Add/Update/Delete eNB & Update gNB REST APIs Design

Introduction

This design matches the requirements in the following USs:

RIC-431 - Getting issue details... STATUS

RIC-432 - Getting issue details... STATUS

RIC-433 - Getting issue details... STATUS

RIC-434 - Getting issue details... STATUS

RnibReader Changes

We shall add a new proto message, named `AdditionalCellInformation`:

```
message AdditionalCellInformation {
        float cell_latitude = 1;
        float cell_longitude = 2;
        float antenna_height = 3;
        float antenna_azimuth_direction = 4;
        float antenna_tilt_angle = 5;
        float antenna_max_transmit = 6;
        float antenna_max_gain = 7;
        uint32 sector_id = 8;
}
```

- Expand **ServedCellInfo** (enb.proto) with this proto message:

enb.proto

```
message Enb {
    EnbType enb_type = 1;
    repeated ServedCellInfo served_cells = 2;
    repeated string gu_group_ids = 3;
}

message ServedCellInfo {
        ...
        AdditionalCellInformation additional_cell_information = 16;
}
```

- Expand **ServedNRCellInformation** (gnb.proto) with this proto message:

gnb.proto

```
message Gnb {
    repeated ServedNRCell served_nr_cells = 1;
    repeated RanFunction ran_functions = 2;
}

message ServedNRCell {
    ServedNRCellInformation served_nr_cell_information = 1;
    repeated NrNeighbourInformation nr_neighbour_infos = 2;
}

message ServedNRCellInformation {
        ...
        AdditionalCellInformation additional_cell_information = 16;
}
```

E2 Manager Configuration Changes

Add a new configuration key to the yaml file.

Recently we've added:

`stateChangeMessageChannel: RAN_CONNECTION_STATUS_CHANGE`

Now we shall add a new channel for RAN changes:

`ranManipulationMessageChannel: RAN_MANIPULATION`

This key, as others, should be read to the app's configuration which is injected to various flows.

This key will be used by RnibWriter for adding/updating/deleting eNBs and updating gNBs.

RnibWriter will trigger Sdl.SetAndPublish method, sending the RAN_MANIPULATION channel, and one of the following events:

```
<RAN_NAME>_ADDED
<RAN_NAME>_UPDATED
<RAN_NAME>_DELETED
```

RnibWriter Changes

- Modify the following method:

```
SaveNodeb(nbIdentity *entities.NbIdentity, nodebInfo *entities.NodebInfo) error
```

For Gnb type, we execute sdl's *Set* method.

For Enb type, we execute sdl's *SetAndPublish* method instead, sending RAN_MANIPULATION channel and <RAN_NAME>_ADDED event.

- Add the following methods:

```
RemoveServedCells(inventoryName string, servedCells []*entities.ServedCellInfo) error
UpdateEnb(nodebInfo *entities.NodebInfo, servedCells []*entities.ServedCellInfo) error
DeleteEnb(nodebInfo *entities.NodebInfo) error
```

I've already added the below method as part of the gNB update cells US (for a case we get an empty list from the client):

```
RemoveServedNrCells(inventoryName string, servedNrCells []*entities.ServedNRCell) error
```

Add eNB REST API

Agenda

- `POST /v1/nodeb/enb` is triggered, request body is validated
- Execute `GetNodeb` with RAN name coming from the request.
  - If we have a DB error or if it already exists error
- Create a NodebInfo struct with an **Enb** configuration, populate it with the request data and set its connection status to **DISCONNECTED**
- Execute `SaveNodeb`, where `sdl.SetAndPublish` will be triggered for eNB type, sending the **RAN_MANIPULATION** channel and the **<RAN_NAME>_ADDED** event.
- Return `201 CREATED` with NodebInfo response.

POST /v1/nodeb/enb

Request Body

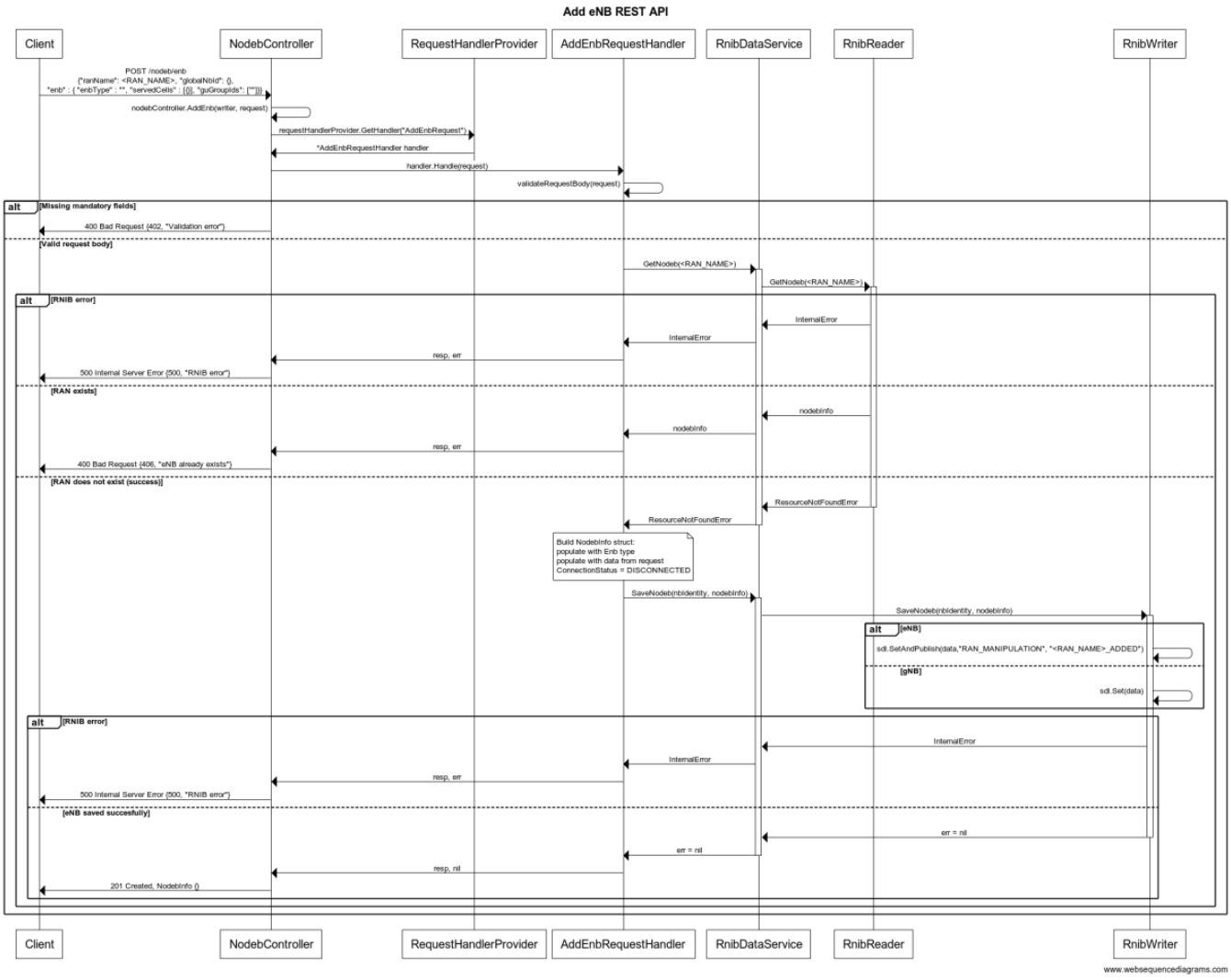
```
{
        "ranName": "",
        "globalNbId": {
                                        plmnId: "",
                                        nbId:        ""
        },
        "ip": "",
        "port": 1234,
        "enb": {
                        "enbType": "",
                        "servedCells": [{}],
                        "guGroupIds": [""]
        }
}
```

AddEnbRequestHandler

AddEnbRequestHandler

```
type AddEnbRequestHandler struct {
        logger                                  *logger.Logger
        rnibDataService                 services.RNibDataService
}
```

Sequence Diagram



Update eNB REST API & Update gNB REST API

Agenda

eNB	gNB	Notes
`PUT /v1/nodeb/enb` `/<RAN_NAME>` is triggered	`PUT /v1/nodeb/gnb` `/<RAN_NAME>` is triggered	Request body is validated
Execute `GetNodeb` with RAN name coming from the request		If we have a DB error or if it doesn't exist error
Execute `RemoveServedCells` to remove the existing eNB cells	Execute `RemoveServedNrCells` to remove the existing gNB cells	
Set the nodeb with the request's data		
Execute `UpdateEnb`	`Execute UpdateGnbCells`	Set cells in both Cell ID key & PCI key Set nodeb in both RAN ID key & RAN NAME key Execute `sdl.SetAndPublish`, sending the **RAN_MANIPULATION** channel and the **<RAN_NAME>_UPDATED** event.
Return 200 OK with NodebInfo response.		

PUT /v1/nodeb/enb/<RAN_NAME>

Request Body

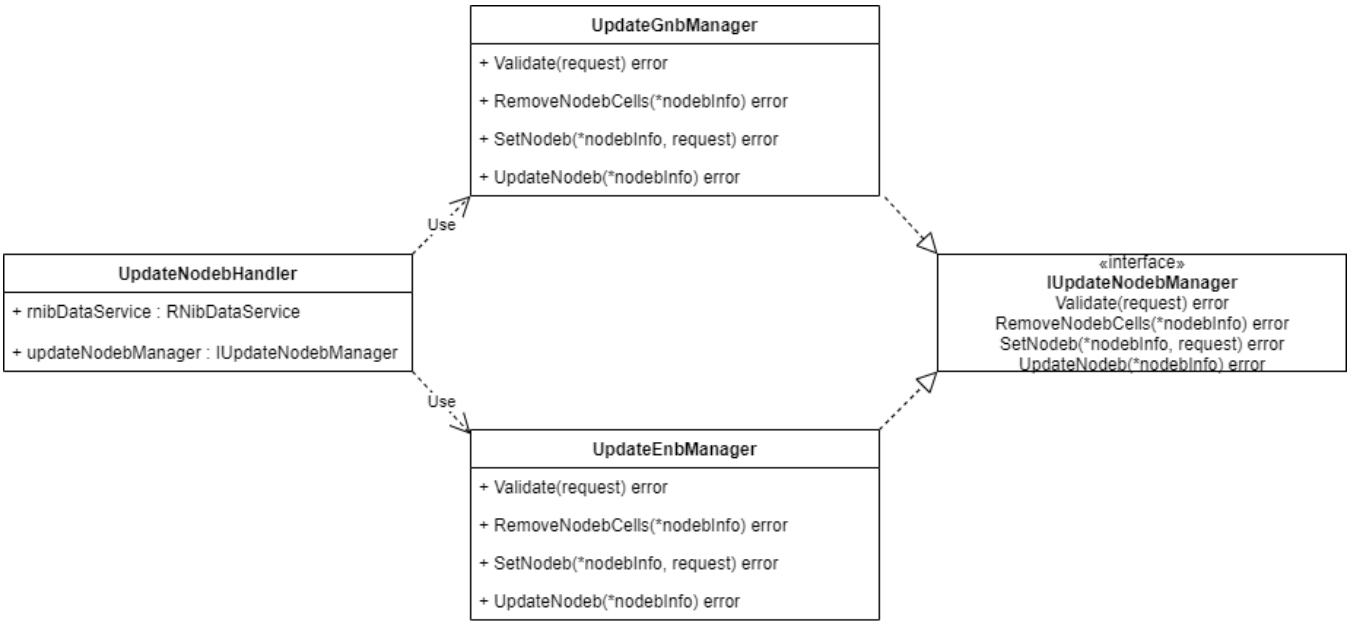
```
{
        "enb": {
                        "enbType": "",
                        "servedCells": [{}],
                        "guGroupIds": [""]
        }
}
```

PUT /v1/nodeb/gnb/<RAN_NAME>

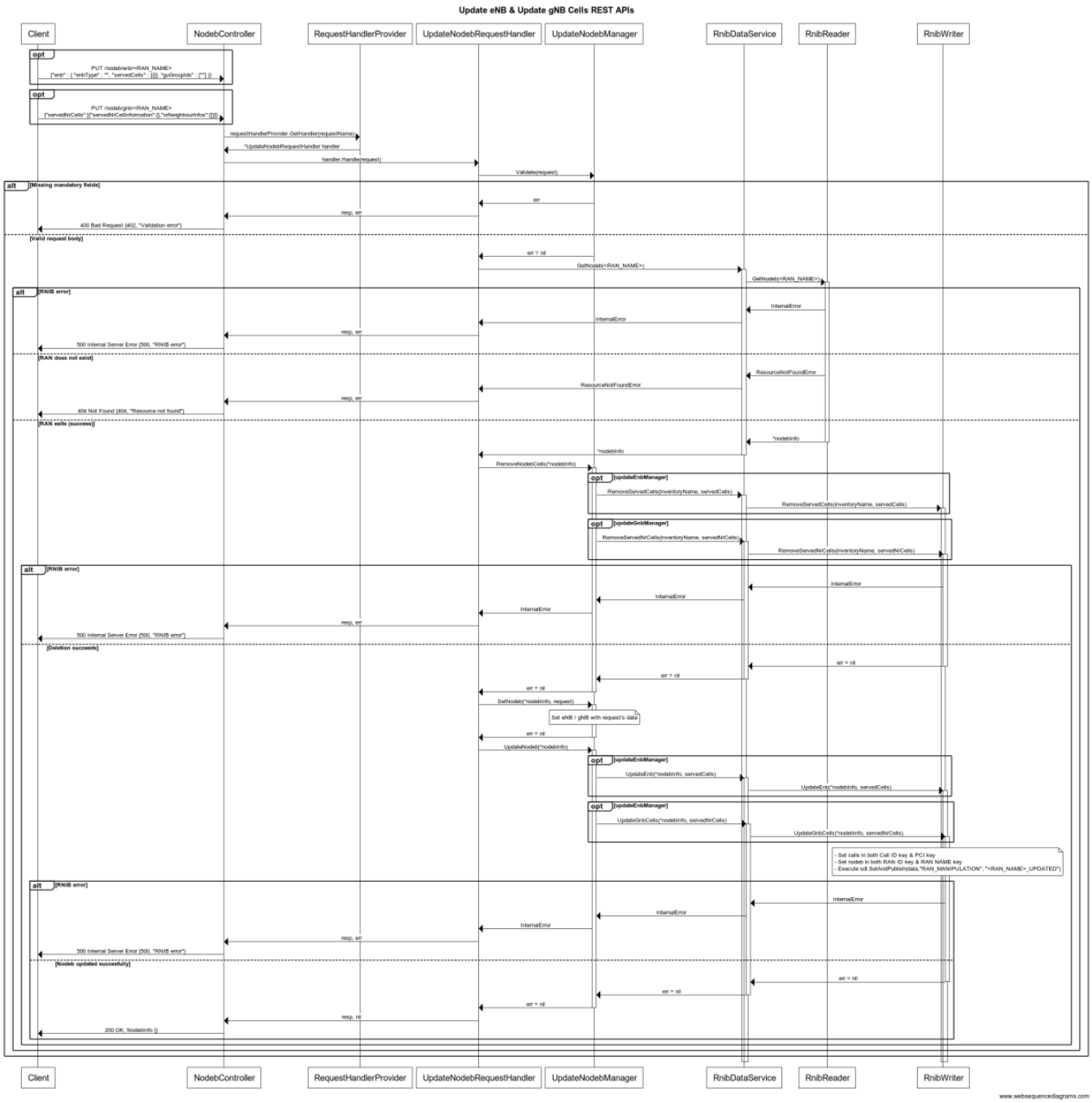
Request Body

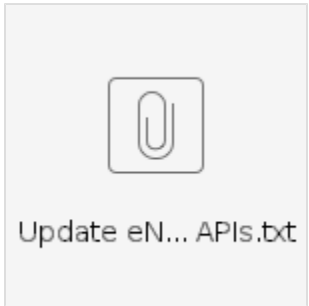
```
{
  "servedNrCells": [
    {
      "servedNrCellInformation": {},
      "nrNeighbourInfos": [{}]
    }
  ]
}
```

UpdateNodebHandler



Sequence Diagram





Delete eNB REST API

Agenda

- `DELETE /v1/nodeb/enb/<RAN_NAME>` is triggered.
- Execute `GetNodeb` with RAN name coming from the request.
  - If we have a DB error or if it doesn't exist error
  - If nodeb is of type gNB error
- Execute `RemoveEnb`
  - Remove cells from both Cell ID key & PCI key (call *RemoveServedCells*)
  - Remove nodeb from both RAN ID key & RAN NAME key
  - Remove nbIdentity
  - Execute sdl.SetAndPublish(data,"RAN_MANIPULATION", "<RAN_NAME>_DELETED")
- Return `204 NO CONTENT` response.

DELETE /v1/nodeb/enb/<RAN_NAME>

DeleteEnbRequestHandler

```
type DeleteEnbRequestHandler struct {
        logger                                  *logger.Logger
        rnibDataService                 services.RNibDataService
}
```

Sequence Diagram