. We next describe HA-ID numbering mechanism and the RHAS selection process.

In order to reference a HAS running under a given network element, we need to concatenate HA-SET-ID and HA-ID as described in Figure 15

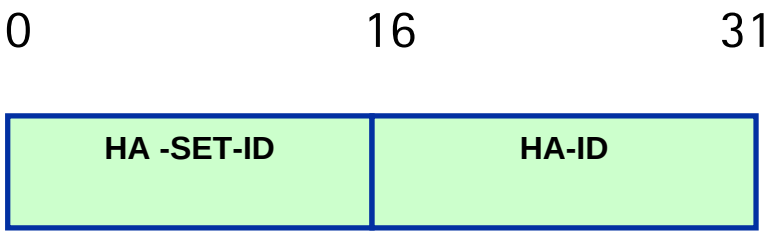
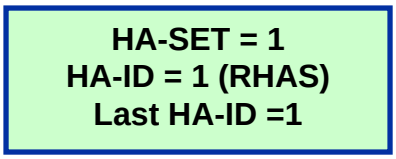


Figure 15 HAS reference

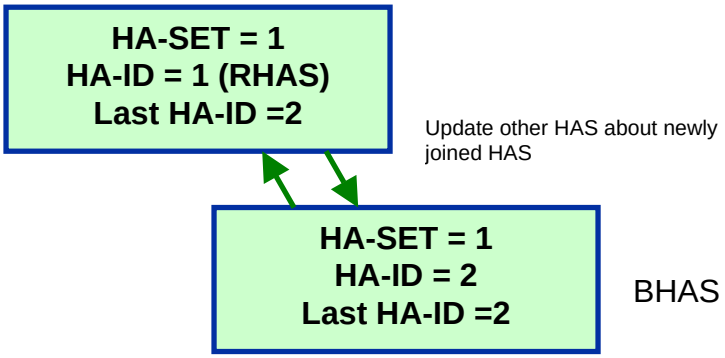
A.1.3. ELECTION OF ROOT HAS

When an HAS is started in a Card, HAS server needs to make sure whether there are any other HASs belonging to the same HA-SET running inside the network element. This is done by generating a HA discovery message on a multicast (layer-2 or laye-3) channel. If none exists, then the HAS will take up the role of Root HAS (RHAS) and it will assign itself an HA-ID = 1.

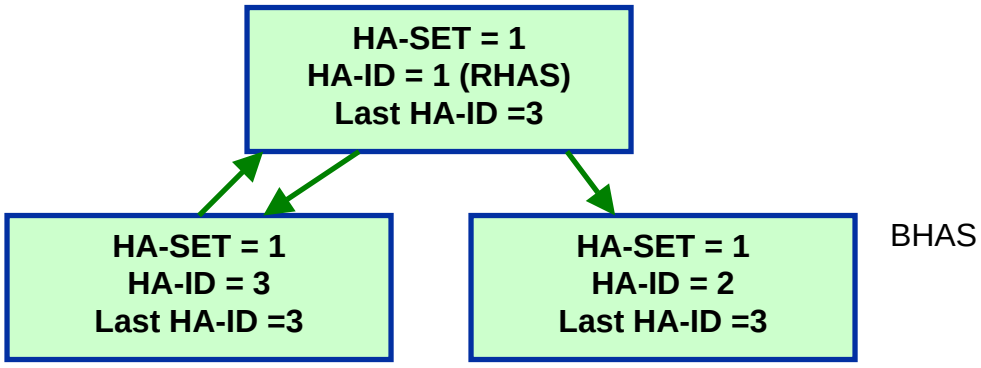
At later point if another HAS becomes active, it will send an HA discovery message on the multicast channel. However, this time RHAS will respond to the newly joining HAS. The response information contains the details of the HAS endpoint information. Upon receiving the response message, the joining server connects to the RHAS through a unicast TCP connection. The RHAS computes an HA_ID and passes the ID to the newly joined HAS. Computation of HA-ID is monotonically increasing in value and it is always one higher than the previously assigned HA-ID. After successful completion of joining process, the newly joined HAS will have HA-ID = 2 and will send an update regarding its readiness to RHAS.



(a) First HAS started



(b) Second HAS started within the same HA set



(c) Third HAS started within the same HA set

Figure 16 HA-ID assignment procedures

A third HAS joining at a later time follows the same procedure and is assigned an HA-ID=3. The status of each HAS and how the updates are made is described in.

A.1.4. REASSIGNMENT OF RHAS

The HAS which first becomes active in a HA-SET will take up the role of the RHAS. If for some reasons the active RHAS fails or is being shutdown normally, then the HAS which has next higher HA-ID will become the RHAS.

Figure 17 depicts the RHAS assignment procedure. Initially, the HAS with HA-ID 1 is the RHAS. Now if this HAS goes down then the HAS with next higher id, which happens to be the HAS with id 2 in the given example, becomes the RHAS. When the HAS that was down comes back then it gets a new id as shown in Figure 17.

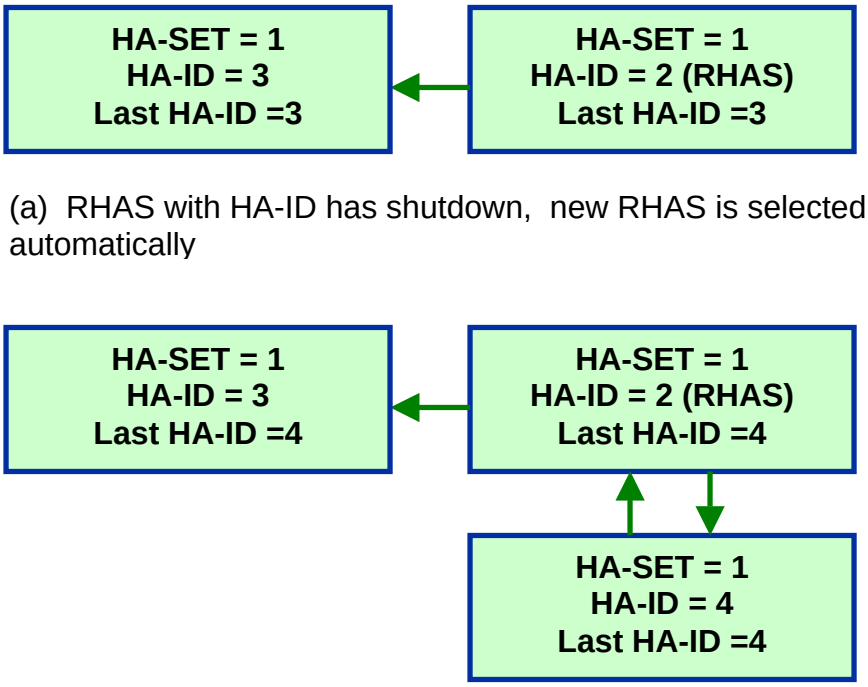


Figure 17 RHAS reassignment

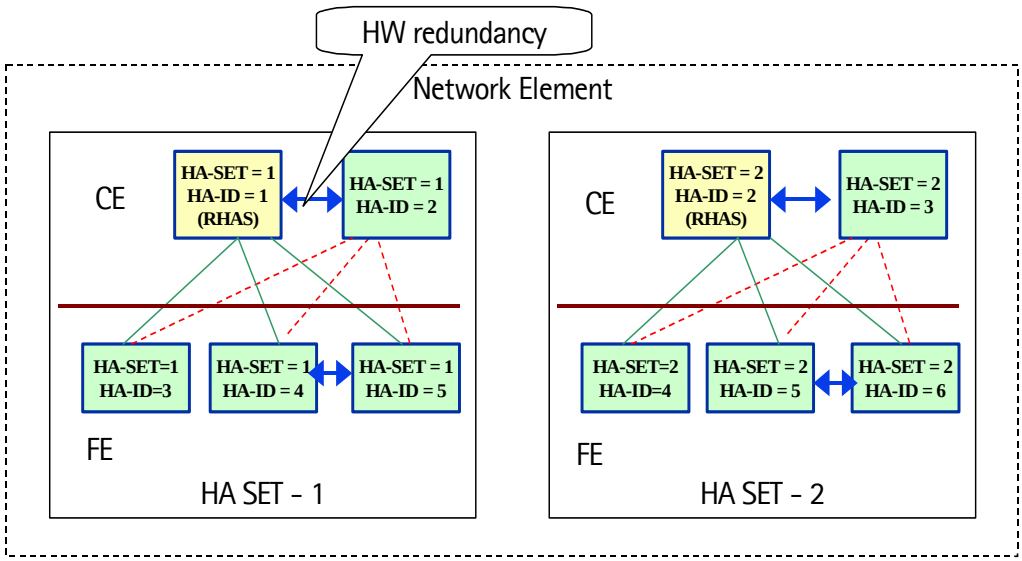
To minimize the fail-over time to the new RHAS and reduce the number of messages exchanged, it is recommended that all HAS should have pre-establish transport level connection with the BHAS (Backup HAS). The BHAS is the pre-assigned backup HAS with higher HA-ID that takes the role of RHAS when the currently active RHAS fails. Each HAS may be connected to both RHAS and the Backup HAS. This mechanism reduces the latency time during fail-over operation.

A.1.5. HA SWITCH OVER

HASs within a HA-SET synchronize all the states. This means that for seamless continuity service switchover should happen within the HA-SET and operator needs to carefully group the similar applications under one HA-SET for continuous service.

Switchover operation can happen across control cards or across line cards. It may be possible that line cards may have intelligent software running on it and if there are any existing hardware redundancy mechanisms then these should be conveyed to RHAS during the HAS startup procedure. This way HASs in a HA-SET know to which HAS to switch in the event of a failure.

Figure 18 illustrates switchover operations between line cards and also between control cards in a NE. We have four control cards and two of them belong to HA-SET 1 and other two belong to HA-SET 2. Six line cards with HA-ID 3,4,5 are part of HA-SET1 and line cards whose HA-ID are 4,5 and 6 are part of HA-SET 2. There is an existing hardware redundancy mechanism between the control cards in both the sets and also there is an existing hardware redundancy mechanism between line cards with HA-ID 4 and 5 in HA SET 1 and line cards with HA-ID 5 and 6 in HA-SET 2.



1) HAS Set owner needs to be running on ACTIVE CE
2) HAS running on FEs may get updates from other FE HAS server belonging to same HA Set
3) HAS Set owner will synchronize the HAS and HAR states with other HAS Set owners

Figure 18 Switch over operation in a HA-SET

Each Card runs one instance of HAS. If HAS with HA-ID =1 fails in HA-SET =1 then HAS with HA-ID=2 in HA-SET =1 will become RHAS.

In order to provide fast switchover and provide line cards redundancy, the switchover should happen based on the hardware redundancy mechanism. These configuration information needs to be conveyed to the HAS running on the Card. Figure 12 shows an example, where we have one active (HA-ID=5) and two standby line cards (HA-ID=3,4) in the HA-SET 1. Now if the active line card

fails then the switchover is done to line card with id 4 instead of line card with id 3 to take advantage of existing hardware redundancy mechanism between line card 4 and 5.

APPENDIX B ACKNOWLEDGEMENTS

Working Group Chair: Vinoj Kumar

Task Group Chair: Ram Gopal. L

The following individuals are acknowledged for their participation to High Availability task group teleconferences, plenary meetings, mailing list, and/or for their NPF contributions used for the development of this Implementation Agreement. This list may not be all-inclusive since only names supplied by member companies for inclusion here will be listed. The NPF wishes to thank all active participants to this Implementation Agreement, whether listed here or not.

The list is in alphabetical order of last names:

Bachmutsky Alex(Nokia)

Balakrishnan Satosh (Intel)

Damon Pilippe (IBM)

Keany Berny(Intel)

Keisu Torbjorn (Ericsson)

Kumar Vinoj(Agere)

Lewing Van (PMC-Sierra)

Muralidhar Rajeev (Intel)

Papp William (Nokia)

Renwick John (Agere)

Sam (Intel)

Sridhar T(FutureSoft)

Stone Alan (Intel)

Vandalore Bobby (Nokia)

Vedvyas Shanbhogue (Intel)

Verma Sanjeev(Nokia)

APPENDIX C LIST OF COMPANIES BELONGING TO NPF DURING APPROVAL PROCESS

Agere Systems	FutureSoft	Nokia
Cypress Semiconductor	Intel	PMC Sierra
Ericsson	IP Infusion	Sun Microsystems
ETRI	Kawasaki LSI	Xilinx