# NSI Connection State Machine

## Overview

Each NSA runs a (local) state machine for the section of the connection it is responsible for, i.e. the local resources it manages through its NM, and the segments provided by the child NSAs in case of a federating NSA. An NSI State Machine (NSI SM) is spawned when a Connection Request is received from an RA. The NSI SM interacts through message exchange with RA (through provider NSI), a number (0-N) PAs (through requester NSI) and the local system (NRM, PCE, Scheduler, etc.). This is depicted in the figures below.

**State Machine**

**NSI (provider)**

**NSI (requester)**

**NRM, PCE, Scheduler**

Figure : State Machine interactions with RA, PAs and local system.

Figure 2 shows an example service tree for a certain path. Message exchanges are point-to-point following the black arrows. A is the user (requester) agent originating the connection request. NSAs B and C are federating agents; for the particular example depicted here, B interacts with C and D only (and upstream with A). C interacts with E and F (and upstream with B).

Not shown in the figure is the possibility of a third-party agent, such as run e.g. by a NOC, acting as a requester to the NSA in its domain. Such an agent can issue requests relating to a circuit it does not “own”.

Notifications

Replies

Requests

A

B

C

D

E

F

Figure : A connection scenario depicting the service tree structure and the interactions between the NSAs.

## Message and Event Handling

Messages received through the NSI are dispatched by the (NSA global) message handler to the State Machine instance identified through the CID field in the Service Common Attributes section of the message. If the CID is unknown, and the Service Primitive is *Reserve*, a new NSI SM instance is created. Otherwise, and an *Unknown Connection* reply is sent back, and the message is discarded.

Each state machine instance has its own event handler. Events are processed in FIFO manner. Reception of messages is a sub-set of possible events. Timer triggers (reservation start, reservation end) are another. Some events trigger a state transition in the top-level state machine, while some require additional conditions to be fulfilled. One example of the latter is the need for all provider agents to confirm a successful reservation before transition from Reserving to Scheduled state (c.f. below). Another interpretation is that some states are state machines themselves, and some events will trigger local transitions within the top-level state.

## Inter-NSA Protocol messages

As general rule, only events (protocol state changes) which trigger an Action are notified through the NSI protocol. All other (informational) messages are out of scope of the connection service, and relegated to e.g. monitoring service. Protocol messages are received by a Message Handler, and dispatched to the SM in the form of Events. The dispatcher maintains a FIFO queue of events per SM, local events and received messages are delivered to the SM in the time order or arrival.

Notation used for message IDs: ServicePrimitive/Content. “Content” intends to identify what the message content is in particular in the responses.

Notation used for events: Source\_Event.

|  |
| --- |
| **NSI Protocol Messages** |
| **Message** | **Src→Dst** | **Meaning** | **Associated Event** |
| Reserve/Request | RA→PA | PA spawns a new SM instance entering in Reserving state |  |
| Reserve/Confirmed  | PA→RA | PA moved to Scheduled state | PA\_Res\_Conf |
| Reserve/Failed | PA→RA | Reservation request failed, PA moved to Terminating state.Note: used also to indicate resources no more available if in Scheduling state | PA\_Res\_Failed |
| Cancel/Request | RA→PA | Request connection termination: release provisioned resources (if any) and remove reservation from calendar | RA\_Cancel |
| Cancel/Acknowledged | PA→RA | *Needed only if we want to wait for confirmation from PAs before deleting the SM instance – do we?* |  |
| Provision/Request | RA→PA | Request provisioning | RA\_Provision |
| Provision/Confirmed | PA→RA | PA moved to In-Service | PA\_Prov\_Conf |
| Provision/Failed | PA→RA | Provisioning failed, PA moved to Releasing state (and back to Scheduled) | PA\_Prov\_Failed |
| Provision/Release | RA→PA | Request Release of Resources, but keep reservation. | RA\_Release |
| Provision/ReleaseACK | PA→RA | Confirm resources released | PA\_Rel\_Conf |
|  |  |  |  |

## Connection States

7 States are defined, listed below. States 1-3 relate to reservation management (path computation and scheduling), while states 4-7 are relate to provisioning and operational status of the connection.

1. **Reserving** (Initial state): Starts path computation and scheduling; awaits termination either as successful or failed reservation.
2. **Scheduled**: Connection is scheduled but not provisioned
3. **Terminating** (Final state): Frees schedule slots for this reservation.
4. **Provisioning**: Starts provisioning; awaits confirmation from PAs
5. **In-Service**: All resources are provisioned
6. **Releasing**: Releases provisioned resources, but does not affect the reservation status; transitional state (back) to Scheduled.
7. **Releasing\_Terminate**: Releases provisioned resources, but does not affect the reservation status; transitional state to Terminating.

The Reserving initial state is a State Machine in itself, as it needs to interact with PCE, Scheduler and the remote domains’ PAs.

## NSI Connection Events

The complete list of events is given here:

1. Res\_Success: Path computation and connection reservation was successful.
2. Res\_Failed: Path computation or connection reservation (scheduling) failed.
3. PA\_Res\_Conf: PA confirmed reservation successful (moved to Scheduled state)
4. PA\_Res\_Failed: PA failed to reserve resources (moved to Terminating state)
5. Timer\_Start: Reservation start time arrived.
6. RA\_Provision: Received RA request to provision (RA moved to Provisioning state).
7. Prov\_Start: Received provisioning request from RA AND current time > reservation start time.
8. RA\_Cancel: received terminate request from RA. This releases provisioned resources (if any) and clears the reservation from the calendar.
9. PA\_No\_Resource: at least one PA is unable to maintain the scheduled resource allocation
10. Timer\_End: End of reservation arrived
11. PA\_Prov\_Conf: Received Prov\_Conf from a PA (PA moved to In-Service state)
12. PA\_Prov\_Failed: Received Provision/Failed from a PA (PA moved to Releasing state)
13. Prov\_Conf: ALL PAs have confirmed provisioning complete (moving to In-Service state)
14. RA\_Release: RA requests release of provisioned resources; reservation is kept
15. PA\_Rel\_Conf: PA confirms release of resources (moved to Scheduled state)
16. Rel\_Conf: ALL PAs have confirmed that resources are released

## NSI State Diagram



## NSI Actions and Transition Tables

|  |
| --- |
| **Reserving** |
| **Event** | **Action** | **Next State** |
| Entry | Start Path Computation |  |
| PA\_Res\_Conf | Mark this PA as Confirmed;If (all PAs confirmed) trigger Res\_Confirmed; | Reserving |
| PA\_Res\_Fail | Trigger PCE action | Reserving |
| Res\_Confirmed  | Send Reserve\_Confirmed to RA;Set timer event to start time | Scheduled |
| Res\_Failed | Send Cancel to PAs;Send Reserve/Failed to RA | Terminating |
| RA\_Cancel | Send Cancel to PAs | Terminating |

|  |
| --- |
| **Scheduled** |
| **Event** | **Action** | **Next State** |
| Entry | None |  |
| Timer\_Start | None | if (received RA\_prov) Provisioning;else Scheduled; |
| RA\_Provision | Send RA\_Provision to PAs;Mark RA\_provision as received; | If (t\_current > t\_start) Provisioning;Else Scheduled; |
| RA\_Cancel | None | Terminating |
| PA\_No\_Resource | Send PA\_No\_Resource to RA | Terminating |
| Timer\_End | None | Terminating |
|  |  |  |

|  |
| --- |
| **Terminating** |
| **Event** | **Action** | **Next State** |
| Entry | Send Cancel to PAs;(wait for confirmation before..?)Clear reservation from calendar; | None |

|  |
| --- |
| **Provisioning** |
| **Event** | **Action** | **Next State** |
| Entry | Send Provision/Request to PAs |  |
| PA\_Prov\_Conf | Mark PA provisioned;If (all PAs provisioned) trigger Prov\_Conf; | Provisioning |
| PA\_Prov\_Failed | Send RA\_Release to PAs | Releasing |
| Prov\_Conf | Send Provision/Confirmed to RA | In-Service |

|  |
| --- |
| **In-Service** |
| **Event** | **Action** | **Next State** |
| Entry | None |  |
| RA\_Release | Send Provision/Release to PAs | Releasing |
| RA\_Cancel | Send Prov\_Terminate to PAs | Releasing\_Term |
| Timer\_End | None | Releasing\_Term |

|  |
| --- |
| **Releasing** |
| **Event** | **Action** | **Next State** |
| Entry |  |  |
| PA\_Rel\_Conf | Mark PA as released;If (all PAs released) Trigger Rel\_Conf; | Releasing |
| Rel\_Conf | Send Provision/ReleaseACK to RA | Scheduled |

|  |
| --- |
| **Releasing\_Term** |
| **Event** | **Action** | **Next State** |
| Entry |  |  |
| PA\_Rel\_Conf | Mark PA as released;If (all PAs released) Trigger Rel\_Conf; | Releasing\_Term |
| Rel\_Conf | Send Provision/ReleaseACK to RA | Terminating |

## Data Plane Error Handling

This State Machine does not handle data plane faults at this point. There’s three possibilities:

* Fault notification is in the domain of monitoring. The SM does not reflect the data plane health state, it deals only with the reservation and provisioning status. The resources are reserved and provisioned, even if data cannot flow due to e.g. a fire cut. Operators (NOC or automated agents) are notified through the monitoring system, and can take decision on whether to release resources.
* Data plane failure leads automatically to release of resources. This can be easily added through additional Events/Transitions. IMO, this is too drastic, many faults are survivable.
* SM is extended to add states reflecting data plane failures. Not sure it’s needed at this point, as no real actions apart from potential release are possible here. A possible extension in V2?

## Control Plane Error Handling

Still on the to-do list…

* Check and provide for possibility of timing mismatch (e.g. PA\_Rel\_Conf received in Provisioning state).
* Define timeouts and actions when encountered.
* Bootstrap procedure after NSA failure.
* …