034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.1.1

Requirements and specifications of interfaces and architecture for interoperability between NRPS, GMPLS, Middleware

Due date of deliverable: 2007-03-01
Actual submission date: 2007-04-13
Document code: Phosphorus-WP1-D.1.1

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
I2CAT Foundation

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

This deliverable specifies a set of interfaces, methods and an architecture that will be implemented and demonstrated within the Phosphorus experimental activities that are going to be carried out within WP6. The document will focus its effort on defining the requirements needed to perform interoperability between different NRPS, and also between the standard GMPLS Control Plane and the middleware through the Network Service Plane.

Table of Contents

Table of Figures

0 Executive Summary

This Deliverable, named “Requirements and specifications of interfaces and architecture for interoperability between NRPS, GMPLS, and the Middleware”, defines the requirements of a set of interfaces and methods that will be implemented and demonstrated within the Phosphorus experimental test-bed. The Network Resources Provisioning Systems (NRPS) that this project involves are, ARGON, DRAC and UCLPv2.

The set of interfaces defined in this document provide different types of interoperability. Firstly, a NRPS Adapter placed on top of each NRPS system implements a generic interface that abstracts the particularities of each NRPS. It also provides interoperability with a GMPLS Control Plane through a Web Service interface towards the NRPS Adapter. However, only when the GMPLS control plane has a NRPS on top of it, advance reservations can be guaranteed within the GMPLS domain.

In order to achieve a general interoperability, a Network Service Plane (NSP) with a NRPS Broker inside has been specified. The NRPS Broker deals with the NRPSs in order to provide end-to-end paths, while the NSP manages advance reservations, AAA issues the resource utilization and coordinates the different actions done in all NRPSs. Moreover, another interface has been proposed at the NSP layer that may provide interoperability with the developments done within the GÉANT2 project and under the JRA3 activity (Bandwidth on Demand services). This interface allows an enhanced scalability and an easier future interoperation with other international projects like EnLIGHTened and G-Lambda.

A second set of interfaces specified within the NSP and towards the Grid Middleware has also been specified. Thus, these interfaces will request the NSP to set up the necessary paths to allow any of the applications implemented under the scope of Phosphorus to access the different set of Grid resources needed for their computational purposes.

As during the first phase of the project the interoperability of network and Grid resources will be realised at the middleware layer, this deliverable has only proposed a draft architecture towards the Grid enhanced GMPLS control plane (G^2MPLS) to be developed during the second phase. The G^2MPLS system will directly interact with network and grid resources, and therefore will deal with the NSP when resources under the control of an NRPS are required to reach Grid resources.

1 System architecture

1.1 Objective

The objective of this work package is to define an architecture and a set of interfaces that will provide interoperability between different provisioning systems, the GMPLS control plane and the Grid Middleware, in order to create guaranteed scheduled end-to-end connections and provide advanced functionalities that are required by Grid applications. During Phase 1 (M1-M18) three objectives will be achieved, the interoperability between different NRPSs, the interoperability between the NRPSs and GMPLS and the interoperability between the Grid Middleware and the NRPSs and GMPLS. During Phase 2, the effort will be focused on the interoperability with other systems like the ones developed in WP2 or in GÉANT2 JRA3 [GN2-JRA3].

To achieve such interoperability a two layers approach is presented: the to-be-developed Network Service Plane (NSP) and the Network Resource Provisioning System (NRPS) layer. A Transport Network (TN) or a GMPLS control plane could be found underneath the NRPS layer. Additionally a Grid enabled version of the GMPLS control plane called G^{2}MPLS (developed within WP2), could also be located besides the NSP during Phase 2 of the project.

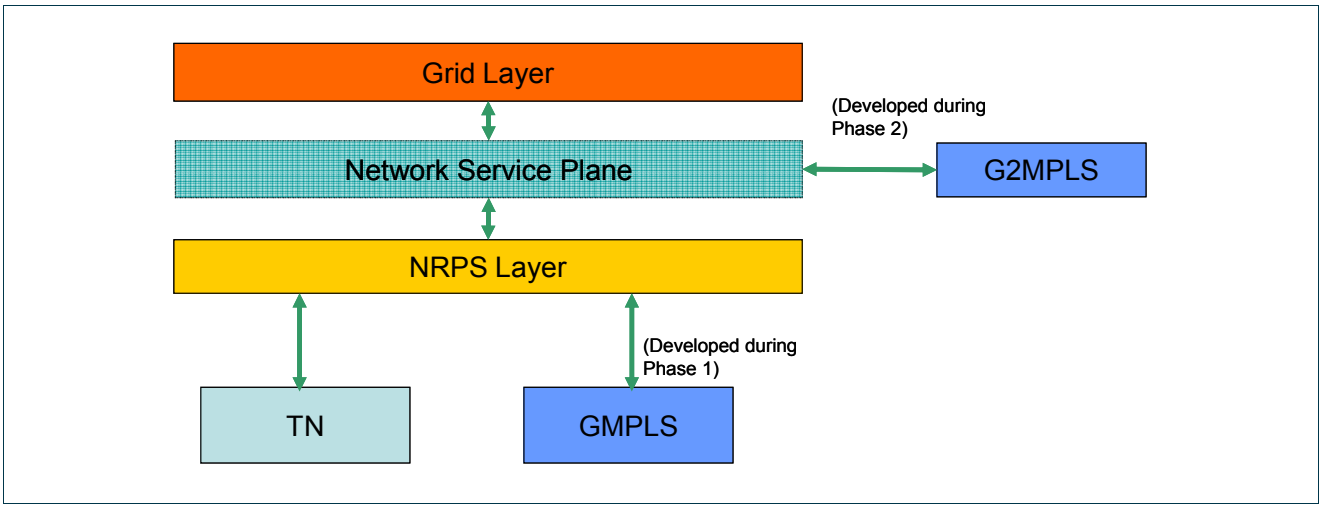

Figure 1.1: WP1 related layers

1.2 Terminology

In these paragraphs different layers used in the architecture will be defined, as well as the most important concepts that will be used in this document:

- Grid Layer: This layer will be developed within WP3 and will be based on the Globus Toolkit 4 [GT4] and UNICORE [UniCore] software. It is responsible for running a set of applications defined by WP3, it also forwards to the Network Service Plane the network requirements to run these applications. The Grid Layer is also referred in this document as Grid Middleware.

- Network Service Plane (NSP): The network service plane is the adaptation layer between the Grid layer and the NRPS layer. It coordinates the different networks in the NRPS layer to meet network resources requests coming from the Grid Layer. This block is a key element to provide interoperability between the different NRPSs, GMPLS and the Grid Layer.

- NRPS Layer: This layer is composed by several Network Resource Provisioning Systems managing different domains. For the scope of the Phosphorus project these NRPSs can be either a ARGON [ARGON], a DRAC [DRAC] or a UCLPv2 [UCLPv2] system. Each system is responsible for the management of the resources available in its domain. This management can be achieved by directly accessing the nodes of the transport network, or, if available, by means of a GMPLS control plane that configures the network.

- NRPS Adapter: Since each NRPS has a particular interface an adapter between the NRPSs and the NSP is needed. The NRPS Adapter is the responsible for the mapping of these interfaces to a common interface.

- Transport Network (TN): within the Phosphorus project a Transport Network is a group of physical devices and links managed by an entity (an NRPS or a GMPLS CP) that allows the transport of information between endpoints.

- Advance Reservations: All the applications that will be used in WP3 have as a requirement the possibility to do advance reservations of resources. This means that network resources can be requested in advance; therefore an advance reservation includes a Start and End Time indicating when this reservation should take place.

1.3 Possible architectures

In this section, some proposed architectures are discussed in order to achieve interoperability between different Network Resource Provisioning Systems (NRPSs), the Grid middleware and the GMPLS Control Plane. “Interoperability” refers to the possibility to create advance reservations from the Grid middleware starting in one domain and ending in the same domain or in other domains. It is important to note that implementation details are beyond the objectives of this document. However one of the aims of this document is to identify architectures and interfaces that provide interoperability between different NRPSs. Two architectures will be discussed: the so called Peer to Peer architecture and the Client-Server architecture. In addition, a distributed approach for Phase 2 will be also presented.

1.3.1 Peer to Peer architecture

As can be seen in **Figure 1.2**, in the Peer to Peer architecture every single NRPS can communicate with any other NRPS using a software module called NRPS Adapter. At the same time, an additional NRPS interface will be created in order to allow the different NRPSs to interact with the Network Service Plane. When a domain (i.e. an NRPS domain) wants to create a connection through other domains, some signalling must be sent between the NRPS which starts the connection and its NRPS Adapter, and between this NRPS Adapter and the NRPS Adapters of other domains involved in the connection. This way the service request is propagated along a chain of domains until the service endpoint is reached.

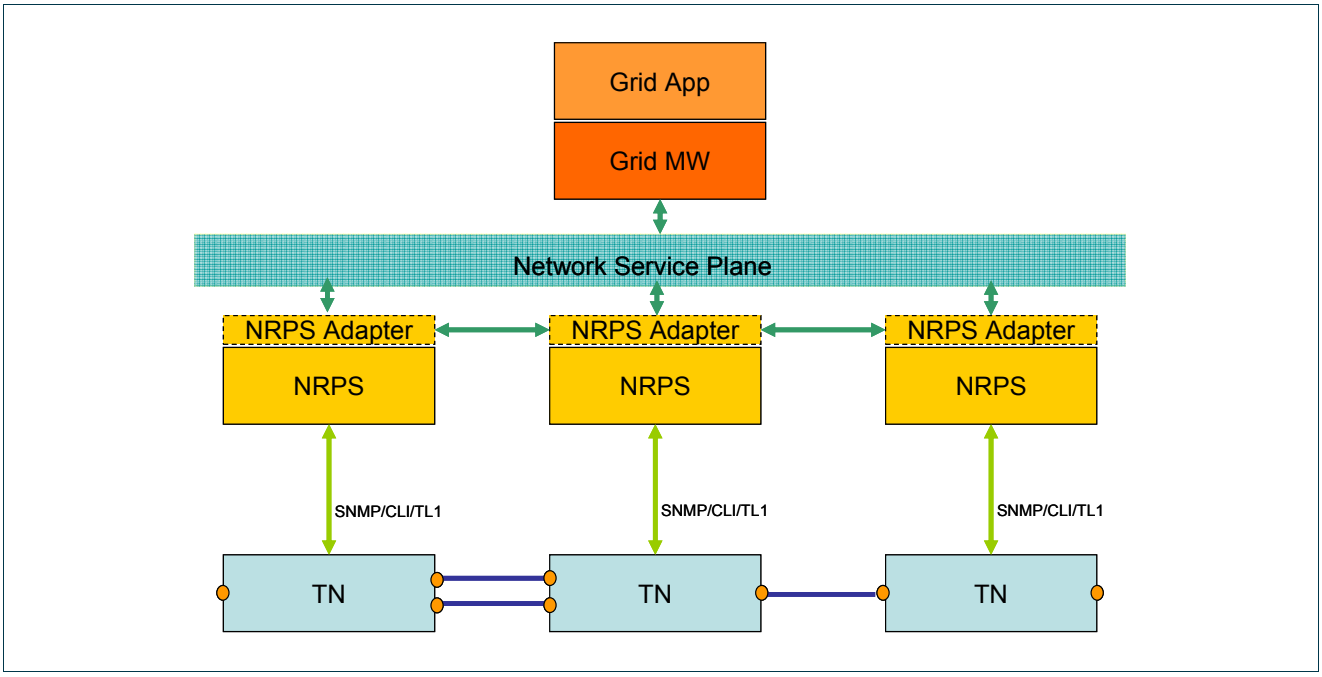

Figure 1.2: Peer to Peer architecture

One of the benefits of this architecture is that the intelligence of the system remains distributed increasing the fault tolerance of the system. On the other hand, this architecture is more complex than the Client-Server architecture due to the need to keep the different databases of all the NRPS Adapters aligned. This is required due to the fact that all NRPS Adapters need to know the topology of the complete network that interconnects the different NRPS Domains, in order to select the next domain to contact to create an end-to-end path. This kind of architecture has already been proposed by GÉANT2 JRA3. In particular, looking at the Interdomain Manager and Domain Manager, many commonalities between them and the NRPS Adapters and the NRPSs can be seen.

1.3.1.1 *NRPS Adapter*

In this architecture the NRPS Adapter is an NRPS dependant piece of software that implements the methods specified in the south bound interface of the Network Service Plane and the East/West interfaces to other NRPS Adapters. It is important to note that all NRPS Adapters need to have information about the topology of all the other domains, or at least sufficient information to be able to select the next domain to contact. Under this scheme the NRPS Adapters also need to manage the creation of connections in a distributed way. Consequently, the flow of information is critical under this architecture since any change in any domain must be propagated to the other domains. The NRPS adapter implements the following functionalities:

- Interchange topological information with other NRPS Adapters
- Select the next domain in the chain to create an end to end path
- Find paths between domains that meet the temporal requirements of advance reservations
- Participates in commit processes between NRPS Adapters participating in the creation of end to end paths.
- Sends commands to the NRPS underneath to establish intra-domain connections to serve end to end advance reservations.

The interface offered by the NRPS Adapter to the NSP should implement the methods listed in Section 2.2.1.1 (Reservation management). The NRPS Adapter will also have to present another interface that implements all link related methods described in Section 2.2.1.3(Topology management) in order to have information regarding the links connecting the domain controlled by this NRPS Adapter to other domains.

1.3.2 Client-Server architecture

In the Client-Server architecture all the interdomain logics are implemented in a module located in the Network Service Plane called the NRPS Broker. When a user or an application wants to create a connection that crosses more than one domain, it sends a command indicating the endpoints to the NRPS Broker and after that the NRPS Broker will trigger the required connections in the different domains.

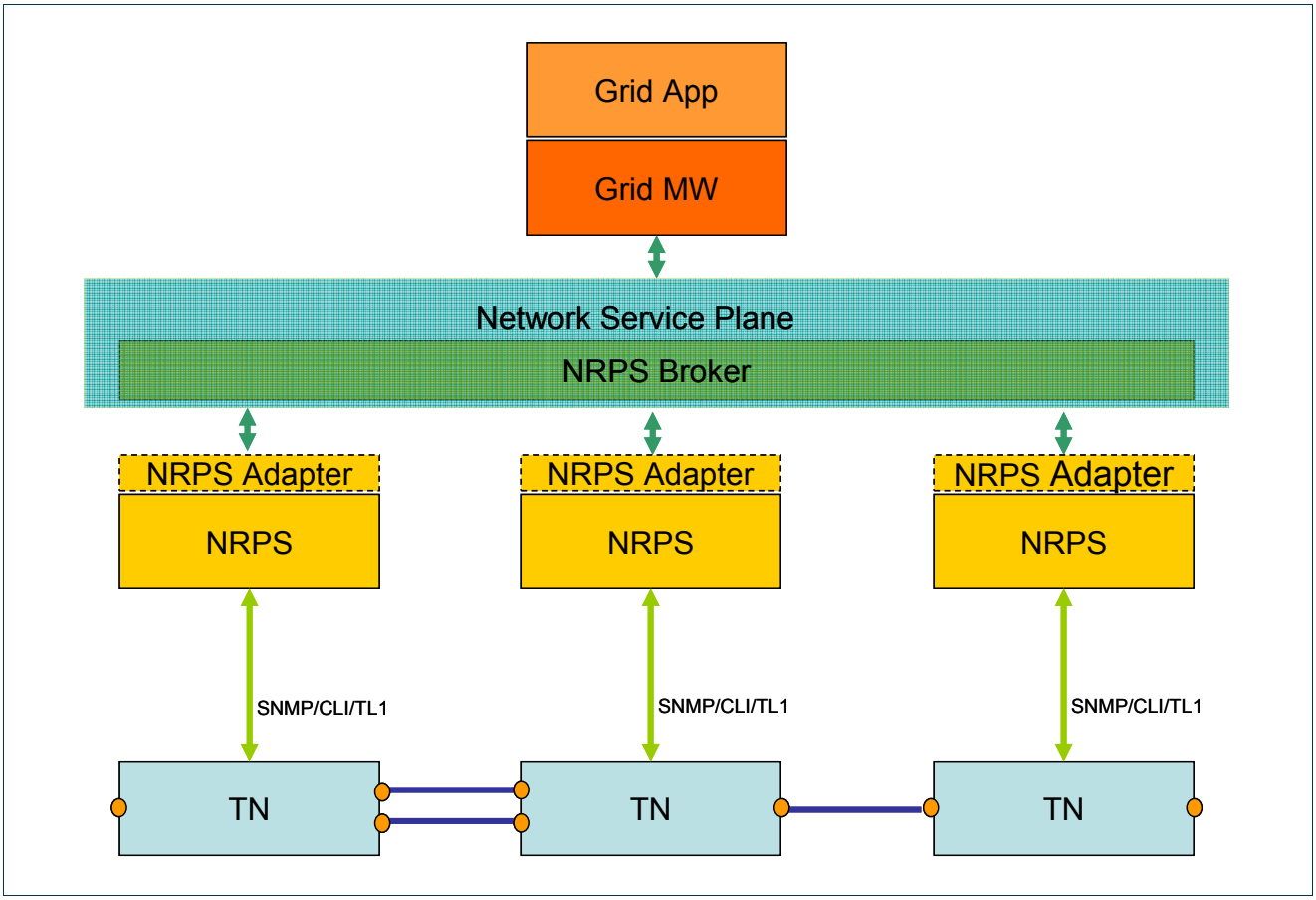

Figure 1.3: Client-Server architecture

1.3.2.1 *The NRPS Broker*

In this architecture the NRPS Broker is a block of the Network Service Plane responsible for the interconnection and interoperability of the different domains. The NRPS Broker performs the following actions:

- Gather information about the links connecting different domains.
- Find a route matching the bandwidth and latency requirements of the Reservation request. This will be done in two steps: Phase 1 will take into account only the bandwidth and Phase 2 will consider the delay as well.
- Trigger connections in the different domains in order to create end to end connections.
- Coordinate the work between all NRPS Adapters participating in the creation of end to end paths by using a transaction system.
- Realize the communication between the higher layers and the NRPSs and between pairs of NRPSs.

- Allow the communication of the NRPSs with the higher layers.

The NRPS Broker will receive requests from higher layers through other modules of the NSP and will forward them to the NRPSs, with the required adaptations in each case. When the proper actions are executed in the NRPS, it will return the result to the NSP through the NRPS Broker

1.3.2.2 *NRPS Adapter*

In this architecture the NRPS Adapter is a piece of software which is NRPS dependant that implements a common interface specified in the south bound interface of the NRPS Broker. This way, if a new NRPS is to be included in the system, only an adapter mapping the NRPS specific functionalities to the functionalities specified in the interface needs to be coded. However, if there are features specified in the interface that the NRPS does not support, the NRPS Adapter will have to indicate it. The NRPS Adapter will implement the interface with the operations explained in Section 3.4.

1.3.2.3 *A Distributed Approach*

Although this architecture scales in the same way as the Peer to Peer architecture from the topological database point of view, it is less fault tolerant since the NSP represents a critical point of failure. In order to overcome this problem and improve the interoperability with other projects, a distributed structure composed of a number of NSPs has been proposed to be studied in Phase 2.

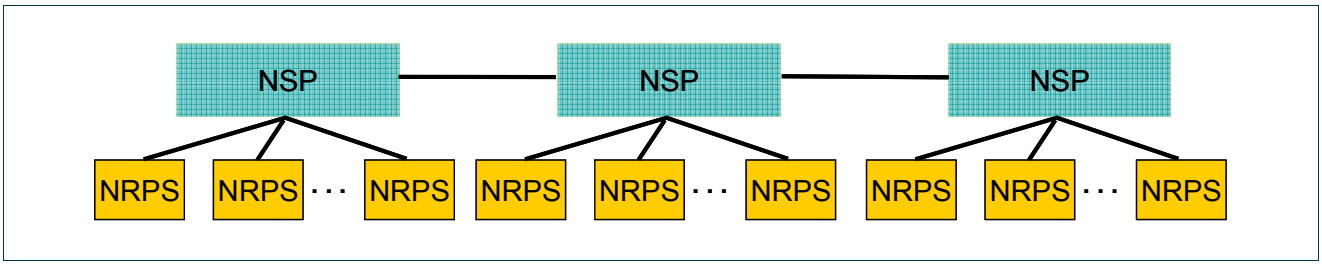

Figure 1.4: Possible distributed NSP architecture

Using this scheme an NSP would be able to interact with other NSPs to distribute and share its information. When creating a reservation, each one of the NSPs would know where are the resources that participate in the connection because each one of them would keep information not only of its own resources but also of other NSPs.

In this case the failure of a NSP will only make fail the connections involving the NRPSs managed by the staled NSP, improving the fault tolerance of this architecture and also its scalability. Additionally this architecture can make easier the interoperability with other projects, because it provides higher flexibility and scalability.

This architecture will only be taken into account in Phase 2. Since a first prototype of the project must be delivered by month 12, it has been considered that a complex distributed architecture can not be implemented by that time. Therefore, during Phase 1 the single NSP client-server architecture will be used. Afterwards, in Phase 2, the possibility to modify the NSP providing all this features will be studied.

1.3.3 Commonalities with GÉANT2 JRA3

In the previous sections two architectures have been proposed to provide interoperability between different NRPSs. In this section some of the commonalities with the architectures previously described and the work carried out inside GÉANT2 JRA3 will be identified.

As in GÉANT2 JRA3, to achieve the main objectives of the Phosphorus project it is needed to allow the creation of end-to-end paths that require the concatenation of various edge-to-edge paths in each domain through the stitching of different transport technologies. In GÉANT2 JRA3 this is achieved using the Interdomain Manager (IDM). The IDM is responsible for processing each incoming service request and for propagating the accepted request either locally to the Domain Manager (DM) or to another IDM in a neighbour domain. The service request is propagated along a chain of domains until the service endpoint is reached.

The main function of the DM is to set up Bandwidth on Demand (BoD) instances within the domain. As can be seen there are a lot of similarities between the Domain Manager and an NRPS and between the Interdomain Manager and the NRPS Adapter described in Section 1.3.1.1. Regarding the architecture used in GÉANT2 JRA3, on the one hand uses a Client-Server architecture between the IDM and the DM but on the other hand it uses a Peer to Peer architecture between Interdomain Managers in order to establish connections spawning more than one domain.

If the functionalities of the building blocks of the two architectures proposed in this document are compared with the functionalities of the DM and the IDM of GÉANT2 JRA3 it can be realized that the functionalities of the IDM can be mapped to the building blocks proposed in this document. The IDM functionalities can be mapped into the functionalities of the NSP, while the functionalities of the DM can be directly mapped to the NRPS functionalities. Taking into account the points mentioned above it can be said that there are a lot of similarities between the Peer to Peer architecture described in this document and the solution proposed in GÉANT2 JRA3.

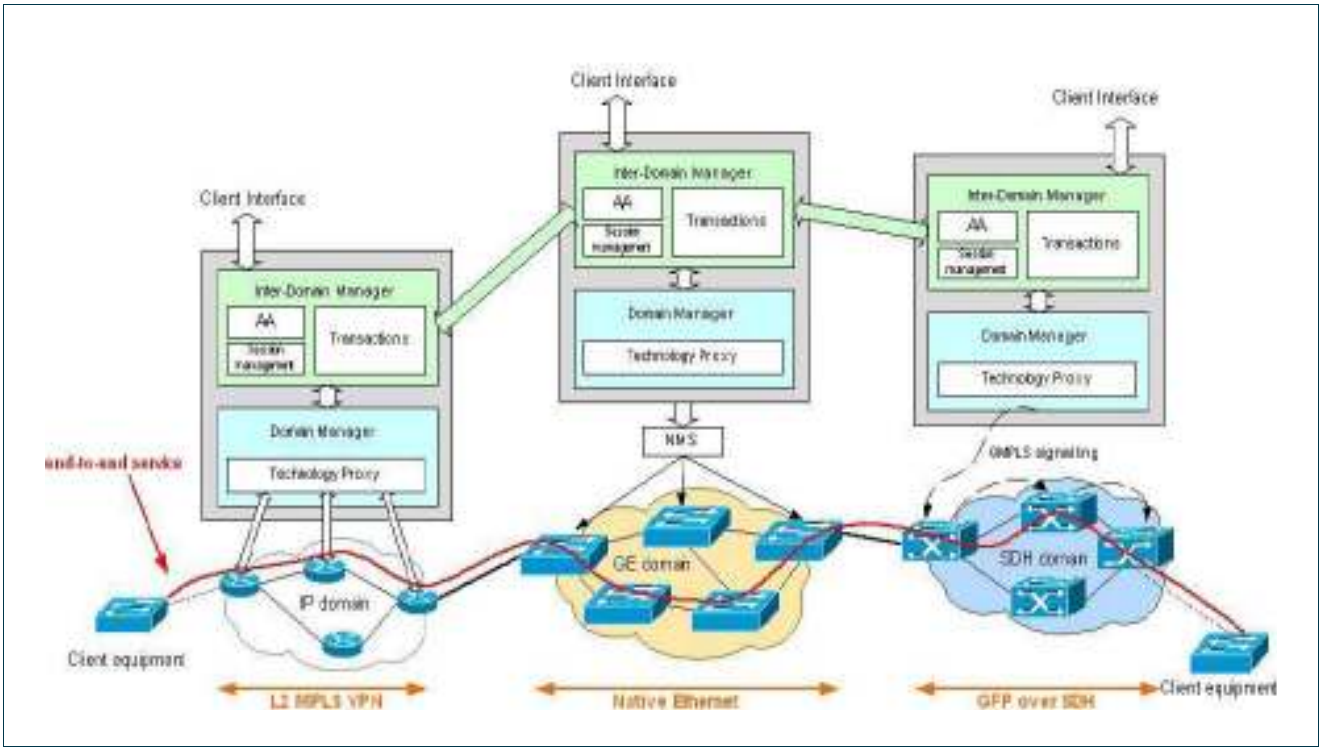

Figure 1.5: GÉANT2 JRA3 multi-domain BoD service

1.3.4 Architecture design principles

One of the main objectives of WP1 is to create an environment where any NRPS deployed in Europe would be able to interact with any other NRPS. Therefore the architecture selected should be the one that will allow the development to focus the work on inter-NRPS issues and enable the creation of a prototype by M12 in order to achieve Milestone 1.1. However, as described in the project description, the developments done in GÉANT2 JRA3 have to be taken into account in order to establish a synergy that will result in a higher quality project. Because of this GÉANT2 JRA3 representatives have been contacted to understand what kind of cooperation can be established or even if some code could be shared.

When looking at the Peer to Peer architecture it is easy to see many commonalities with GÉANT2 JRA3 that would make think that this option could save work and allow the work package to deliver a more complete prototype by M12. However after contacting GÉANT2 JRA3 representatives and evaluating the current status of GÉANT2 JRA3 developments it seems that it is not a valid assumption. The version of the prototype available during the creation of this deliverable has a very limited implementation of the Domain Manager (actually it is a dummy block), and all the routing functionalities (inter and intra domain) are implemented using static XML files or open source routing software. In addition the specification of the DM is not completed since the goal of the first prototype of GÉANT2 JRA3 is to demonstrate IDM functionalities and inter IDM communication.

Unfortunately, from our point of view, taking into account the current stage of GÉANT2 JRA3 developments and that WP1's first prototype must be ready by M12 the only thing that can be done at this point is to make our interfaces compatible. This way in order to create a synergy a Client-Server architecture has been chosen since it allows to focus the work on the interfaces and not on the signalling between domains (where GÉANT2 JRA3 has allocated more effort).

1.3.5 Phosphorus WP1 architecture

During Phase 1 it is planned to work on the Client-Server architecture. However during Phase 2, it will be studied the possibility to modify the NSP to allow the interoperation with the IDM (and other NSPs). Another possibility for the Phase 2 could be the interoperation with other projects like G-Lambda [G-Lambda] and EnLIGHTened [EnLIGHTened].

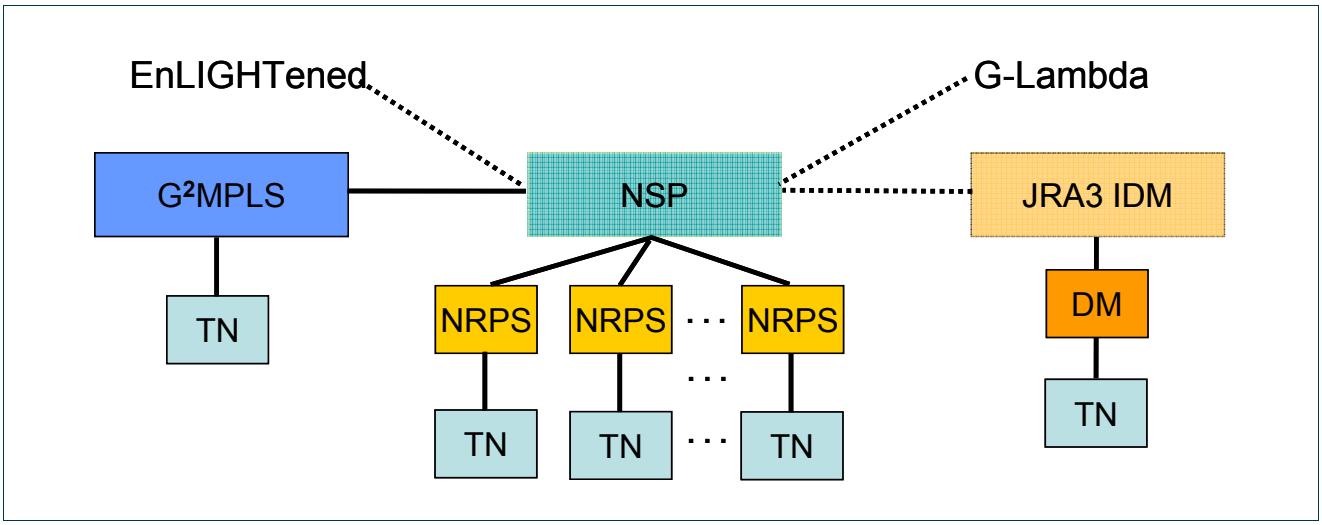

Figure 1.6: Interconnection possibilities for Phase 2 distributed architecture

1.4 AAA considerations

The basic idea behind AAA is to define a generic mechanism that allows regulating which users can use certain resources at what cost. In short, AAA is about:

- Authentication: Who are you?
- Authorization: What are you allowed to do?
- Accounting: How much have you used?

1.4.1 AAA sequences

Based on the user's identity and some policy that is stored in the Policy database a decision is made on what resources the user is allowed to use. In RFC2904, three basic authorization sequences are defined.

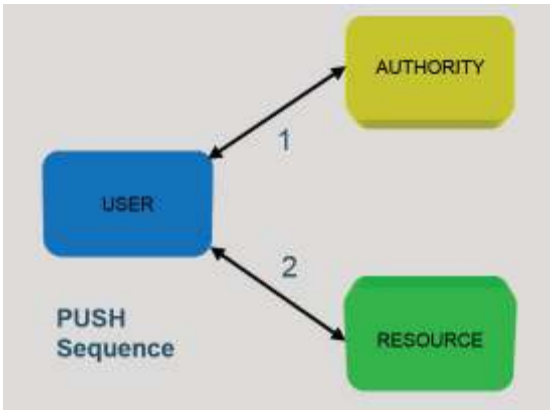

Figure 1.7: The PUSH sequence in which the user contacts the authority and pushes the decision of the authority to the resource

According to **Figure 1.7**, in this sequence the user first contacts the authority to ask permission to use a resource, in the case of a positive decision, the user forwards this decision to the resource.

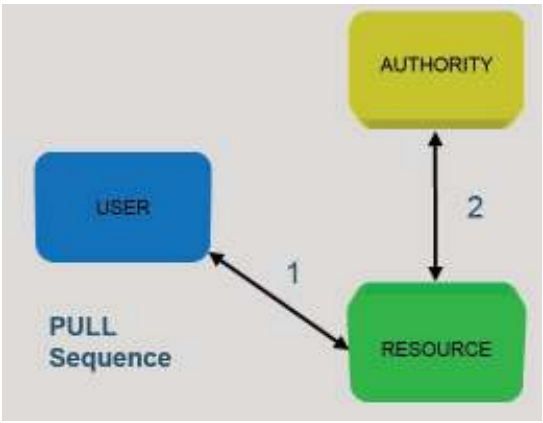

Figure 1.8: The PULL sequence in which the user contacts the resource and the resource pulls the authority for the decision

Figure 1.8 shows the PULL sequence. According to this sequence the users request first goes to the resource. Then, the resource contacts the authority to verify if the user has enough rights to use the resource.

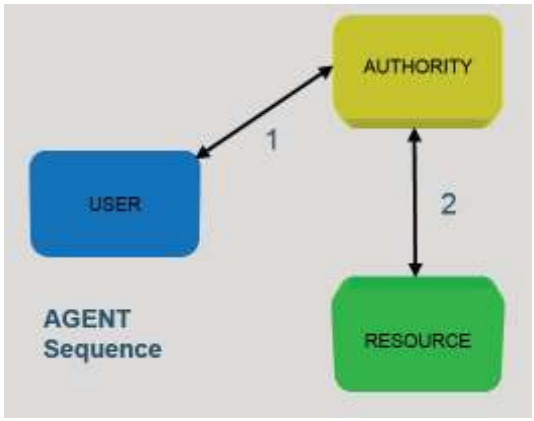

Figure 1.9: The AGENT sequence in which the user contacts the authority and the authority forwards the request with the decision to the resource

Figure 1.9 shows the AGENT sequence. According to this sequence the users request first goes to the authority. In the case of a positive decision the request is forwarded to the resource by the authority.

Apart from the three sequences described in RFC2904, a new sequence is currently being developed.

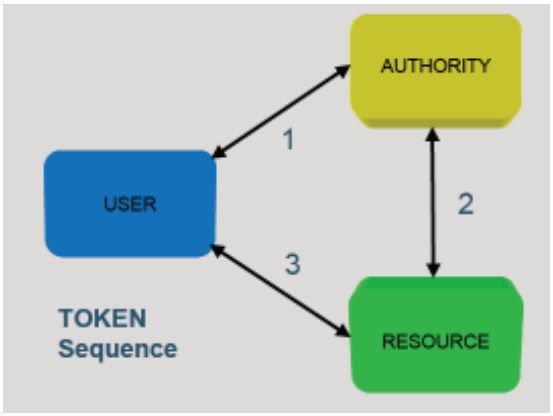

Figure 1.10: The TOKEN sequence in which the user contacts the authority and the authority sends back a key to generate tokens to the user, the decision is also forwarded to the resource with a key to validate the tokens that are generated by the user

Figure 1.10 shows the TOKEN sequence. According to this one, the users request first goes to the authority. In the case of a positive decision the authority sends back a key and transaction ID, and sends a key to the resource. After that, the user can use the key to generate tokens that enforce the use of the resource. The resource uses the other key to validate the tokens that are generated by the user. It is possible to have one single AAA server for the project, but it is also possible to use local AAA servers and have separate keys for all domains involved. Since it is assumed that each domain implements its own AAA server and the validation of the tokens is performed by the service itself, the NSP only needs to forward the key that is needed to generate tokens back to the user. Only one key is send back to the user. If different domains use different keys, new tokens should be generated by the domains that use the TOKEN sequence. In this case keys to generate TOKENS need to be distributed to all domains that use the TOKEN model. This can either be done by the AAA server directly or via the NRPS adapter and NRPS broker.

1.4.2 Policies

The authority bases its decisions on a policy that is stored in some policy database. If access to the policy database is uniquely restricted to the authority, only after each user request it is known if a specific user has rights to use a specific resource. For the Phosphorus project it is assumed that each domain has its own policy database.

1.4.3 AAA objects

Three AAA objects are defined in our system.

1.4.3.1 *The AAA user object*

This object contains the username, user's home organization and secret where:

- USER_ID = the identity by which the user is known to the home organization
- USER_ORG = a pointer to a service or server that can verify the users identity
- SECRET = the users proof of identity (like password of certificate)

As the user object gives access to services it should always be forwarded in a secure way, i.e. over a secure channel between two trusted services like the NRPS and the NRPS adapter or as a certificate send with the service request.

1.4.3.2 *The AAA service object*

This object contains the owner of a service where:

- OWNER = (a pointer to) the administrative owner of a service.

- SEQUENCE = the AAA sequence that is applied for this service.

The service object should preferably be public information. The administrative owner is the authority that can decide over the specific service. For different services of a single domain the administrative owners and sequences may be different. This may especially hold for resources that interconnect two domains. The AAA service object should always be included with every service that is published even if a service is freely available to all users.

1.4.3.3 *The AAA transaction object*

This object contains the owner of a service where:

- KEY = a key that gives access to a service.
- ID = a transaction ID identifying the AAA request.

The transaction object is passed to the Network Service Plane by the AAA server; the object is forwarded to the user and to the NRPS.

1.4.4 AAA scenarios

This section will describe the possible AAA scenarios for the client-server architecture that is chosen for the Phosphorus project. As in this architecture each NRPS talks to any other NRPS only through the NRPS Broker, the NRPS broker should be able to make all AAA decisions or AAA information should flow between the NPRS and the NPRS adapter and between the NRPS adapter and the NRPS Broker. As local AAA servers are assumed for each domain, the latter will be the case.

The PULL sequence requires the user to send the request directly to the service and the PUSH, AGENT and TOKEN model require the user to send the request to the authority. As the Network Service Plane is involved in all inter-domain requests, all requests for all four sequences will pass through the NSP. In the client server architecture, the NPRS adapter maps local to global users and forwards the request to the NRPS Broker. The NRPS Broker checks the user's identity, decomposes the request and forwards requests to the NRPS adapters of the domains involved. Each NRPS adapter maps global to local users and either contacts the AAA server or forwards the request to the NRPS that in its turn will contact the AAA server.

The decision of the AAA server is returned via the NRPS adapter and the NRPS broker to the Network Service Plane and then forwarded to the user. For the PULL, PUSH and AGENT sequence, only the right to use specific resources is validated. It is assumed that all traffic that enters a domain at a given port is authorized to use the network.

In case of the TOKEN sequence not only the reservation and activation of resources, but also the usage can be enforced. For this sequence, the message send back to the user includes a key by which the user can generate tokens to sign the traffic. These tokens are used to enforce the usage of the resource. A requirement for the

TOKEN model is that traffic with tokens can be sent transparently through domains that do not use the token model. For the Phosphorus project this means that traffic with tokens must be valid Ethernet traffic.

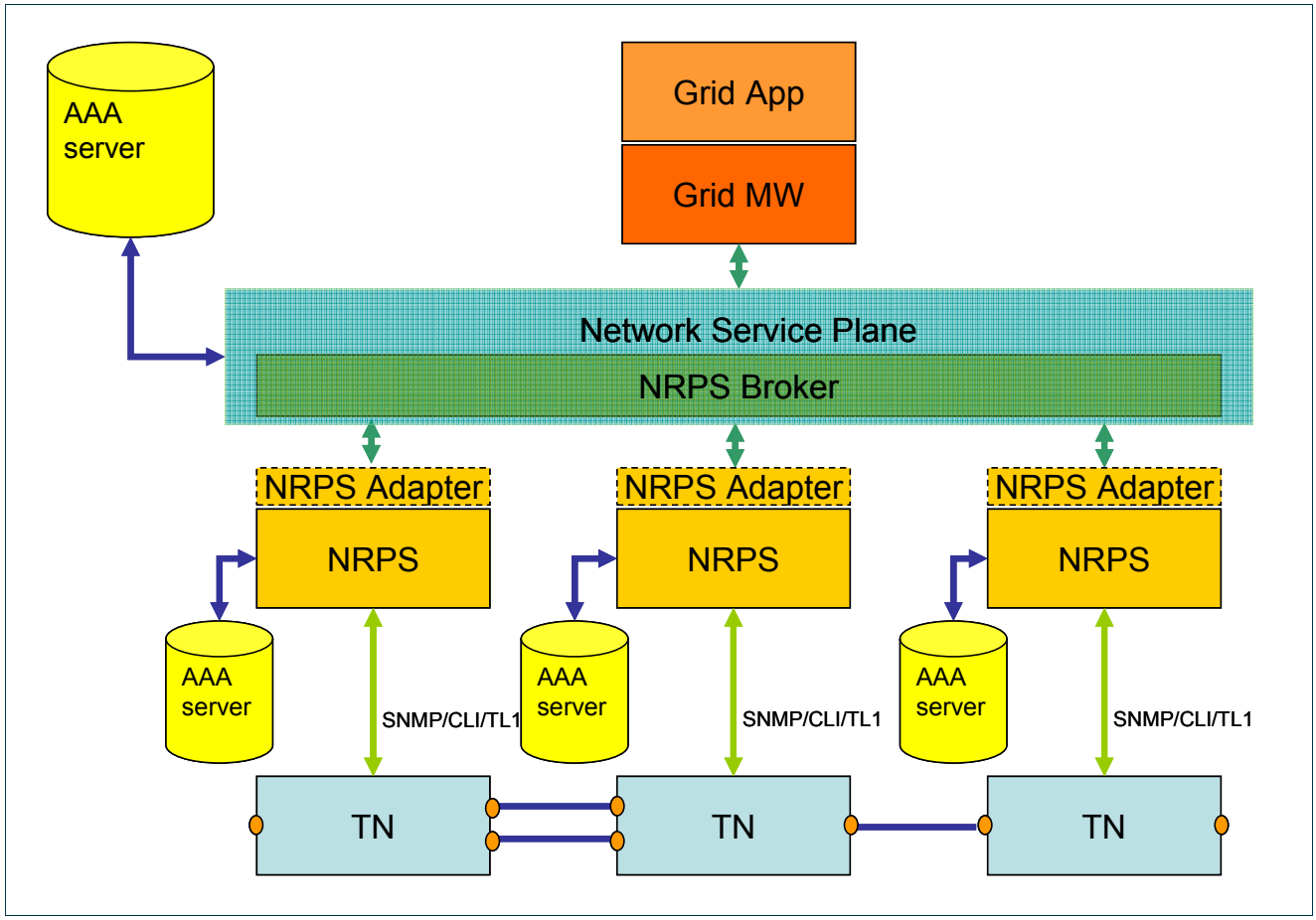

Figure 1.11: AAA servers interaction within NSP and NRPSs in the client server architecture as chosen for the Phosphorus project

1.4.5 AAA Architecture

As presented in previous sections, the client-server architecture is going to be implemented. There will be a global user database that is used by the Network Service Plane. Users will be identified by username and password. Domain administrators will need to maintain the central user database manually. Each domain administrator needs to provide the mapping from global users to local users. The reason for choosing a centralised user database is that not all NRPS's have the ability to handle "remote user databases" and some NRPS's only distinguish between different groups of users. The NSP checks the user's credentials and forwards them via the NRPS broker to the NRPS adapter. It is assumed that each domain has its own AAA server, policy database and user database. The NRPS adapter, which needs to have access to the global user database, can validate the user's credentials again and maps global users to local users. This way the NRPS

does not need to be modified to handle the central user database. Regarding the AAA sequences, all four AAA sequences will be supported. The PULL sequence is used in GÉANT2 JRA3 and it will be useful to solve interoperability issues. On the other hand, WP1 and WP4 will implement an agent for each NRPS that can handle GMPLS signalling. With this agent, it is possible to use the TOKEN sequence.

2 Scenarios and use cases

2.1 WP 1 Scenarios

During Phase 1 WP1 will develop the NSP that allows existing NRPSs to create end-to-end connections, whereas during Phase 2 the system will be adapted to enable the use of G^{2}MPLS in combination with existing NRPSs. Therefore a different set of scenarios for each phase are going to be used. These scenarios will be the platform to test the developments carried out within this work package as well as to validate the interoperability with other work package developments. During phase 1 WP1 will also develop a web service interface to a standard GMPLS protocol stack, which will be provided by WP2.

2.1.1 Phase 1 scenarios

During this phase three scenarios are going to be used. These scenarios will showcase the basic interoperability between NRPSs, the NSP and the standard GMPLS Control Plane. In all scenarios it is possible to request advance reservations with endpoints located in the same or different domains. An Advance Reservation Request can be invoked through the Network Service Plane or directly through an NRPS Adapter.

The scenarios proposed are the following:

- Scenario I: This scenario allows WP1 to demonstrate interoperability between different NRPSs and the Middleware. In this scenario no GMPLS control plane is used, and all the communications with the network equipment is done at the NRPSs.

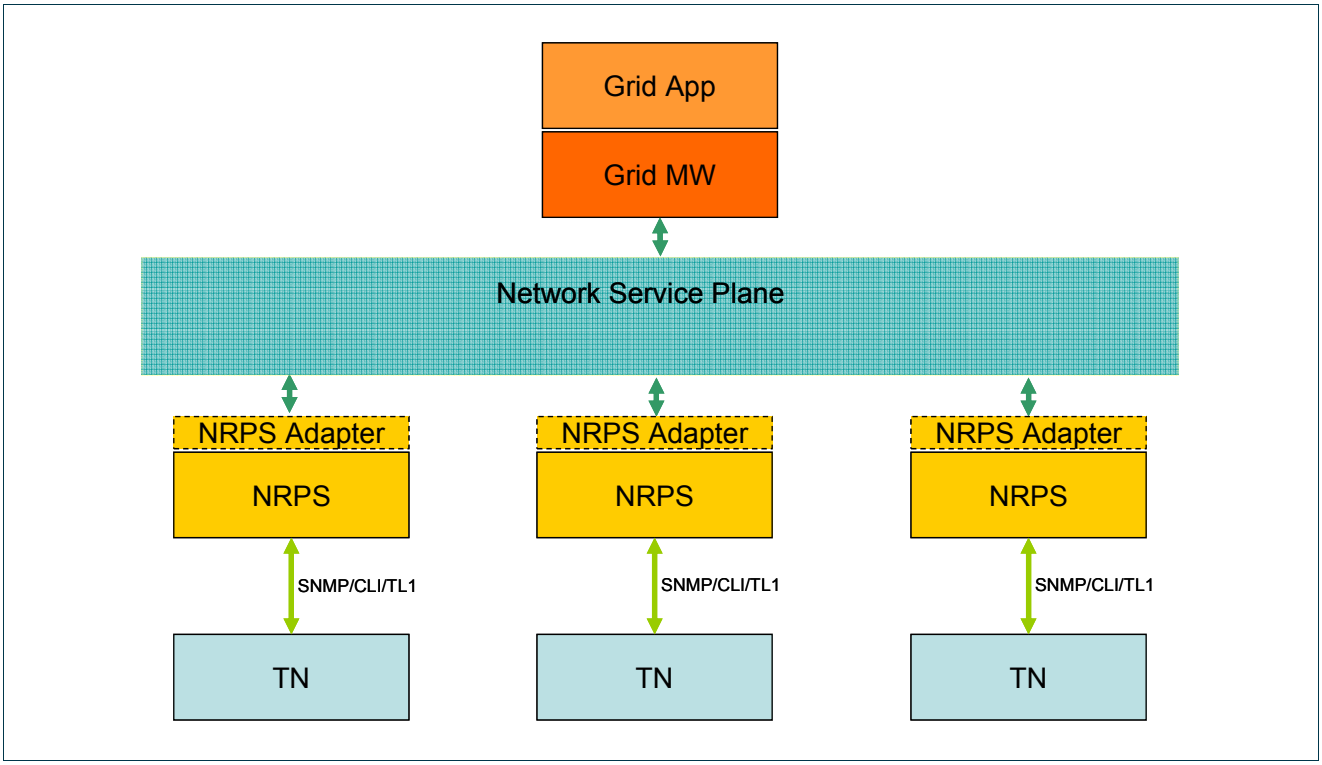

Figure 2.1: Scenario I

- Scenario II: This scenario is proposed as an interim solution for those domains that do not have an NRPS, but rely on GMPLS to do connection provisioning and management. This way interoperability between NRPSs, Grid Middleware and a standard GMPLS will be demonstrated. To achieve this interoperability an NRPS Adapter is defined on top of the GMPLS control plane. This will realise a "thin NRPS" that allows the NSP to communicate with the GMPLS domain through the same common interface it uses with the NRPSs. This "thin NRPS" will not support advance reservation. The effective interface between the NRPS Adapter and GMPLS we be the same as in scenario III, but with less functionality. As advance reservation is not supported, explicit routes for the connections are not required and the topology information is limited to the retrieval of endpoints

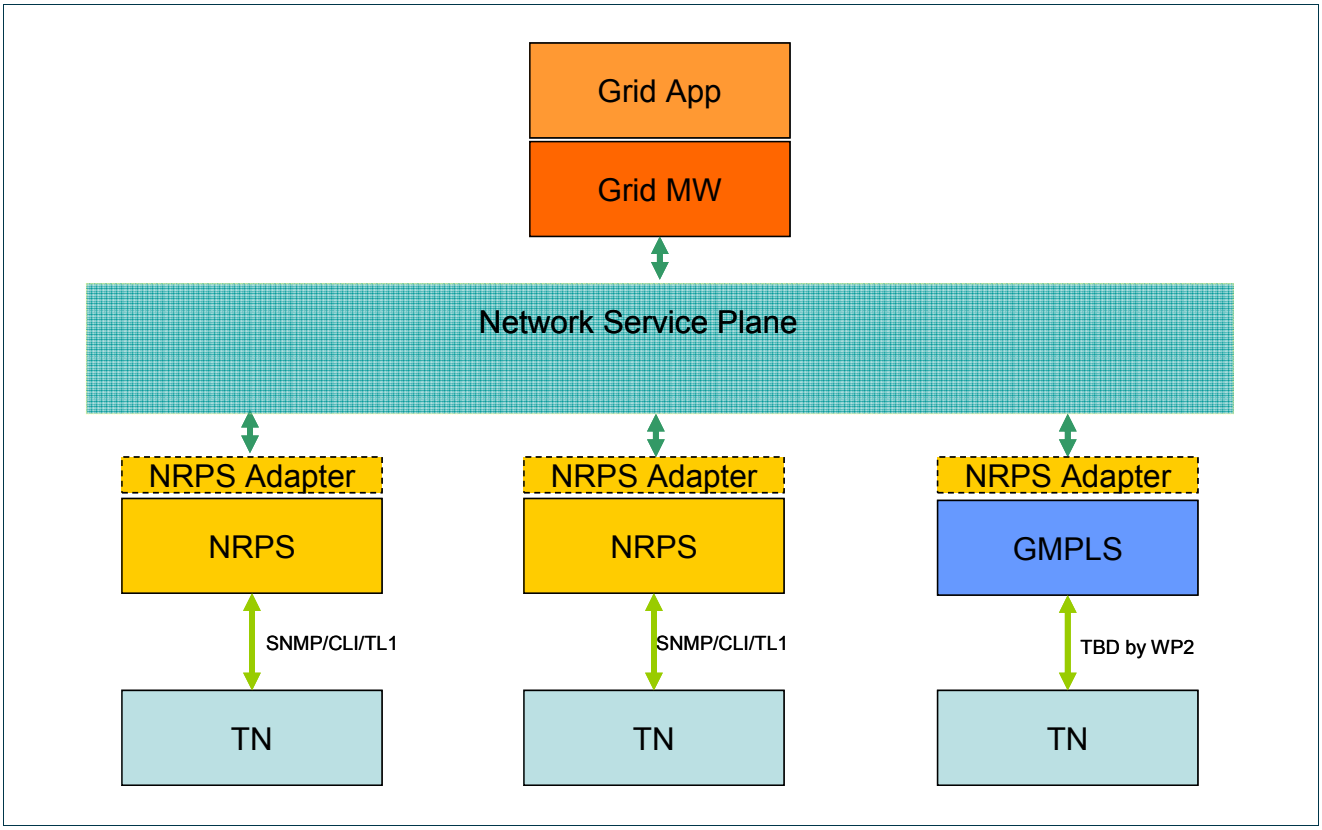

Figure 2.2: Scenario II

- Scenario III: This scenario will allow WP1 to demonstrate interoperability between different NRPSs, the Middleware and a standard GMPLS Control Plane being a client of an NRPS. The NRPS of a specific domain will ask the underlying GMPLS Domain for end to end connections starting and ending in the GMPLS domain. Since advance reservations can be implemented at the NRPS layer, they can be provided on a GMPLS controlled domain if an NRPS is created on top of GMPLS, and it is assumed that nobody else will set up connections in the GMPLS domain. This scheme allows a user or an application to manage either GMPLS enabled or non GMPLS enabled devices belonging to the same administrative domain. This interface provides the possibility to set up connections with explicit routes, and delivers full topology information (not only the endpoints) which is needed at the NRPS to provide guaranteed advance reservations, since the alignment between the NRPS advanced path computation & booking and the underlying GMPLS path computation must be guaranteed.

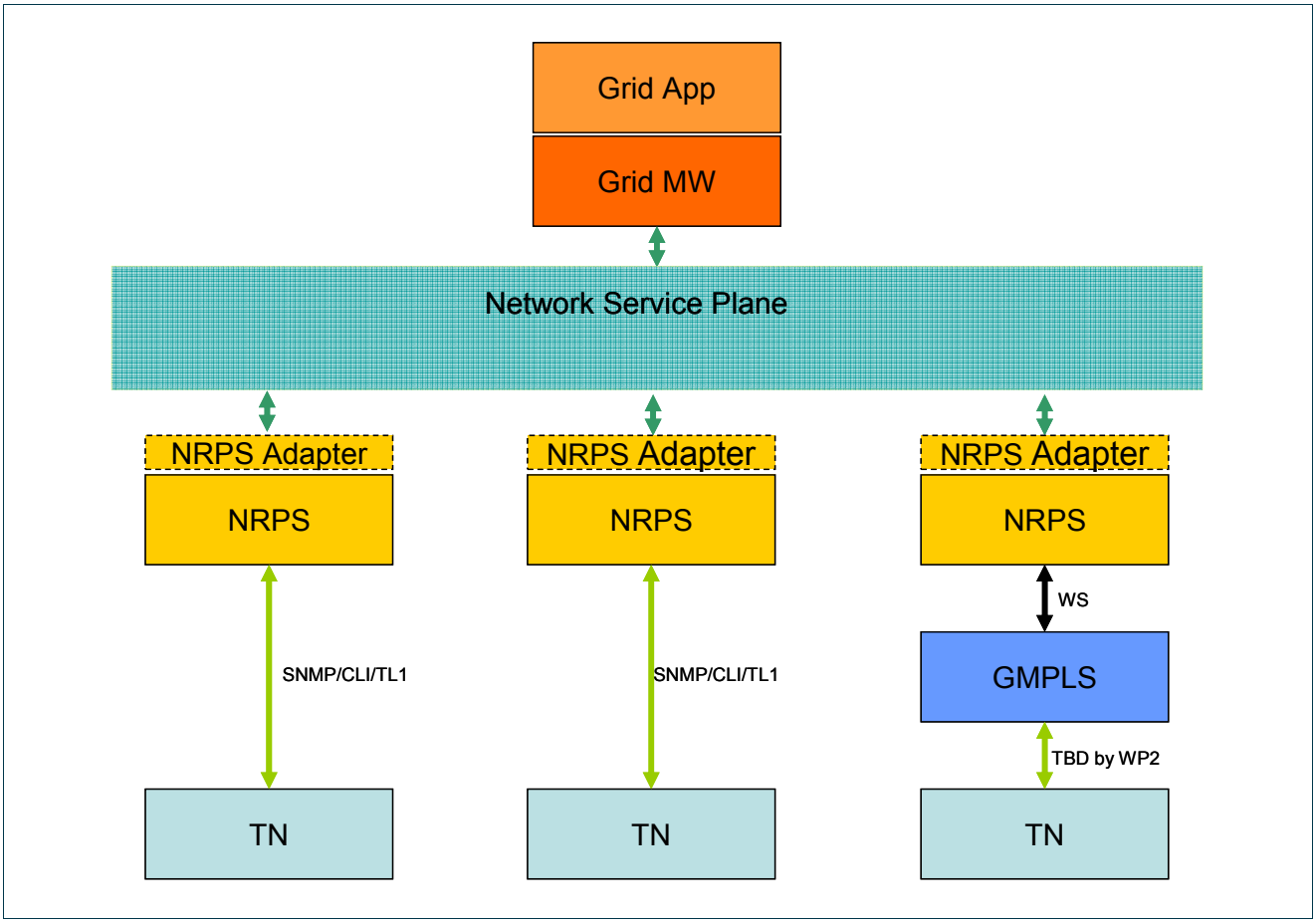

Figure 2.3: WP1 phase I, scenario III

2.1.2 Phase 2 scenarios

- Scenario I: During Phase 2, WP2 will develop an enhanced version of GMPLS with capabilities to control Grid resources. This new version of GMPLS is referred to as G^{2}MPLS (Grid enabled GMPLS). The figure below depicts a use case where a user or an application requests for a data transfer from the “Data source” to “Computational resources A”. This scenario will demonstrate the interoperability between NRPSs, the Grid Middleware and G^{2}MPLS.

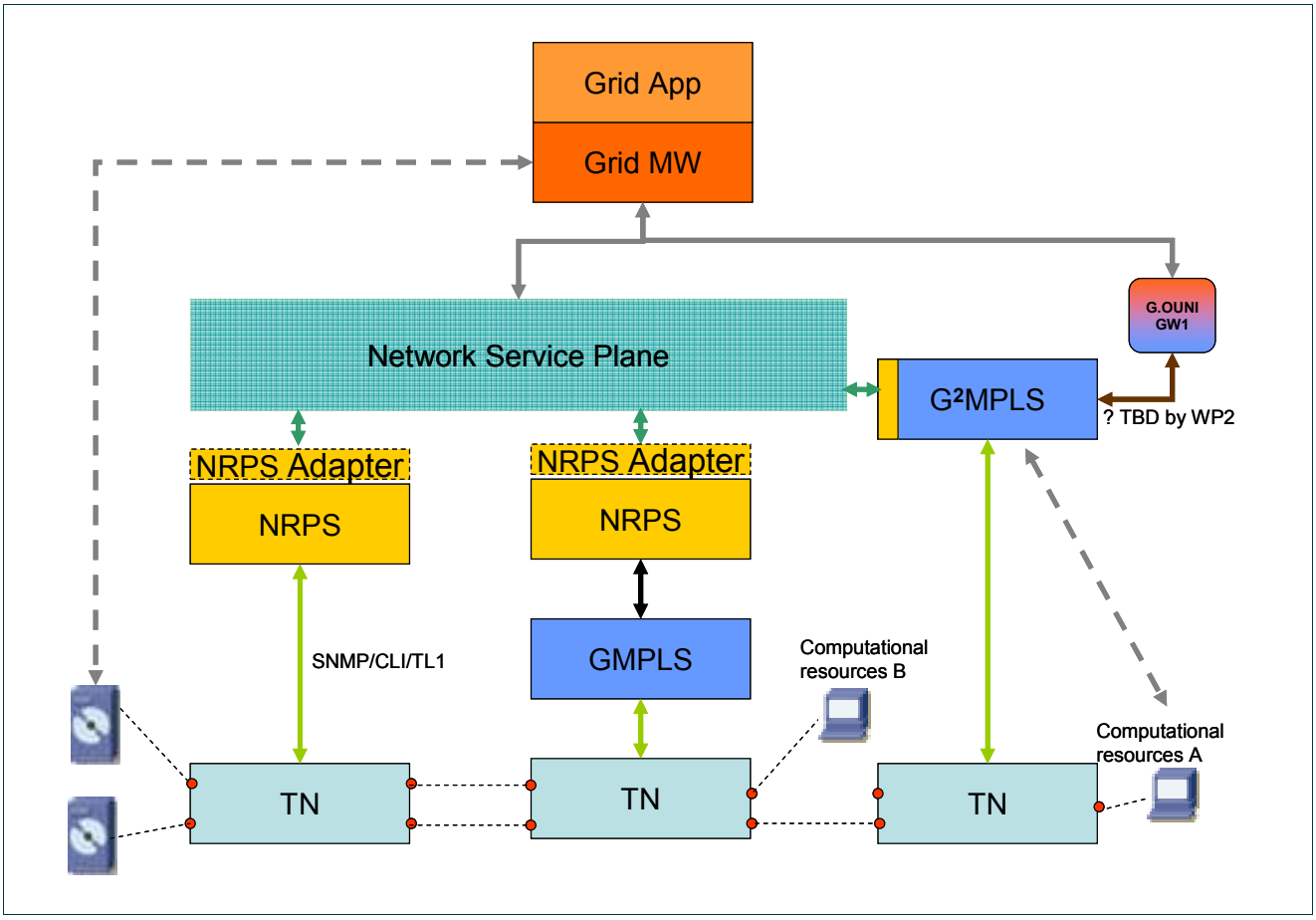

Figure 2.4: scenario I

Some variants to scenario I could be explored during Phase 2 of the project.

2.2 User interface requirements

This section will define the requirements of the interfaces and the use cases. The objective is to define the basic interaction between the users and the blocks that build the system. At this point, no reference to the architecture or specific blocks will be done yet, since this section is focused on clarifying the basic interactions between the software/human users and the system to be developed.

2.2.1 Interface requirements

In this section two different groups of interface functionalities between the NSP, the NRPS and the user are described. Reservation management is dedicated to the creation, deletion and querying of connections across

domains, whereas topology management is concerned with the functions required to enable the system to keep track of the resources it can call upon.

2.2.1.1 *Reservation management*

One of the main requirements of the project is reservation management. The user has to be able to create, delete and query advance reservations in the network through the system. Therefore a user will be able to access the NSP or the NRPS to request advance reservations.

An advance reservation is composed by a set of services and the services are composed by one or more connections. To define these items the following information is needed:

- Connection: A connection is defined by the following parameters:
    - Two or more endpoints: Exactly one of these endpoints is tagged as source endpoint. This tag is important for unidirectional and bidirectional trees.
    - Bandwidth/delay constraints: Minimum and maximum bandwidth (which may be equal), maximum latency.
    - Directionality (in the case of exactly two endpoints, the unidirectional tree is simply a unidirectional connection and both the bidirectional tree and the full mesh can be reduced to a bidirectional connection):
        - Unidirectional tree: There are unidirectional connections from the “source endpoint” to each of the “target endpoints”. Each of these connections fulfils the given bandwidth/delay constraints.
        - Bidirectional tree: The “source endpoint” has a bidirectional connection fulfilling the bandwidth/delay constraints in each direction to every “target endpoint”.
        - Full mesh: Every endpoint is connected to every other endpoint with the given bandwidth/delay constraints.

- Service: While “connections” are of topological nature, “services” add time constraints. A service is defined by a service type and consists of one or more connections. All connections grouped in a service are characterised by exactly the same time constraints.
    - Service ID: This is a unique identifier for a service within a set of services aggregated to a single reservation.
    - Service Type: In the current project phase, this field can only have the value “connections”. It was considered useful however to have a type field that can be used to include other services in the future, e.g. VPLS.
    - Connection Data: Since for now, a service is restricted to connections as defined above, this field consists of one or more connections.
    - Reservation Type: Reservations can be fixed, deferrable or malleable. A fixed reservation has a fixed starting time and a fixed duration. A deferrable reservation also has a fixed duration, but a variable starting time. A malleable reservation finally has a variable starting time, a variable duration

and, as a result, variable bandwidth. These three reservation types are visualized in the following figure.

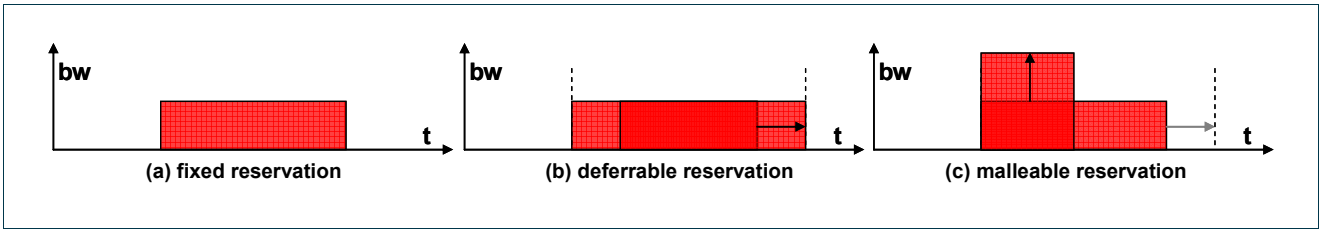

Figure 2.5: Types of reservations

- Time constraints (note that bandwidth/delay constraints are given in the connection data).
  - For fixed reservations: Start time and duration.
  - For deferrable reservations: Earliest start time, duration, and deadline.
  - For malleable reservations: Earliest start time, amount of data to be transferred, and deadline (time by which data completely has to be transferred). Higher layers should not expect that a fixed bandwidth is reserved for the whole duration of the connection. Instead, the bandwidth assignment can be variable over time.

The functions that the user can invoke to deal with reservations are the following:

- Availability Request: Checks the availability of the resources required for connections with endpoints in the same domain or in different domains. The resources are not reserved; instead the Network Service Plane returns a response indicating whether the reservation would be feasible or not. In case it is not feasible, additional information on alternative starting times should be returned in the response. This functionality is needed by the MetaScheduler for co-allocation of grid and network resources. Since the availability request does not block any resources, a separate reservation request has to be sent to the Network Service Plane in case resources that are available shall actually be reserved.

- Reservation Request: The reservation request reserves network resources for one or more services for the requesting entity. This does not mean that in case of success, the selected resources will automatically set up at “start time”; this method just tags some resources as reserved for the duration of a requested service. These resources are exclusively reserved between the specified start and end times. Reservations are maintained for the entire period of a service, even if an activation is never received. Also, a service is automatically terminated at end time if it is not cancelled. As an answer to a Reservation Request, the user will receive a Reservation Response message indicating if the reservation is feasible or not. Additionally to these “permanent” reservations, there is also the possibility to make “pre-reservations” that time out after a while unless they are confirmed in the meantime. This will be explained in more detail in the context of job management.

- Reservation Status: Queries one or more reservations already made in the network. This functionality can be split into two alternatives: retrieve an owned connection identified by a ReservationID and

retrieve all the owned connections. This falls under the umbrella of access control rules and will be defined in Phase 2 of the project. This action allows the user to check the status of an active reservation. The information returned by this function may include several aspects regarding the reservation and the underlying physical connection.

- Cancel Reservation: Cancel reservation request with endpoints in the same domain or in different domains. Deletes a reservation identified by its "reservationID", once this has been done all the resources that participate in the connection are released and tagged as available. This operation can be called by a user when he wants to cancel a started reservation for any reason or even before the reservation start time has arrived. It can be executed automatically by the NRPS as well when a reservation expires.

- Retrieve Endpoints: Retrieves the available endpoints of the network that are suitable for the user to make the connections.

- Bind request: The bind request is needed to provision layer 3 services. Since only the packets with specific IP source and destination addresses are to be mapped to the endpoint of a NRPS provisioned path, the local network must be informed about these addresses so that the equipment can be configured appropriately. This information may be available only shortly before the start of a service, e.g. in case of cluster nodes that are dynamically assigned for a computation.

2.2.1.2 *Job management*

Several independent reservations can be grouped to a single "job". This feature basically allows for a higher convenience when the Middleware plans complex workflows: Planning of workflow items depends on how exactly the constraints of preceding workflow items are set such that the required resources are available. While the workflow is not yet completely planned, these workflow items are only loosely blocked as "pre-reservations", their affiliation to the same job is indicated by a common job ID. Therefore, if the complete workflow was successfully planned, these pre-reservations can be modified to regular reservations with a single request to the Network Service Plane. In the same manner, the complete set of pre-reservations can be cancelled with a single request in case the workflow cannot be implemented. The following two functions are implemented for job management and will be available to the users through the NSP access point :

- Complete job: In case a workflow has successfully been planned, this function is used to convert all pre-reservations made in the context of this workflow to permanent reservations.

- Cancel job: In case planning of a workflow cannot be completed successfully, this function is called to cancel all pre-reservations made so far for this workflow

There are three status for jobs:

- Open job: this job is in process. The resources to be reserved are still considered as pre-reservations. This job has not either completed nor cancelled.

- Completed job: this job has successfully reserved all the resources that were requested. Hence, the pre-reservations have been converted to reservations.

- Cancelled job: the cancel job method has been invoked for this job and all the related pre-reservations have been consequently cancelled.

2.2.1.3 *Topology management*

The NSP has to know which resources are under the control of the system. Therefore the system must store, retrieve, modify and delete the resource-related information according to the topology of the controlled network domains and reservations. This will require the implementation of a persistence module and a data model to maintain the information about the topology and the reservations. This information will be useful as well to perform several functions (e.g. interdomain routing) required in the NSP. The definition of Link and Endpoint that is going to be used in this work package is the following:

- Link: A link represents a physical connection between two different domains. It is associated to the two endpoints connected physically. The information required for a link should include:

    - Identifier: This unique ID could be generated internally by the system
    - Name: Name provided by the user that creates the link object
    - Source endpoint: The source endpoint of the link
    - Destination endpoint: The destination endpoint of the link
    - Bandwidth: Bandwidth of the link, calculated from the lowest bandwidth of the two endpoints
    - Delay: Time delay experienced in the link.

- Endpoint: An endpoint is an edge port of a domain, it can be a port connected to another domain or a port presented to the client of the network.The endpoints that interconnect two domains will form part of a link. Otherwise, the ports will be selectable to be the source or destination ports of a reservation. The information required for an endpoint should include:

    - Identifier: This unique ID could be generated internally by the system
    - Name: Name provided by the user that creates the endpoint object
    - Domain: Domain where the endpoint is located
    - Bandwidth: Bandwidth of the port
    - IP address: IP address of the port
    - Type: Interdomain/client
    - Link: Link that the port belongs to (if any).

The information about the topology will be learned in different ways depending on the type of resource. For the endpoints managed by an NRPS, the NSP will discover them automatically by making requests to the NRPSs at particular moments (periodically or when an action related to endpoints is executed). For the links between domains and the domain management, there is no automatic learning mechanism since the network does not

have information about them. Therefore, the user will have to create domains and links in the NSP by introducing the required information, i.e. selecting the endpoints of the network that will form the link or specifying the information about the domain.

This way there is a need for some functions or methods to create/modify/delete domains and interdomain links in the NSP. These methods are only accessible by local administrator user profile which is accessing the NSP thoughout the topological interface. These methods are:

- Add Domain (or NRPS): Adds a domain controlled by an NRPS or not. After adding the domain, the user should also enter the information regarding the interdomain links related with this domain.

- Update Domain (or NRPS): Updates the information for a registered domain. This will be useful to update the information of a domain in case that it is changed.

- Delete Domain (or NRPS): Deletes a domain from the system. This method will be used when a domain will not be used any more in the system to create connections.

- Query Domains: It allows the user to retrieve information about a single domain or all the domains he is able to manage.

- Add Link: Adds links between domains to interconnect them. This operation will allow the administrator of a domain to logically connect two domains in the system in order to be able to establish connections between them using the ports given for the link which is being added.

- Update Link: Updates the information for a registered link. This function could be useful for example, when an endpoint that belongs to a link is physically disconnected from a port and connected to another one. Using this function the administrator could update the information of the endpoint that has been changed and change its associated port.

- Delete Link: Deletes a link between two domains. This method allows a user to delete a link from the system.

- Query Links: It allows the local administrator to retrieve information regarding the links between two domains or several domains.

2.2.2 Use cases for the user accessible methods

In this subsection the primary actions that the users will execute in the system are going to be described. These actions are the initiators of the processes that will be performed in the system in order to offer the required functionalities. These processes will be specified in sections 3.1.9 and 4. The user accessible methods can be divided in two groups, the reservation management and the topology management functions:

- Reservation management (Middleware user):

- Availability Request
- Reservation Request
- Cancel Reservation
- Reservation Status
- Bind Request
- Activation Request
- Complete Job
- Cancel Job

- Topology management (local administrator):
  - Retrieve Endpoints
  - Domain Management
  - Link Management
  - Endpoint Management

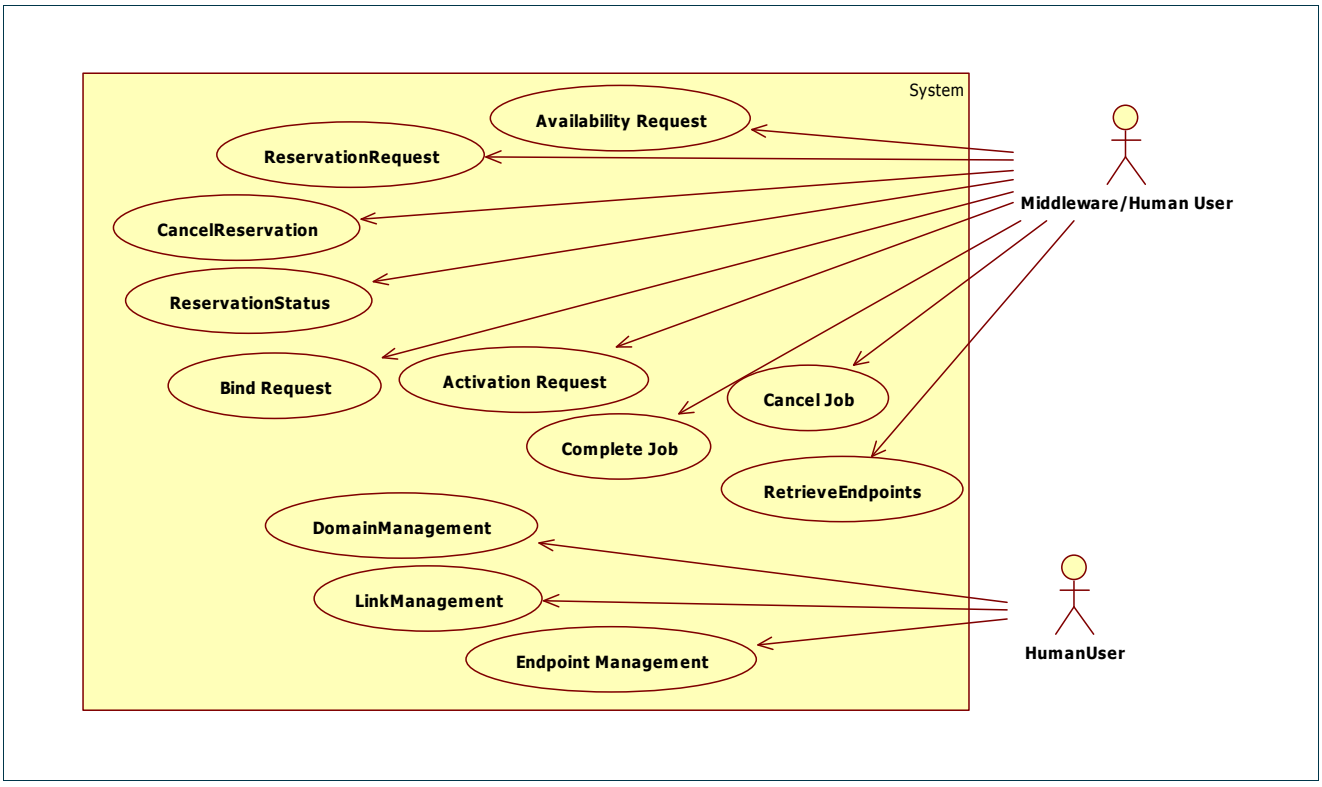

Figure 2.6: Use case diagram

2.2.2.1 *Availability request*

Goal	Checks for the availability of network resources
Actors	User (through the Middleware)
Preconditions	• The Endpoint Request has been executed (by the user or by the system) to know which are the possible connections' termination points • The user knows the parameters for the reservation, services and connections • The NRPSs are active
Basic flow	1. The user indicates all the parameters for the reservation he wants to check 2. The system checks in the database if the resources are available 3. The system contacts the NRPSs to check the availability if required 4. If the entire process has been successful, the system returns an OK message 5. The system gives the possibility to make the reservation if it is feasible
Alternative flow	2a. If there are not enough resources, return a message indicating it 4a. If some NRPS has not enough resources, return an error message
Postconditions	The system has checked if the resources of a reservation are available or not

Figure 2.7: Availability request

2.2.2.2 *Reservation request*

Goal	Make a reservation of network resources
Actors	User (through the Middleware)
Preconditions	• There are available resources to be included in the reservation • The NRPSs are active
Basic flow	1. The user indicates the resources that have to participate in the reservation (maybe after a previous availability request) 2. The user gives the time-related restrictions for the reservation 3. The system contacts the NRPSs and calls the appropriate reservation function 4. If there are no errors from the NRPSs, the system schedules the reservation and registers the related information in the database 5. The system activates the reservation when the start time arrives
Alternative flow	1a. If resources are not available, the system sends back an error 2a. If time-slots are not available, the system sends back an error 4a. If one or more NRPSs return an error message, the reservation is not scheduled and a message indicating it is returned
Postconditions	The system has scheduled a reservation that will be activated when the start time arrives

Figure 2.8: Reservation request

2.2.2.3 *Cancel reservation*

Goal	Cancels a reservation
Actors	User (through the Middleware)
Preconditions	• There is at least one active reservation • The user who requests the cancellation has administration privileges or is the owner of the reservation • The NRPSs are active
Basic flow	1. The user indicates the reservation to be cancelled 2. The system authenticates the user 3. The system contacts the NRPSs and calls the appropriate cancellation function 4. If there are no errors from the NRPSs, the system deletes the related information of the database
Alternative flow	2a. The user has no privilege to cancel the reservation, an error is returned
Postconditions	The system has cancelled the reservation and has deleted all the information of the system

Figure 2.9: Cancel reservation

2.2.2.4 *Reservation status*

Goal	Queries the status of a reservation
Actors	User (through the Middleware)
Preconditions	• There is at least one active reservation • The user is an administrator or the owner of the reservation to be queried
Basic flow	1. The user indicates the reservation that he wants to query 2. The system authenticates the user 3. The system retrieves the information from the database and presents it to the user
Alternative flow	2a. The user is not allowed to manage the queried reservation, the system returns an error
Postconditions	The system has obtained and forwarded to the user the information about the reservation

Figure 2.10: Reservation status

2.2.2.5 *Bind request*

Goal	Binds the access ports to the reserved network resources
Actors	User

Preconditions	• There is at least one active reservation • The user is an administrator or the owner of the reservation to be bound • The NRPSs are active
Basic flow	1. The user indicates the reservation he wants to bind. 2. The system authenticates the user 3. The user indicates the details of the endpoint to be bound 4. The system maps or signals to the client network the IP profile specified on the incoming request on the endpoint
Alternative flow	2a. The user is not allowed to manage the reservation, the system returns an error
Postconditions	The traffic of the client is mapped based on IP characteristics specified on the incoming Bind Request.

Figure 2.11: Bind request

2.2.2.6 *Activation request*

Goal	Activates a service of a reservation
Actors	User, Middleware
Preconditions	• There is at least one inactive reservation in the system • The NRPSs are active
Basic flow	1. The user or the middleware indicate the reservationID of the reservation and the serviceID of the service to activate 2. The system checks if the reservation and the service exist 3. The system sends the activation message to the NRPSs 4. The NRPSs establish the required connections for that service and the NSP updates the database with the required information
Alternative flow	2a. If the reservation or the service do not exist, the system returns an error message 4a. If one or more NRPSs have some problem to establish the connections, an error message is returned and the system rolls-back the activations already done.
Postconditions	The service of the reservation has been activated

Figure 2.12: Activation request

2.2.2.7 *Complete Job*

Goal	Completes a job
Actors	User (through the middleware)

Preconditions	• There is an active job in the system and the user knows its identifier. • The NRPSs are active
Basic flow	1. The user indicates the JobID of the job to be completed. 2. The system checks if the job exists and is still open. 3. The system converts all pre-reservations made in the context of this job to permanent reservations.
Alternative flow	2a. If the job does not exist or is not open anymore, the system returns an error message.
Postconditions	The job has been completed

Figure 2.13: Complete job

2.2.2.8 *Cancel job*

Goal	Cancels a job
Actors	User (through the middleware)
Preconditions	• There is an open job in the system and the user knows its identifier. • The NRPSs are active
Basic flow	1. The user indicates the JobID of the job to be cancelled. 2. The system checks if the job exists and is still open. 3. The system cancels all pre-reservations made in the context of the given job.
Alternative flow	2a. If the job does not exist or is not open anymore, the system returns an error.
Postconditions	The job has been cancelled

Figure 2.14: Cancel job

2.2.2.9 *Retrieve endpoints*

Goal	Retrieves information about the endpoints
Actors	User
Preconditions	There are registered endpoints in the system.
Basic flow	1. The user requests the endpoints to know the availability before making a reservation 2. The system retrieves the endpoints from the database and returns their information to the user

Alternative flow	2a. If there are no registered endpoints (or there is none available), the system returns a message indicating it.
Postconditions	The system has returned to the user the information about the endpoints

Figure 2.15: Retrieve endpoints

2.2.2.10 *Domain management*

Goal	Manage the domains controlled by the system (create, modify, query, delete)
Actors	User (administrator role)
Preconditions	• For modify, query and delete, at least one domain has to be created previously
Basic flow	1. An administrator gives the information required for the operation that he wants to execute 2. The system executes the operation (retrieving/updating the information of the database) and presents the information (if any) to the user
Alternative flow	2a. If there is not any domain created in the system, a message indicating it is returned
Postconditions	The system has queried/created/modified/deleted a domain or has retrieved the information associated

Figure 2.16: Domain management

2.2.2.11 *Link management*

Goal	Manages the links that interconnect the domains controlled by the system (create, modify, query, delete)
Actors	User
Preconditions	• For modify, query and delete, at least one link has to be created previously
Basic flow	1. The user (administrator) gives the information required for the operation that he wants to execute 2. The system executes the operation (retrieving/updating the information of the database) and presents the information (if any) to the user
Alternative flow	1a. If creating a link, the system gives information about the endpoints that can be part of links and then the user selects them 2a. If there is not any link created in the system, a message indicating it is returned
Postconditions	The system has queried/created/modified/deleted a link or has retrieved the information associated

Figure 2.17: Link management

2.2.2.12 *Endpoint management*

Goal	Manages the endpoints belonging to the different domains controlled by the system (create, modify, query, delete)
Actors	User
Preconditions	• For modify, query and delete, at least one endpoint has to be created previously and the user has to execute the Retrieve Endpoints operation
Basic flow	1. The user (administrator) gives the information required for the operation that he wants to execute 2. The system executes the operation (retrieving/updating the information of the database) and presents the information (if any) to the user
Alternative flow	1a. If the information provided is not valid, en error is returned
Postconditions	The system has queried/created/modified/deleted an endpoint or has retrieved the information associated

Figure 2.18: Endpoint management

3 System description

In this section the specification of the interfaces required to build the system is going to be presented. As can be seen in **Figure 3.1**, 5 interfaces are going to be presented since the interface between the NRPS and the Transport Network is already defined by each NRPS and it's out of the scope of this project. It is important to note that two interfaces towards GMPLS and an enhanced version of GMPLS (G^2MPLS) are presented, however the definition of the interface between the NSP and the G^2MPLS may change as the developments of WP1 and WP2 progress.

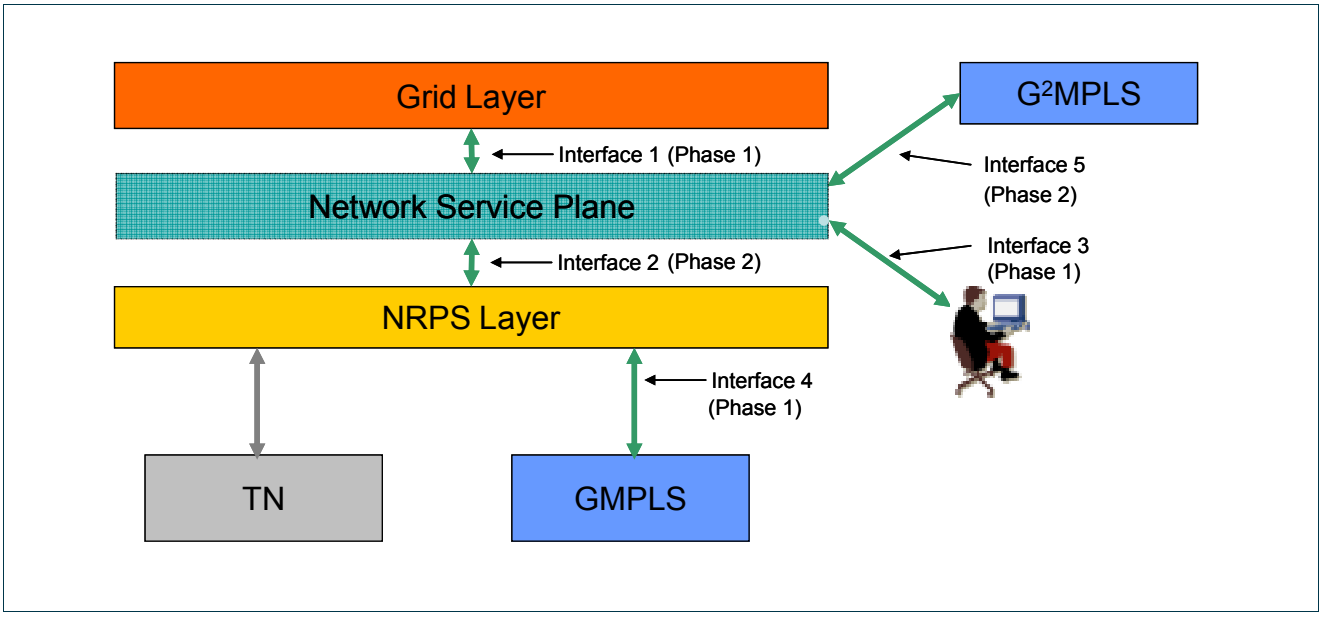

Figure 3.1: System interfaces

It is important to note that the interfaces will be based on Web Services and therefore the messages that will be sent to or received from the interface are going to be defined. However, due to the similarities between layers,

some messages will be reused even though the action triggered by these messages on the different layers will be different. The methods included in interfaces 1, 2 and 3 can be seen in **Figure 3.2** .

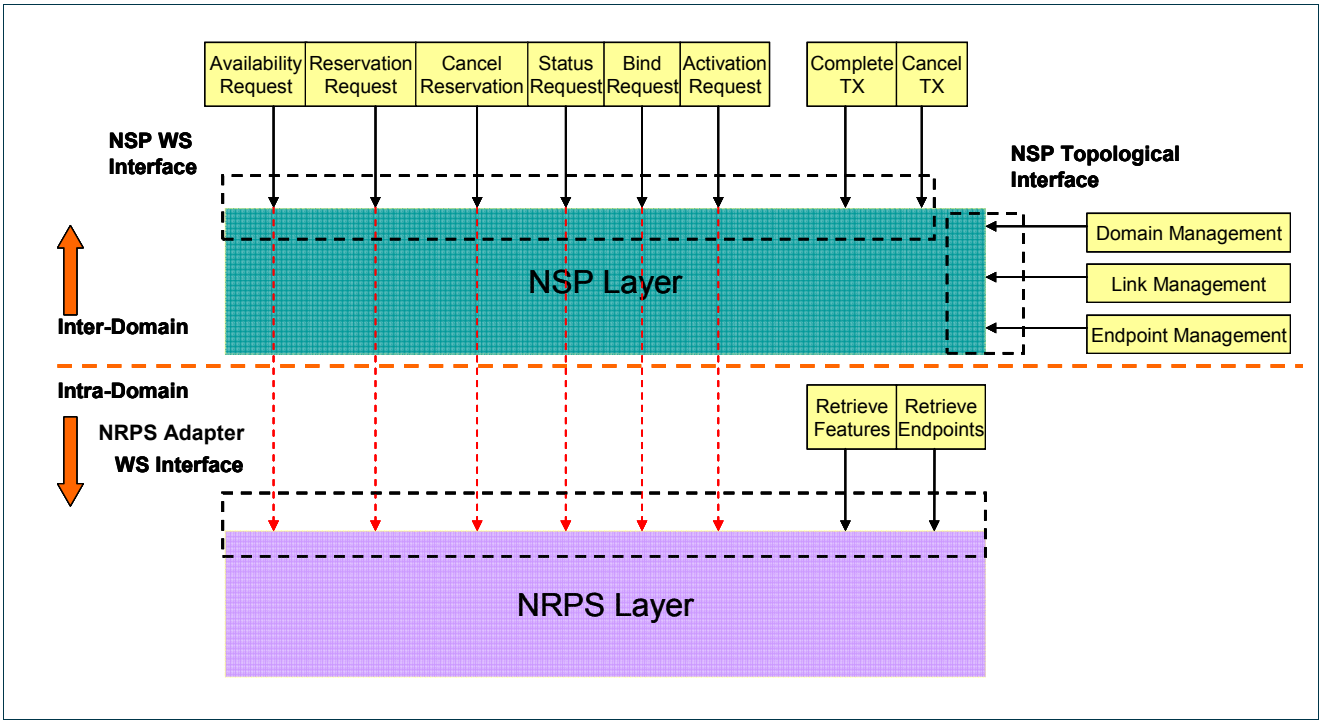

Figure 3.2: Interface 1, 2 and 3 summary

3.1 Network Service Plane internal architecture

In this chapter the preliminary design of the Network Service Plane internal architecture is introduced. The most important functionalities will be identified and briefly described and the main building blocks will be listed and described in order to depict the general architecture and behaviour of the NSP. However during the development phase some of the blocks and functionalities that are described below might be merged, changed or eliminated from the proposal. In case some essential changes to the architecture occur during the development, they will be reflected on this deliverable by means of the release of new versions.

The Network Service Plane (NSP) will act as the coordinator between the Middleware layer and the NRPS layer. It will perform the following functions:

- It processes the user requests and executes the associated actions
- It manages the information about all the objects of the system (users, resources, topology, reservations, …)
- It is responsible for the communication between the Middleware and the Network and between the different NRPSs

- It manages the reservations
- It validates the users
- It performs the interdomain routing
- It manages the transactions
- It keeps track of the underlying network topology

The NRPS Broker is an important block of the NSP responsible for the communication with the NRPSs. The NSP and the NRPS Broker will contain several modules each one of them dedicated to one of the tasks the system has to perform. As a first approach, the modules (or the tasks, not necessarily real modules) that will be integrated in the NSP are:

1. Northbound Interface: this is the interface that receives the requests from the middleware layer. It offers the methods that can be called externally and initiates the actions to perform the required operation.

2. Request Dispatcher: this module distributes the required operations depending on the method that has been called. If the operation is related to reservations, it will pass the call to the Connection Controller.

3. Connection Controller: this will be the engine that executes the work related to management of reservations. It will have a Connection Admission Control that checks if the reservation with the indicated parameters can be done.

4. AAA Module: this block will be the responsible for AAA issues. All the operations requiring authentication will interact with this module. It will be implemented in Phase 2 of the project.

5. DB & DB Controller: this is the module that is responsible for the data persistence. It keeps information about all the objects of the system aligned and the rest of modules interact with it to save, retrieve and modify this information. This will be done through the DB Controller that will offer the DB services to the rest of the modules.

The following modules will be part of the NRPS Broker inside the NSP:

6. Txn Manager: It will be the controller of the transactions. This module will ensure that the operations performed are atomic. A transaction will include the set of operations to be executed in a reservation-related request. If some of the operations do not succeed, the transaction will abort and the system will return to the previous state.

7. Interdomain Routing Module: This block will calculate the paths and routes between the domains when a reservation is requested. It will take into account the objects of the DB before path computation to avoid unnecessary work.

8. NRPS Manager: This module will deal with the communication to/from NRPSs. The communication will be bidirectional between the NRPS Adapters and this module in order to allow interoperability between NRPSs through the NRPS Broker, which implements the East/West interfaces. This way the communication between different NRPSs will be done through the NSP.

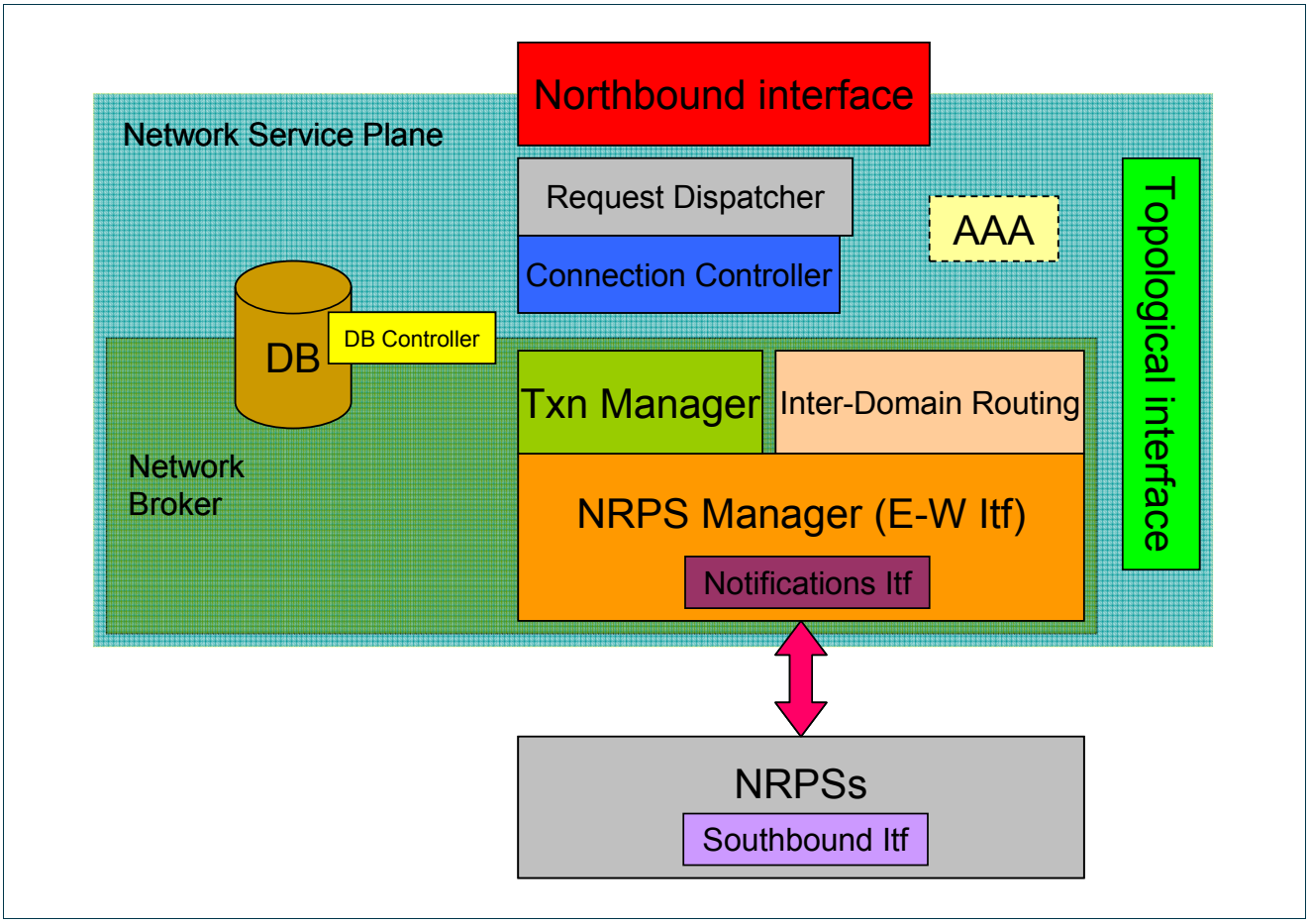

Figure 3.3: First approach to the internal architecture of the NSP

3.1.1 Northbound interface

This is the access point for the requests of the higher layers. It offers the capability to invoke the actions that the Middleware needs in order to establish the end-to-end connections. The operations that can be invoked (explained in detail in section 3.3) are:

- Availability request
- Reservation request
- Cancel reservation
- Status request
- Bind request
- Activation request
- Complete Job
- Cancel Job
- Retrieve features

In order to interact with the NSP, the Middleware must know the location of the northbound interface (the location of the Web Service that contains the interface) to be able to establish contact and invoke the offered operations. This interface receives the requests and returns back messages with information about the results of the invoked action after its execution.

When a request is received, the message is processed (this would be a task of the Request Dispatcher) and is redirected to the proper module which performs the associated actions inside the NSP to offer the requested service. Once the action is executed, the reply message is sent back from the interface to the invoker to inform about the result. Besides the “normal” return messages, the interface could return error messages that should be processed by the invoker.

3.1.2 Request dispatcher

Since the operations that can be requested to the northbound interface are varied, they should be processed in different ways. For this reason there should exist a module that redirects the different types of requests to be processed by other modules or executes directly some operations to give the result efficiently.

The operations related to reservation management will be forwarded directly (after some processing) to the connection manager since this is the more complex process of all. For the other operations, like the ones that only retrieve information, this module could interact with the database directly without forwarding the request to any other module in order to give the response to the invoker in an efficient way.

3.1.3 Connection controller

When the request includes a reservation management operation, the connection controller will be the responsible for the processing. This processing will include some calls to the database to check if the information provided is correct and if the request can be provided a priori. If this verification is successful, the request is processed by the next modules and directed to the lowest layers (NRPSs, GMPLS). Otherwise, the system returns an error message containing the failure reason.

This module should keep the database up to date with the information about the connections in any operation.

3.1.4 AAA module

This part of the software is responsible for the AAA management. The modules that deal with user information should be able to use the functionalities of this module to authenticate the received requests. For a more detailed description please see chapter 1.4 .

3.1.5 DB and DB controller

The database will keep the information about all the physical and logical entities of the system. All the other modules should be able to access the database to retrieve, update and delete its stored information. The use of the database (to check for availability of resources) prior to the execution of some complex operations could improve the performance of the system since it would avoid the invocation of the functions of the lower layers.

The database module will implement a DB controller that will offer an API to manage the DB in a simple and effective way. This controller should be accessible internally from any other part of the NSP and should isolate the specific technology used for the database.

3.1.6 Transaction manager

This part of the NSP will ensure that the complex operations will be executed atomically. Since in the system there are several sub-systems (NRPSs) that will interoperate, it is necessary that they act as only one system to synchronize their operations. This way, if in the execution of an operation there is one of the systems that can not offer the required service and the others can, the transaction where the action is placed will rollback the work and the system (and the sub-systems) will return to a consistent state.

3.1.7 Interdomain routing

The different domains controlled by the system will be connected in different ways through one or more physical links. When a user requests an end-to-end connection the system must show to the user the different paths that the connection could have. Therefore, to find the path or paths for a connection, the interdomain routing module is required. From the information available in the database about endpoints, links and reservations, this module will calculate the possible path or paths for a connection with the given parameters. This block will use a constrained based routing algorithm using some parameters (bandwidth, delay) as constraints to calculate the path that really matches user needs.

3.1.8 NRPS manager

This module will be the one that provides the direct interoperability and communication between the NSP layer and NRPSs and between the NRPSs. The manager will communicate with the NRPSs contacting the NRPS Adapter located at the NRPS side; therefore the NSP have to know the location of each one of the NRPS Adapters to communicate with them. In the same way, the NRPS can send messages to the NSP through this module. The NRPS Adapter will offer the operations defined in the NRPS Adapter WS interface to the NSP (See section 3.4).

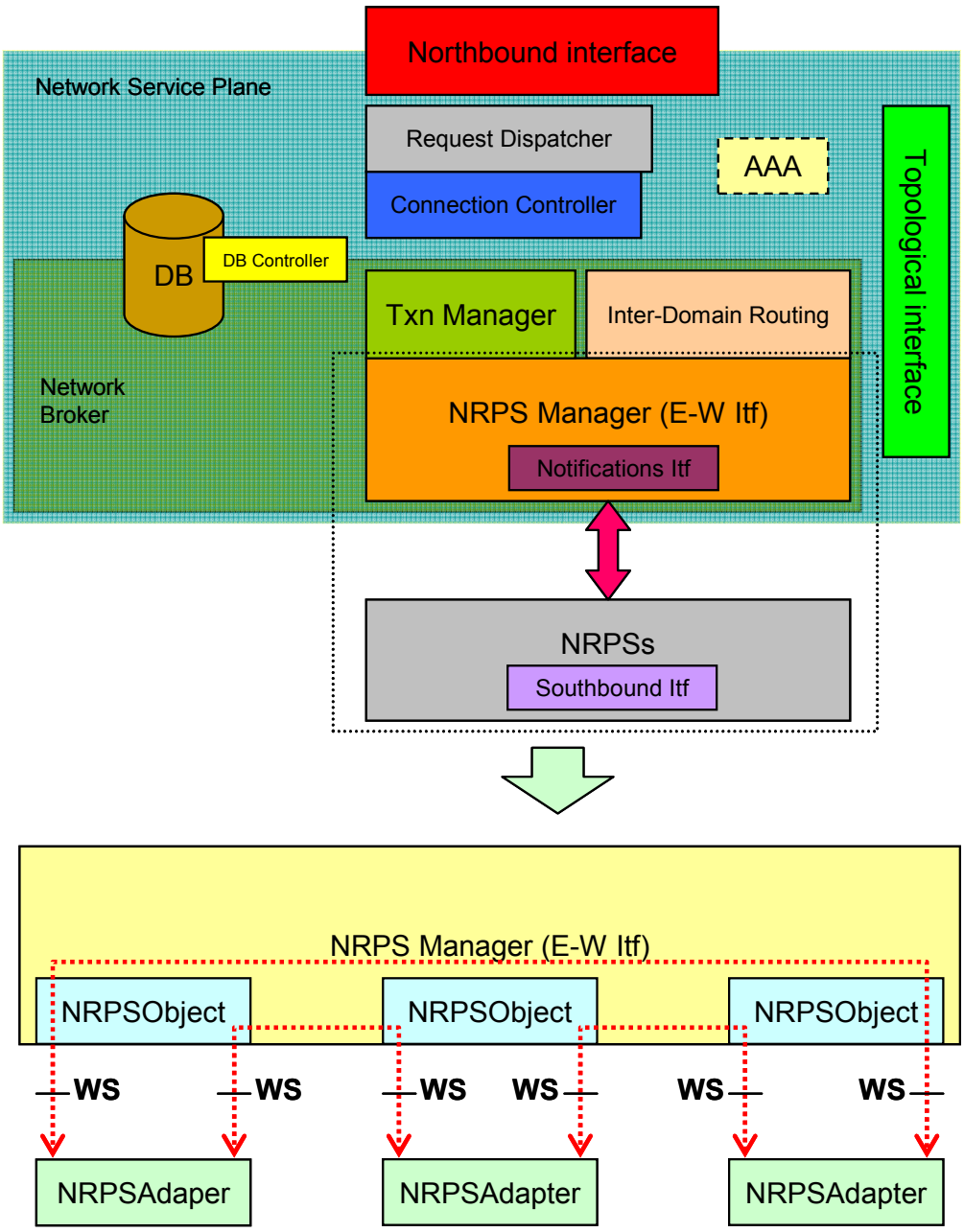

Figure 3.4: Interoperability NSP-NRPS and NRPS-NRPS

According to **Figure 3.4**, the communication between NRPSs and between a single NRPS and the NSP is realized within the NRPS Manager. Inside this module there are several objects (NRPSObject in the figure) that keep the information of each one of the NRPSs. These objects will interact between them to make possible the interoperability between NRPSs and will implement the communication module to contact the NRPS Adapters. In order to make the different NRPS interoperable the NSP will contain all the logics to control the NRPSs and

the NRPS layer will implement the WS interface proposed in this document, avoiding large modifications on this layer.

3.1.9 System execution flow

According to the presented blocks, this section depicts a first approach of the normal execution flow of the system since the moment in which a request arrives to the system until the return of the response message. In particular the figure below describes a sample message flow triggered by a Reservation Request:

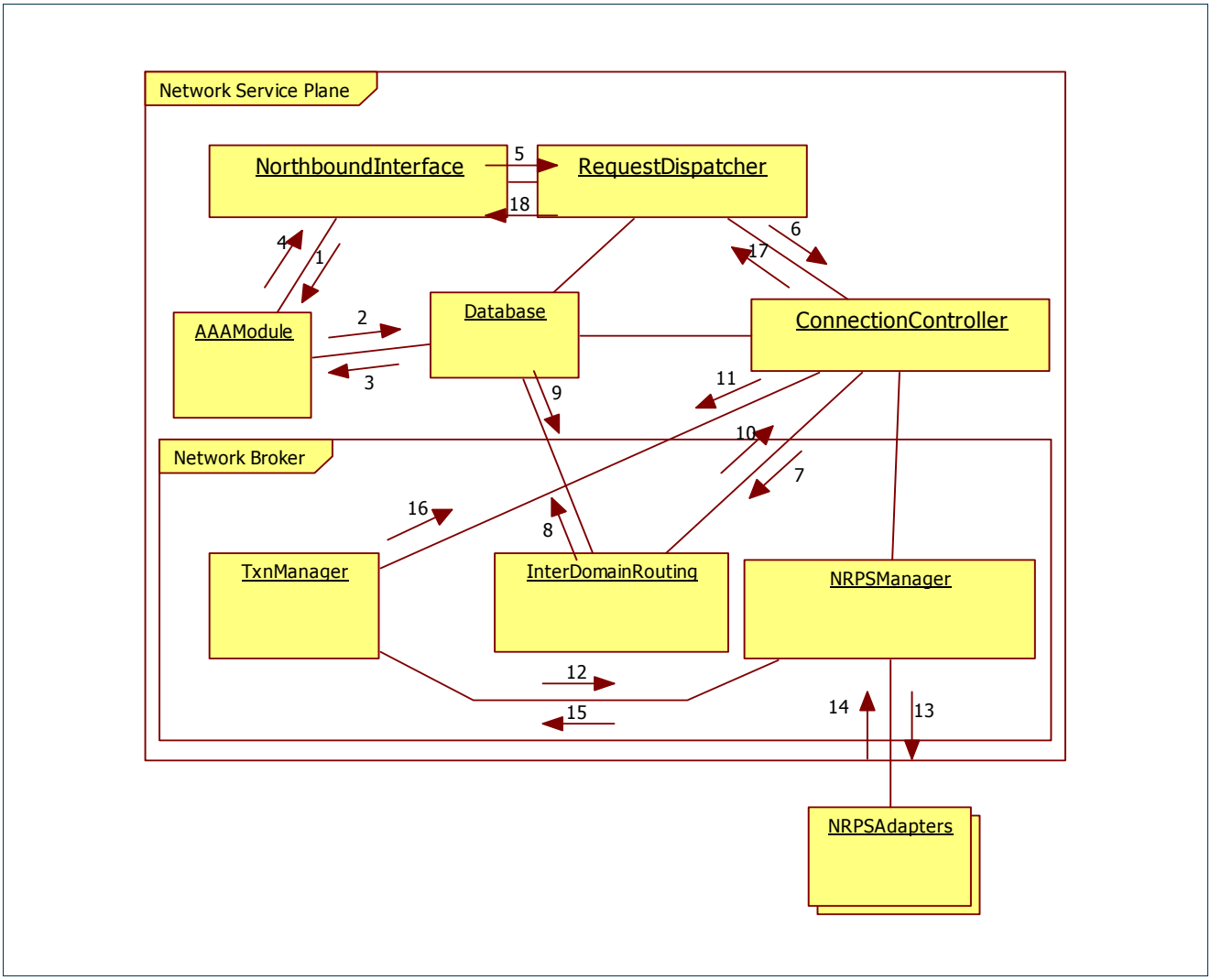

Figure 3.5: sample message flow triggered by a Reservation Request.

Message description:

- (0) A Reservation Request message arrives through the Northbound interface
- (1) Before forwarding the request to the RequestDispatcher the NorthboundInterface checks the user's AAA credentials.
- (2) The AAAModule checks user's AAA credentials against the Database
- (3-5) A message indicating if the user has rights to run this operation is forwarded to the RequestDispatcher along with the operation requested.
- (6) Since the operation invoked requires scheduling features the request is forwarded to the ConnectionController.
- (7-10) The Connection Controller queries the InterDomainRouting block for one path or several paths matching the user requirements in terms of bandwidth or delay. In case there is at least one such path, message number 10 will contain a list of end to end connections that must be set up in each domain to create the desired end to end connection.
- (11-16) Once the ConnectionController knows which interdomain links must be used, it sends a message to the TxnManager containing a list of the Reservation Requests that must be served in each domain to serve the complete Reservation Request. Then the TxnManager will contact all the NRPS Adapters involved in the advance reservation through the NRPSManager and will assure that all domains are able to process successfully. In case one domain is not able to provide the required advance reservation all the actions will be rolled back to return to a consistent status. In case of success message 16 will notify the result of the operation to the ConnectionController. Note that steps (13) and (14) are executed for each domain.
- (17-18) The result of the operation is forwarded to the user.

3.2 Common data types for network services

In this section the data types that are going to be used in the definition of the interfaces will be presented.

3.2.1 AAA data types

3.2.1.1 *The AAA user object*

This object contains the username, user's home organization and secret where:

USER_ID	
Description	Identity by which the user is known to his home organization
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

USER_ORG	
Description	A pointer to a service or server that can verify the users identity
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

SECRET	
Description	The users proof of identity (like password of certificate)
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

3.2.1.2 *The AAA service object*

This object contains the owner of a service where:

OWNER

Description	A pointer to the administrative owner of a service
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

SEQUENCE	
Description	The AAA sequence that is applied for this service.
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	The allowed values are: • PUSH • PULL • AGENT • TOKEN

3.2.2 Connections

The complex ConnectionType stores basic information about a connection: A connection ID (that is unique within all connections aggregated in a single service), source and target endpoints, and the directionality.

3.2.2.1 *ConnectionType*

ConnectionID	
Description	Identifier of the connection, unique within a single service
XML Type	Short
Java Type	Short
Multiplicity	One time per connection
Mandatory	Yes

Details	N.A.

Source	
Description	Source endpoint
XML Type	String
Java Type	String
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

Target	
Description	Target endpoint
XML Type	String
Java Type	String
Multiplicity	Unbounded
Mandatory	Yes
Details	N.A.

Directionality	
Description	Directionality of the connection.
XML Type	Int
Java Type	Int
Multiplicity	One time per connection
Mandatory	Yes
Details	Possible values: • 0="unidirectional tree" • 1="bidirectional tree", • 3="full mesh".

To store a connection together with the constraints it should fulfil, the ConnectionConstraintType is used.

3.2.2.2 *ConnectionConstraintType (extends ConnectionType)*

MinBW	
Description	Minimum bandwidth in Mbps
XML Type	Int
Java Type	Int
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

MaxBW	
Description	Maximum bandwidth in Mbps
XML Type	Int
Java Type	Int
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

MaxDelay	
Description	Maximum delay in milliseconds
XML Type	Int
Java Type	Int
Multiplicity	One time per connection
Mandatory	No
Details	N.A.

To store information about the actual status of a connection, the ConnectionStatusType is used.

3.2.2.3 *ConnectionStatusType (extends ConnectionType)*

ActualBW	
Description	Actual bandwidth in Mbps
XML Type	Int
Java Type	Int
Multiplicity	One time per connection
Mandatory	Yes
Details	N.A.

3.2.3 Services

A service is composed of one or several connections and time constraints that apply to all of these connections (i.e., while connections are of topological nature, services add time constraints).

3.2.3.1 *ServiceType*

ServiceID	
Description	Identifier of a service. It needs to be unique only within the same reservation.
XML Type	Short
Java Type	Short
Multiplicity	One time per service
Mandatory	Yes
Details	N.A.

TypeOfService	
Description	It indicates the technology used.
XML Type	Int
Java Type	Int
Multiplicity	One time per service

Mandatory	Yes
Details	The allowed values are: • 1=L1 • 2=L2 • 3=L3.

TypeOfReservation	
Description	Indicates the constraint type that applies to a service.
XML Type	Int
Java Type	Int
Multiplicity	One time per service
Mandatory	Yes
Details	The allowed values are: • 1=fixed • 2=deferrable • 3=malleable

FixedReservationConstraints	
Description	Service constraints for fixed reservations
XML Type	FixedReservationConstraintType
Java Type	FixedReservationConstraintType
Multiplicity	One time per service
Mandatory	Only if TypeOfReservation=1
Details	N.A.

DeferrableReservationConstraints	
Description	Service constraints for deferrable reservations
XML Type	DeferrableReservationConstraintType
Java Type	DeferrableReservationConstraintType
Multiplicity	One time per service
Mandatory	Only if TypeOfReservation=2

Details	N.A.

MalleableReservationConstraints	
Description	Service constraints for malleable reservations
XML Type	MalleableReservationConstraintType
Java Type	MalleableReservationConstraintType
Multiplicity	One time per service
Mandatory	Only if TypeOfReservation=3
Details	N.A.

AutomaticActivation	
Description	Indicates if the Service will be set up automatically at start time (true) or not (false).
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per service
Mandatory	Yes
Details	N.A.

3.2.3.2 *ServiceConstraintType (extends ServiceType)*

Connections	
Description	Requested connections and their constraints.
XML Type	ConnectionConstraintType
Java Type	ConnectionConstraintType
Multiplicity	Unbounded
Mandatory	Yes
Details	N.A.

3.2.3.3 *DetailedStatusType (extends ServiceType)*

Connections	
Description	Connections aggregated in this service
XML Type	ConnectionStatusType
Java Type	ConnectionStatusType
Multiplicity	Unbounded
Mandatory	Yes
Details	N.A.

ExpectedStatusChange	
Description	Expected starting time for pending service / expected ending time for active service.
XML Type	dateTime
Java Type	dateTime
Multiplicity	Once per DetailedStatusType object
Mandatory	No
Details	N.A.

3.2.3.4 *ServiceStatusType*

ServiceID	
Description	Service ID referenced.
XML Type	Short
Java Type	Short
Multiplicity	Once per ServiceStatusType object
Mandatory	Yes
Details	N.A.

Status	
Description	Status of service
XML Type	Int
Java Type	Int
Multiplicity	Once per ServiceStatusType object
Mandatory	Yes
Details	Allowed values: • 0=error • 1=pending • 2=active • 3=completed • 4=cancelled

ErrorMessage	
Description	Detailed error message
XML Type	String
Java Type	String
Multiplicity	Once per ServiceStatusType object
Mandatory	Only if Status=0
Details	N.A.

DetailedStatus	
Description	Detailed status of service
XML Type	DetailedStatusType
Java Type	DetailedStatusType
Multiplicity	Once per ServiceStatusType object
Mandatory	Only if Status>0
Details	N.A.

3.3 GRID - Network Service Plane Interface

For the communication between the Grid Middleware and the NSP there are two types of users: One the one hand, the applications can directly access the Network Service Plane, and on the other hand, a MetaScheduler performs co-allocation of Grid and network resources. However, it has been agreed upon with WP3 that the same interface can be used in both cases.

All operations in the context of resource reservations are handled by the Network Service Plane in a request/reply manner. Additionally, a notification scheme is to be developed that allows the Network Service Plane to signal a change of a reservation's status.

As will be presented in the following sections Availability Requests, Pre-Reservation Requests, and Reservation Requests can include one or more services. This allows the Middleware or Application Layer to aggregate several services which are only useful if all of them can be set up. If at least one of these services cannot be set up, then the user does not want the other services to be set up either.

In case an availability request fails, alternative starting times for the services have to be returned to allow an efficient workflow planning by the MetaScheduler. There are various possibilities how the constraints between different services can be interpreted: the more powerful the method for describing these constraints, the more services can be aggregated to a single request, which can be advantageous since solutions to resource requests that are feasible might not found if the requests are split and processed in several stages. On the other hand, more complexity is needed in the interface. Therefore, it has been decided to describe three different options in this document, the simplest of which will be mandatory, while the other two can be implemented optionally in later project phases.

In the simple implementation, it is assumed that the time differences between service starting times are to be preserved. E.g., if service 1 has an earliest starting time of t_1 and service 2 has an earliest starting time of $t_2 > t_1$, then the difference between the actual starting times of service 1 and service 2 must be $t_2 - t_1$. In this case, only a single "offset" value is needed to indicate alternative start times for a request that is not feasible. This offset can simply be added to all time values of the original request, yielding a request that is feasible (in case resources are not blocked by other requests in the meantime).

A more advanced feature is to explicitly specify time constraints in the request using an acyclic directed graph, the vertices of which are start and end times of services and whose edges are weighted with time offsets. E.g., an edge with a weight of 300s from the source vertex service2.earliest_start_time to the destination vertex service1.stop_time specifies that service2 may start no earlier than 300s after service1 has terminated. In this case, these constraints can be considered for calculating alternative starting times. Again, a single time offset could be returned, which would have to be added to the absolute deadlines to make the request feasible. However, the processing time of the reservation request might generally be reduced significantly if a vector of alternative starting times is and thus less freedom in planning the complex reservation is given.

The most advanced solution would be to implement a WS-agreement web service just as the MetaScheduler offers towards the applications.

In the next sections the following methods are going to be presented:

- Availability request
- Reservation request
- Cancel reservation
- Status request
- Bind request
- Activation request
- Complete Job
- Cancel Job
- Retrieve features
- Retrieve endpoints
- Notification service

3.3.1 Availability Request

The Availability Request is used to check for availability of resources while planning complex workflows. This request type does not lead to the blocking of resources. This method allows doing advanced queries in order to avoid repetitive reservation/cancel request sequences as best fit across the network is determined. The answer for this query contains information about the network resources' availability (earliest available starting time, etc.). A user can execute this action before trying to make a reservation request in order to know which the available resources are. Then, depending on the answer, the user can request the reservation using the available resources.

3.3.1.1 *Input parameters*

The input for an Availability request method is an "AvailabilityRequest" message that contains the following information:

AAAUserInformation	
Description	AAA information of the user requesting resource availability
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per AvailabilityRequest
Mandatory	Yes
Details	N.A.

Service	
Description	Requested resources
XML Type	ServiceConstraintType
Java Type	ServiceConstraintType
Multiplicity	Unbounded (per AvailabilityRequest)
Mandatory	Yes
Details	N.A.

3.3.1.2 *Output parameters*

The output for an Availability request method is an “AvailabilityReport” message that contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per AvailabilityReport
Mandatory	Yes
Details	N.A.

Result	
Description	Availability of all services together.
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per AvailabilityReport
Mandatory	Yes
Details	A return value of “true” indicates that all services can be set up adhering to the constraints given for each of the services/connections.

ResultDescription	
Description	Reason for availability request failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	Once per AvailabilityReport
Mandatory	No
Details	N.A.

AlternativeStartTime	
Description	Time offset (in seconds) to be added to start times for feasible alternative start times.
XML Type	Long
Java Type	Long
Multiplicity	Once per AvailabilityReport
Mandatory	No
Details	N.A.

DetailedResult	
Description	Detailed availability information for each of the requested connections
XML Type	DetailedAvailabilityResultType
Java Type	DetailedAvailabilityResultType
Multiplicity	Unbounded (per AvailabilityReport)
Mandatory	No
Details	N.A.

DetailedAvailabilityResultType

ServiceID	
Description	Unique identifier of the service
XML Type	Short

Java Type	Short
Multiplicity	Once per DetailedAvailabilityResultType object
Mandatory	Yes
Details	N.A.

ConnectionID	
Description	Unique identifier of the connection
XML Type	Short
Java Type	Short
Multiplicity	Once per DetailedAvailabilityResultType object
Mandatory	Yes
Details	N.A.

Availability	
Description	Indicates if a service is available or not.
XML Type	Int
Java Type	Int
Multiplicity	Once per DetailedAvailabilityResultType
Mandatory	Yes
Details	A value of 1 means that the service corresponding to the given service ID is available whereas a value of 0 means that it is not. A value of 2 indicates that the availability of a service has not been checked. An availability value of 0 or 1 is valid under the assumption that all feasible services preceding this service in the result vector (i.e. services with the result 1 to the left of the current service in the result vector) are also set up. Note that the order of services in the result vector plays an important role since services may compete with each other for resources, so that two services may both be feasible by themselves, but not together The allowed values are: • 0=not available • 1=available • 2=did not check

3.3.2 Reservation / Pre-Reservation Request

A Reservation Request is used to actually reserve resources. A pre-reservation can be used by a Middleware entity to block resources for a certain time while performing several operations to reserve Grid and network resources. In case of a failure, resources already blocked do not have to be cancelled manually but are freed after a certain timeout.

It should be noted that the blocking of pre-reserved services might not be definite. While blocked resources surely will not be available in forthcoming requests from the same user, the system may choose to make these resources available to other users, e.g. if they have a higher priority level.

3.3.2.1 *Input parameters*

The input for a Reservation/Pre-Reservation request method is an “ReservationRequest” message that contains the following information:

AAAUserInformation	
Description	AAA information of user requesting resources
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per ReservationRequest
Mandatory	Yes
Details	N.A.

Service	
Description	Requested resources
XML Type	ServiceConstraintType
Java Type	ServiceConstraintType
Multiplicity	Unbounded (per ReservationRequest)
Mandatory	Yes
Details	N.A.

JobID	
Description	Identifier of the job this pre-reservation belongs to (in case it is a pre-reservation request)
XML Type	JobIdentifierType (long)
Java Type	JobIdentifierType (long)
Multiplicity	Once per ReservationRequest
Mandatory	No
Details	If a job ID is defined, then this is a pre-reservation request, else it is a (permanent) reservation request. A value of 0 means that this request starts a new transaction and hence, a new unique job ID is to be returned in the response.

NotificationConsumerURL	
Description	URL of a NotificationConsumer that is to be notified when any of the services status changes.
XML Type	String
Java Type	String
Multiplicity	Once per ReservationRequest
Mandatory	No
Details	N.A.

3.3.2.2 *Output parameters*

The output for a Reservation/Pre-Reservation request method is a “ReservationReport” message that contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per ReservationReport
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates whether the resources were successfully reserved (true) or not (false).
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per ReservationReport
Mandatory	Yes
Details	N.A.

ReservationID	
Description	If the reservation was successful, a new reservation ID is generated.
XML Type	ReservationIdentifierType (long)
Java Type	ReservationIdentifierType (long)
Multiplicity	Once per ReservationReport
Mandatory	No
Details	N.A.

JobID	
Description	A new job ID is generated by the Network Service Plane in response to a pre-reservation request that does not contain a job ID.
XML Type	JobIdentifierType (long)
Java Type	JobIdentifierType (long)
Multiplicity	Once per ReservationReport
Mandatory	No
Details	N.A.

3.3.3 Cancel reservation

It is possible to cancel a reservation using this method, the input and output parameters are:

3.3.3.1 *Input parameters*

The input parameter for the Cancel Reservation operation is a “CancelReservationRequest” message that contains the following information:

AAAUserInformation	
Description	AAA information of user requesting resource availability
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per CancelReservationRequest
Mandatory	Yes
Details	N.A.

ReservationID	
Description	The identifier of the reservation to be cancelled.
XML Type	Long
Java Type	Long
Multiplicity	Once per CancelReservationRequest
Mandatory	Yes
Details	N.A.

3.3.3.2 *Output parameters*

The output parameter of a Cancel Reservation method is a “CancelReservationReport” message that contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane

XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per CancelReservationReport
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates if the request has been successful (True) or not (False)
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per CancelReservationReport
Mandatory	Yes
Details	N.A.

ResultDescription	
Description	Reason for request failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	Once per CancelReservationReport
Mandatory	No
Details	N.A.

3.3.4 Status request

Status requests are used to query the status of all services contained in a reservation or a specific subset of these services.

3.3.4.1 *Input parameters*

The input parameter of the Status request method is a “StatusRequest” message that contains the following information:

AAAUserInformation	
Description	AAA information of user requesting the status of resources
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per StatusRequest
Mandatory	Yes
Details	N.A.

ReservationID	
Description	ID of the reservation whose status is queried
XML Type	ReservationIdentifierType (long)
Java Type	ReservationIdentifierType (long)
Multiplicity	Once per StatusRequest
Mandatory	Yes
Details	N.A.

ServiceID	
Description	ID of a specific service whose status is queried
XML Type	ServiceIdentifierType (short)
Java Type	ServiceIdentifierType (short)
Multiplicity	Unbounded
Mandatory	No
Details	N.A.

3.3.4.2 *Output parameters*

The output parameter of the Status request method is a “StatusResponse” message that contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per StatusResponse
Mandatory	Yes
Details	N.A.

ServiceStatus	
Description	ID of the reservation whose status is queried
XML Type	ServiceStatusType
Java Type	ServiceStatusType
Multiplicity	Unbounded (per StatusResponse)
Mandatory	Yes
Details	N.A.

3.3.5 Bind Request

This method allows binding the access ports to the reserved network resources later. This is due to the fact that in many cases the final user port is known only a short time before the reservation has to be started.

3.3.5.1 *Input parameters*

The input to the Bind Request method is a “BindRequest” message which contains following information:

AAAUserInformation	
Description	AAA information of user

XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per BindRequest
Mandatory	Yes
Details	N.A.

ReservationID	
Description	Unique identifier of the reservation
XML Type	ReservationIdentifierType (Short)
Java Type	ReservationIdentifierType (Short)
Multiplicity	Once per BindRequest
Mandatory	Yes
Details	N.A.

ServiceID	
Description	Unique identifier of the service
XML Type	ServiceIdentifierType (short)
Java Type	ServiceIdentifierType (short)
Multiplicity	Once per BindRequest
Mandatory	Yes
Details	N.A.

ConnectionID	
Description	Unique identifier of the connection
XML Type	ConnectionIdentifierType (short)
Java Type	ConnectionIdentifierType (short)
Multiplicity	Once per BindRequest
Mandatory	Yes
Details	N.A.

EndPointID	
Description	Identifier of the endpoint to bind
XML Type	EndpointIdentifierType (String)
Java Type	EndpointIdentifierType (String)
Multiplicity	Once per BindRequest
Mandatory	Yes
Details	N.A.

IPAddress	
Description	IP address
XML Type	String
Java Type	String
Multiplicity	Unbounded (per BindRequest)
Mandatory	Yes
Details	N.A.

3.3.5.2 *Output values*

The answer to a “BindRequest” message is a “BindReport” message. This message contains the information described below:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per BindReport
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates if the binding request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per BindReport
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

ResultDescription	
Description	Reason for binding request failure.
XML Type	String
Java Type	String
Multiplicity	Once per BindReport
Mandatory	No
Details	Optional in case of success

3.3.6 Activation request

3.3.6.1 *Input parameters*

This operation is invoked by the upper layers when the reservation start time arrives. It only needs the AAA information, the ReservationID and the ServiceID of the service that the user wants to activate to execute the operation.

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per ActivationRequest
Mandatory	Yes

Details	N.A.

ReservationID	
Description	Unique identifier of the reservation
XML Type	ReservationIdentifierType (long)
Java Type	ReservationIdentifierType (long)
Multiplicity	Once per ActivationRequest
Mandatory	Yes
Details	N.A.

ServiceID	
Description	Unique identifier of the service
XML Type	ServiceIdentifierType (short)
Java Type	ServiceIdentifierType (short)
Multiplicity	Once per ActivationRequest
Mandatory	Yes
Details	N.A.

3.3.6.2 *Output values*

The output for an ActivationRequest is an ActivationReport message, which contains AAA information and details about the activation.

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per ActivationRequest
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates if the activation request has been successful or not
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per ActivationRequest
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

ResultDescription	
Description	Reason for activation request failure
XML Type	String
Java Type	String
Multiplicity	Once per ActivationRequest
Mandatory	Only in case of activation failure
Details	N.A.

3.3.7 Complete Job

Several independent reservations can be grouped to a single job with a common job ID. While the job is not yet completed, these reservations are only loosely blocked as “pre-reservations”. This method allows the user to complete or finish a job; this way all the pre-reservations belonging to this job can be modified to regular reservations with a single request to the Network Service Plane.

3.3.7.1 *Input parameters*

The input to the Complete Job method is composed by an AAA block called “AAAUserInformation” and a field called “JobID” that contains a unique identifier of the job. The blocks that build the input message are the following:

AAAUserInformation

Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per CompleteJobRequest
Mandatory	Yes
Details	N.A.

JobID	
Description	Unique identifier of the job that shall be completed
XML Type	JobIdentifierType (long)
Java Type	JobIdentifierType (long)
Multiplicity	Once per CompleteJobRequest
Mandatory	Yes
Details	N.A.

3.3.7.2 *Output values*

The output for Complete Job Request is a CompletetJobReport message, which contains AAA information and the success field described below:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per CompleteJobReport
Mandatory	Yes
Details	N.A.

Success	
Description	Indicates whether or not the result of the request has been successful or not.

XML Type	boolean
Java Type	boolean
Multiplicity	Once per CompleteJobReport
Mandatory	Only in case of failure of the operation
Details	N.A.

3.3.8 Cancel Job

Several independent reservations can be grouped to a single job with a common job ID. While the job is not yet completed, these reservations are only loosely blocked as “pre-reservations”. This method allows the user to cancel all these “pre-reservations” with a single request in case the job cannot be completed.

3.3.8.1 *Input parameters*

The input to the Cancel Job method is the same than for the Complete Job method (please see section 3.3.7.1).

3.3.8.2 *Output values*

The output to the Cancel Job method is the same as for the Complete Job method (please see section 3.3.7.2).

3.3.9 Retrieve Features

This method provides information about the supported features of the NRPS.

3.3.9.1 *Input parameters*

The input to the Retrieve Features method is a “RetrieveFeaturesRequest” message which contains security information:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per RetrieveFeaturesRequest

Mandatory	Yes
Details	N.A.

3.3.9.2 *Output values*

The answer to a "RetrieveFeaturesRequest" is a "FeaturesReport" message. This message contains an "AAAServiceType" block and a list of "FeatureName" items.

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per FeaturesReport
Mandatory	Yes
Details	N.A.

FeatureName	
Description	Name of the feature
XML Type	String
Java Type	String
Multiplicity	Onbounded (per FeaturesReport)
Mandatory	Yes
Details	The allowed values are: • AvailabiltyRequest • ReservationRequest • ReservationStatusRequest • CancelReservationRequest • EndpoinstRequest • ActivationRequest • BindRequest • Complete job • Cancel job • Retrieve Endpoints • Point to Multipoint • Advance Reservations

3.3.10 Retrieve End Points

This method allows retrieving the available endpoints of the network that are useful for the user to create connections.

3.3.10.1 *Input parameters*

The input to the Retrieve End Points method is an “EndpointsRequest” message which contains AAA information:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	Once per EndpointsRequest
Mandatory	Yes
Details	N.A.

3.3.10.2 *Output values*

The answer to an “EndpointsRequest” message is an “EndpointsReport” message. This message contains an “AAAServiceType” block and a list of “EndpointInformationType” blocks.

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	Once per EndpointsReport
Mandatory	Yes
Details	N.A.

EndpointInformationType

Identifier	
Description	Endpoint unique identifier

XML Type	EndpointIdentifierType (String)
Java Type	EndpointIdentifierType (String)
Multiplicity	Once per EndpointInformationType
Mandatory	Yes
Details	This unique ID will be generated internally by the system

SiteIdentifier	
Description	Identifier of the site connected via this endpoint.
XML Type	Int
Java Type	Int
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	This identifier is used by the middleware to identify endpoints.

LinkID	
Description	Link identifier where the port belongs to
XML Type	Int
Java Type	Int
Multiplicity	Once per EndpontInformationType
Mandatory	No
Details	Optional in case this End Point does not belong to any link.

IPAddress	
Description	IP address of the port
XML Type	String
Java Type	String
Multiplicity	Once per EndpontInformationType

Mandatory	Yes
Details	N.A.

Description	
Description	Description of the port provided by the user that creates the endpoint object
XML Type	String
Java Type	String
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	N.A.

Domain	
Description	Domain where the endpoint is located
XML Type	String
Java Type	String
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	N.A.

Bandwidth	
Description	Bandwidth of the port in Mbps
XML Type	Int
Java Type	Int
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	N.A.

Type

Description	Describes the type of endpoint
XML Type	String
Java Type	String
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	Allowed values: • Interdomain endpoint • Local endpoint

InUse	
Description	Indicates if the port is in use or not
XML Type	Boolean
Java Type	Boolean
Multiplicity	Once per EndpontInformationType
Mandatory	Yes
Details	Allowed values: • TRUE: the port is in use • FALSE: the port is not in use

3.3.11 Notification service

While the Middleware has the possibility to query the status of specific reservations, it is useful to additionally have the possibility to inform the Middleware about changes of a reservation's status. This allows, for example, to check the availability of a deferrable or malleable reservation when it is made, but scheduling its actual start at a later point in time. Also in case of an error a notification service could be used to notify the Middleware entity in charge of a reservation that this cannot be made anymore.

In order to be conforming to current standards, the WS-notification will be implemented to achieve this. In this framework, there are "notification producers" (the Network Service Plane) and "notification consumers" (Middleware entities with pending reservations). Notification consumers can subscribe to certain topics offered by the notification producers. A notification is made by the notification producer by calling the notification consumers' "notify" functions. For example, one topic could be the change of a specific service's status. The message associated to the topic would be therefore the string "StatusChange: <reservation ID>[:<service ID>]". A notification consumer receiving a notification with such a topic can issue a status request to receive additional information on the details of this status change.

The Network Service Plane will not offer a subscription service. Instead, the appropriate subscriptions are automatically made if a notification consumer URL is included in a reservation request.

3.4 Network Service Plane - NRPS Layer Interface

As described in section 2.2.1.1 a user has to be able to request connections to the NSP. But since the NSP does not control directly any network equipment an interface from the NSP to the NRPSs that actually control the physical devices is needed. Therefore an interface to create, delete and query advance reservations in the network managed by an NRPS and also retrieve the endpoints is needed. This interface will be implemented in the NRPSs adapters as a Web Service, and each method of this service is defined by two messages: a request and a report message. An appropriate request message will be sent to ask for a service and a response will be sent back from the NRPS Adapter to report on the success of the operation.

Apart from these functionalities, the NRPS adapter should implement a notification service to be able to send asynchronous messages to the NSP. This service would be responsible for informing the NSP about the events occurring at the NRPS layer. These events must be relevant for the NSP, i.e. the notifications must inform only about the resources under the control of the NSP. These notifications should include: internal link failures, port failures, internal topology changes and other events of this kind affecting the resources controlled by the NSP.

During the design phase many commonalities between this interface and the interface described in section 3.3 has been identified. Because of this it has been decided to use the same interface as in section 3.3. However it is important to note that although the interface will be the same the logics coded inside the NSP and the NRPS Adapter will be different, the NSP will deal with interdomain issues and NRPS-Adapter will deal with intradomain issues. As a side-effect benefit, it will be possible to connect the different NRPSs directly to the Grid Middleware, making able the Grid Middleware to ask for intra-domain services if desired. The methods offered by this interface are the following:

- Availability request
- Reservation request
- Cancel reservation
- Status request
- Bind request
- Activation request
- Complete Job
- Cancel Job
- Retrieve features
- Retrieve endpoints
- Notification service

3.5 NSP topological interface

As described above, the Network Service Plane must gather information about the different links that interconnect the different systems domains that the NSP is coordinating. It must be pointed out that since different domains based on different provisioning systems will be interconnected, the information regarding which links interconnect those domains is only known by the domain administrators. This is due to the fact that a NRPS is not aware if it is connected to another domain, this is only known by the administrator that has connected the two endpoints belonging to different domains.

In order to allow an administrator to enter this information to the NSP a new interface will be created. This way, when an administrator wants to configure the NSP or update its information, he would only have to launch a client that would interact with the NSP through the interface described below. The methods offered by this interface are the following:

- Add domain
- Delete domain
- Edit domain
- Retrieve domain
- Add Endpoints
- Delete Endpoints
- Edit Endpoints
- Retrieve Endpoints
- Add Link
- Delete Link
- Edit Link
- Retrieve Links

3.5.1 Add domain

With this operation the user adds a new domain to be controlled by the system.

3.5.1.1 *Input parameters*

The parameters of this operation are the following:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType

Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

DomainName	
Description	Name for the domain
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

Location	
Description	URL where the NRPS Adapter of this domain can be found
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

AccessPort

Description	
Description	Description of the port
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	E.g.: port5 in switch OME-NY01 allocated for application Z

Domain	
Description	Domain where this port belongs to
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

PortType	
Description	Indicates the type of port
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: lambda, SONET, SDH, L2, L2-VLAN, L3

Address	
Description	It represents a unique address inside the domain.
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	TBD if it is represented by an IPv6 address

AdditionalData	
Description	It represents a unique address inside the domain.
XML Type	String
Java Type	String

Multiplicity	1..*
Mandatory	Yes
Details	It depends on the PortType. For instance for lambda ports represents a list describing all the supported lambdas for that port, or all the VLANs available in case of a L2-VLAN PortType

Status	
Description	Indicates the status of the port.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: up, down, administratively down

Bandwidth	
Description	Bandwidth of the port in Mbps
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.1.2 *Output parameters*

The return message contains following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType

Multiplicity	One time per report
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates if the Add Domain request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

ResultDescription	
Description	Reason for Add Domain request failure
XML Type	String
Java Type	String
Multiplicity	One time per request in case of failure
Mandatory	Only in case of failure
Details	N.A.

3.5.2 Delete domain

This operation deletes a domain of the system.

3.5.2.1 *Input parameters*

The input parameters of this operation are the following:

AAAUserInformation	
Description	AAA information of user

XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

Domain	
Description	Name of the domain that is to be deleted
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.2.2 *Output parameters*

The return message contains the same information than the AddDomain operation (See section 3.5.1.2).

3.5.3 Edit domain

This operation is used to modify the information related to a domain controlled by the system.

3.5.3.1 *Input parameters*

The input message contains the parameters described below:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes

Details	N.A.

OldDomainName	
Description	Name of the domain to be modified
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

NewDomainName	
Description	New name of the domain
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	No
Details	Used to change the name of the domain

Location	
Description	URL where the NRPS Adapter of this domain can be found
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.3.2 *Output parameters*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.4 Retrieve domains

3.5.4.1 *Input parameters*

This operation requests all the domains registered in the system. The input is composed by the following information:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.4.2 *Output parameters*

The output message contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per report
Mandatory	Yes
Details	N.A.

DomainsName	
Description	List of the names of all domains
XML Type	String
Java Type	String
Multiplicity	As many as domains are controlled by the system

Mandatory	Yes
Details	N.A.

3.5.5 Add endpoints

This operation adds one or more endpoints to a domain. This is needed to make the system able to control new ports of a domain.

3.5.5.1 *Input parameters*

The input parameters for this operation are the following:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

Domain	
Description	Identifier of the domain where the endpoints are to be included
XML Type	String
Java Type	String
Multiplicity	One time per report
Mandatory	Yes
Details	N.A.

EndPoints	
Description	List containing the endpoints to be added

XML Type	EndpointIdentifierType (String)
Java Type	EndpointIdentifierType (String)
Multiplicity	One or more
Mandatory	Yes
Details	N.A.

3.5.5.2 *Output parameters*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.6 Delete endpoints

This operation deletes one or more endpoints of a domain.

3.5.6.1 *Input parameters*

The input parameters are the same than the ones described in section 3.5.5.1.

3.5.6.2 *Output parameters*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.7 Edit endpoints

This operation modifies one or more endpoints of a domain.

3.5.7.1 *Input parameters*

The input parameters are the same as the ones described in section 3.5.5.1. It is recommended to first retrieve the information of the ports to edit using the Retrieve End Points method in order to edit directly the information to be modified from the data structure returned from the NSP. This way this data structure can be reused as an input parameter reducing the probability that an error occurs.

3.5.7.2 *Output parameters*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.8 Retrieve endpoints

3.5.8.1 *Input parameters*

This operation requests all the endpoints registered in the system. The input parameters are the following:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.8.2 *Output parameters*

The output parameters are the same as the ones described in section 3.5.1.2 plus a list of AccessPort objects (defined in section 3.5.1.1) with all endpoints.

3.5.9 Add link request

Adds a link between two different domains to the database of the NSP.

3.5.9.1 *Input parameters*

The input parameters are composed by the following information:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

Link

Description	
Description	Description of the link, i.e. for what is it going to be used, which domains communicate, etc.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	The information contained on this field should help to figure out what is the purpose of this link.

LinkID	
Description	Unique identifier of the link
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

SourcePort	
Description	Description of the source port
XML Type	AccessPort
Java Type	AccessPort
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

DestinationPort	
Description	Description of the destination port
XML Type	AccessPort

Java Type	AccessPort
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.9.2 *Output values*

The answer for an Add Link Request is an Add Link Report that contains the following fields:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per report
Mandatory	Yes
Details	N.A.

Result	
Description	Indicates if the Add link request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

ResultDescription	
Description	Reason for Add Link request failure.
XML Type	String
Java Type	String

Multiplicity	One time per request
Mandatory	No
Details	Optional in case of success

LinkID	
Description	Unique identifier of the link.
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	No
Details	The LinkID is generated automatically by the NSP in case of success of the operation and therefore this field is only required in this case.

3.5.10 Delete link request

Deletes a link between two different domains from the database of the NSP. The link to be removed can be indicated by providing its identifier.

3.5.10.1 *Input parameters*

The input parameters are composed by the following fields:

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

LinkID

Description	Unique identifier of the link to be deleted
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.10.2 *Output values*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.11 Edit link request

Edits all or some of the parameters of a link between two different domains stored in the database of the NSP.

3.5.11.1 *Input parameters*

The input parameters are composed by an “AAAUserType”, two “AccessPort” objects describing the source and destination port of the link and a “LinkID” field. The “LinkID” is the only parameter that can not be changed, since it is used to specify the Link to be edited. All the rest of parameters contained in this message will override the parameters stored in the NSP database.

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

LinkID	
Description	Unique identifier of the link
XML Type	Int

Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

SourcePort	
Description	Description of the source port
XML Type	AccessPort
Java Type	AccessPort
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

DestinationPort	
Description	Description of the destination port
XML Type	AccessPort
Java Type	AccessPort
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.11.2 *Output values*

The output parameters are the same as the ones described in section 3.5.1.2.

3.5.12 Retrieve links request

Provides the user with information about all the links of a Domain stored in the NSP database.

3.5.12.1 *Input parameters*

The input parameters are composed by AAA information and a “Domain” field that indicates from which domain the information must be retrieved.

AAAUserInformation	
Description	AAA information of user
XML Type	AAAUserType
Java Type	AAAUserType
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

Domain	
Description	Name of the Domain from where all links are going to be retrieved.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	N.A.

3.5.12.2 *Output values*

The answer for a Retrieve Link Request is a Retrieve Link Report that contains the following information:

AAAServiceInformation	
Description	AAA information of Network Service Plane
XML Type	AAAServiceType
Java Type	AAAServiceType
Multiplicity	One time per report
Mandatory	Yes

Details	N.A.

Result	
Description	Indicates if the request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per report
Mandatory	Yes
Details	N.A.

ResultDescription	
Description	Reason for request failure
XML Type	String
Java Type	String
Multiplicity	One time per report in case of failure
Mandatory	Only in case of failure
Details	N.A.

Links	
Description	List of “Link” types with information about all links of a domain.
XML Type	Link (See section 3.5.9.1)
Java Type	Link
Multiplicity	Unbounded
Mandatory	No
Details	N.A.

3.6 NRPS – GMPLS CP interface

The purpose of this section is to define a service platform which enables a NRPS to establish and maintain point-to-point connections in a GMPLS instance.

3.6.1 Assumptions

It is assumed that the GMPLS instance exclusively accepts requests from the controlling NRPSs instance. Directly connected users will not be able to request LSP and the GMPLS will not support multi domain connection setups. Protection and restoration mechanisms of the GMPLS instance have to be constricted to a limited set or completely disabled.

These assumptions are necessary to guarantee the validity of the NRPS view on the network and they are imperative to support any kind of advance reservation.

During the development of the fully functional GMPLS interface with explicit routing capabilities and topology discovery there will also be a “thin” solution combined with a “thin” NRPS system, which does not support advance reservation i.e. this version will not support the specification of an explicit path and the retrieval of detailed topology information. In this case only the endpoints (TNA-Addresses) are returned by the topology service

The following sections describe the methods offered by the GMPLS-WS and the content of the messages which are exchanged during service requests.

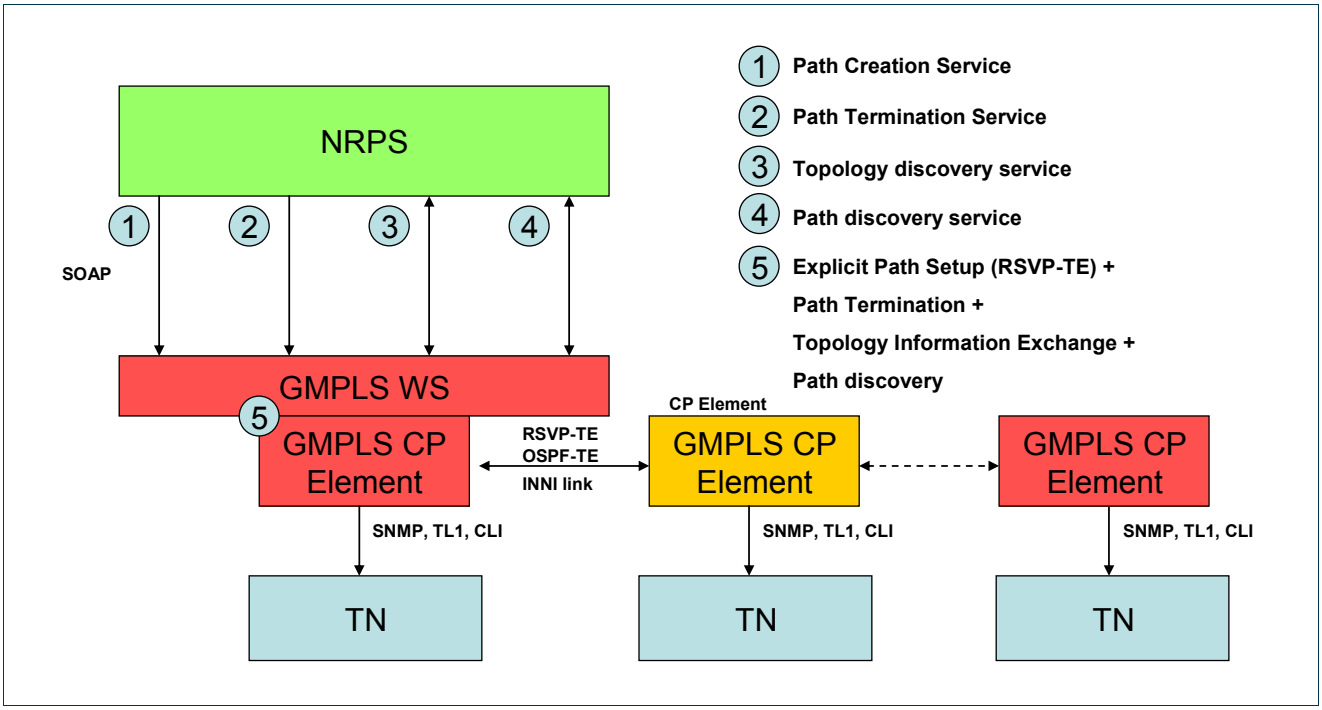

Figure 3.6: NRPS – GMPLS interface architecture overview

3.6.2 Architecture

3.6.2.1 *Pull Model (Web Service)*

Figure 3.6 shows a high level overview on the system architecture. The main interface towards the GMPLS control plane (GMPLS CP) is realized in the GMPLS web service (GMPLS WS). This interface offers several basic services like path creation and path termination along with some more complex services like topology discovery and path discovery towards the NRPS. The functionalities necessary to provide these services are placed in a GMPLS CP element which is directly participating in the OSPF-TE and the RSVP-TE signalling of the GMPLS CP (I-NNI interface).

GMPLS CP elements which are accessible form the web service are marked in red and have to be deployed on every “edge” of the GMPLS network. Edges are defined either as user access points or domain border points.

Instead of putting the required functionality into each GMPLS CP (distributed solution) a dedicated GMPLS CP (stub node) could serve as a central interface to all other GMPLS CPs (centralized solution). More details about this can be found in deliverable D2.1 of WP2.

3.6.2.2 *Push Model*

To provide an up-to-date view to the NRPS on network topology and used network resources as well as status of established LSPs, a second interface is defined. By registering to this service the NRPS will be informed about important changes in the network on an event driven basis. Possible events are topology changes (like new network elements or links) and path status changes (like LSP failures). Figure 3.7 shows how the event driven mechanism is integrated in the system as an addition to the interface presented in Figure 3.6.

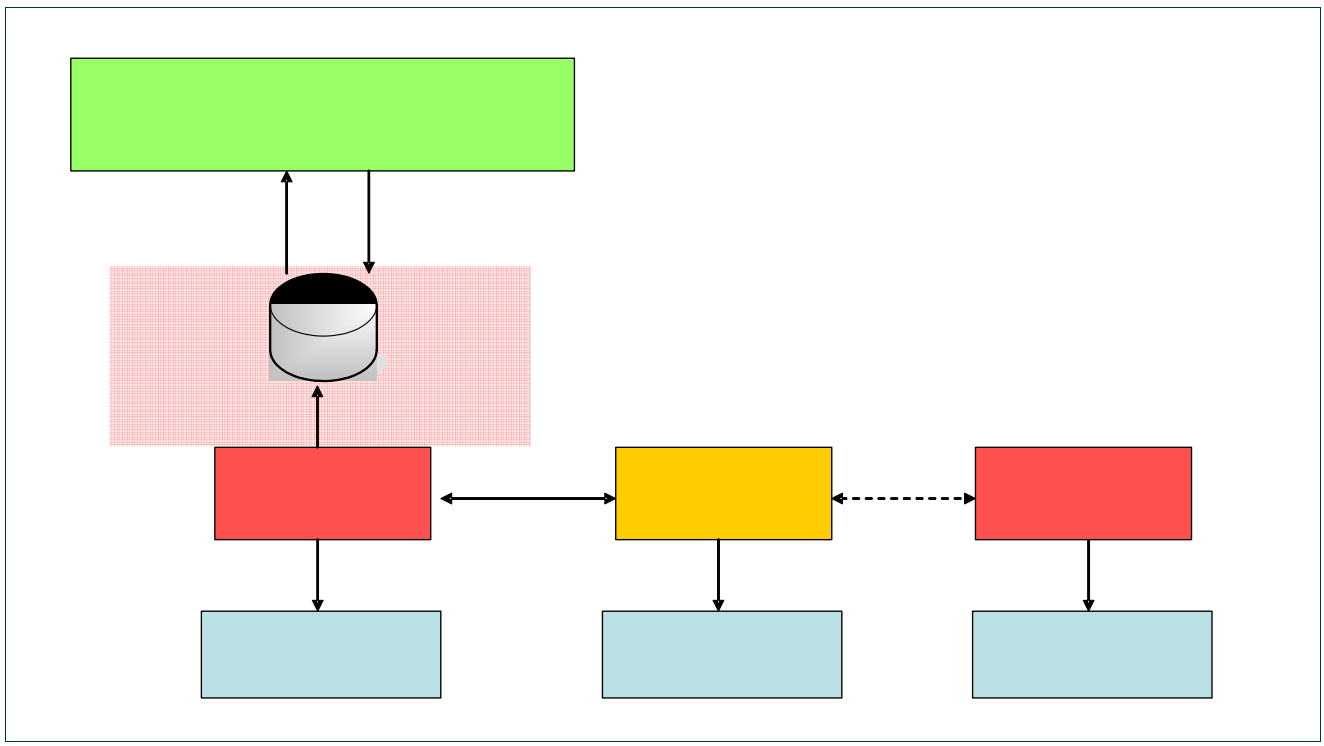

Figure 3.7: Architecture of the event driven GMPLS interface

NRPS

3.6.3 List of services

3.6.3.1 *Pull Model*

Pull Model Services	
Creation Service	The purpose of this service is to provide a basic point-to-point path creation service between two ports specified by TNA addresses.
Termination service	The purpose of this service is to provide a mechanism to tear down LSPs which had been set up by a NRPS.

message

Event queue

Publishing events

GMPLS WS

GMPLS CP

RSVP-TE
OSPF-TE

Path monitoring service	On request this service provides status information about the specified LSP.
Path discovery service	This service provides an interface to retrieve any established point to point connection in the controlled network. It is basically needed to refresh the NRPS view on the network in case of memory losses through system failures or maintenance reboots.
Topology service	The purpose of this service towards a NRPS is to provide access to the OSPF-TE database (LSA information) containing all link and network device topology information within the controlled domain. Filters can be defined to concentrate the output on certain elements of the network.

3.6.3.2 *Push Model*

Push Model Services	
Registration Service	This service registers the NRPS to receive messages from the WS in case of path status changes or topology changes.
Path monitoring service	The Path monitoring services provides status change information for every LSP in the controlled domain. In case of changes the event driven mechanism will send an update on the path status containing the new status and the Path ID to all registered and authenticated systems.
Topology update service	The topology update service provides all registered and authenticated systems with an event driven status update on the topology of the controlled network domain.

3.6.4 Registration service for receiving update messages

This service registers the NRPS to receive messages from the WS in case of path status changes or topology changes.

3.6.4.1 *Input parameters*

Destination	
Description	The destination address for the push messages
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	The string contains the URL of the destination to

	which the push messages are send.

3.6.4.2 *Output values*

Error/Return Code	
Description	Indicates if the registration for push mode has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

ErrorDescription	
Description	Reason for registration failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	No
Details	String containing detailed error information

3.6.5 Creation service

The purpose of this service is to provide a basic point-to-point path creation service between two ports specified by TNA addresses.

3.6.5.1 *Input parameters*

Source TNA	
Description	TNA-Address of the source port of the requested LSP

	on the edge of the network.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	IPv4, IPv6 or NSAP address format.

Destination TNA	
Description	TNA-Address of the destination port of the requested LSP on the edge of the network.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	IPv4, IPv6 or NSAP address format.

Path	
Description	A list of intermediate network nodes and links defining the path from Source TNA to Destination TNA.
XML Type	List of pairs (IPv4 address, structure)
Java Type	List of pairs (IPv4 address, structure)
Multiplicity	One time per request
Mandatory	No, only used in the case of advance reservation
Details	A network node is specified by its loopback address (IPv4-Format). The structure specifies detailed information of the outgoing TE-link from the corresponding node. This information includes at least LinkId and component LinkId in case of bundled links. The structure of the node belonging to the Destination TNA is empty, as there is no further link.

Bandwidth	
Description	Amount of bandwidth required for the LSP in Mbps.
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	Yes
Details	Bidirectional connections are always symmetrical

Bidirectional	
Description	Indicates if the LSP is symmetrical and bidirectional or unidirectional
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	Allowed values: • True: bidirectional and symmetrical • False: unidirectional

Constraint Object	
Description	List of additional requirements/constraints on the requested LSP. Basic elements are QoS constraints like latency and jitter but also protection/reservation features such as diverse paths (shared risk link group values).
XML Type	List of pairs (String, String)
Java Type	List of pairs (String, String)
Multiplicity	One time per request
Mandatory	No
Details	First string indicates constraint, i.e. “latency” Second String indicates value for constraint, i.e. “10ms”.

3.6.5.2 *Output values*

Result	
Description	Indicates if the creation request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

LSP handle	
Description	An ID which identifies the created path in later requests.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	No
Details	Character representation of LSP identifier

ErrorDescription	
Description	Reason for creation request failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes, in case of error
Details	String containing detailed error information

3.6.6 Termination service

The purpose of this service is to provide a mechanism to tear down LSPs which had been set up by a NRPS.

3.6.6.1 *Input Parameters*

LSP handle	
Description	The ID referring to the LSP which should be terminated.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	Character representation of LSP identifier

3.6.6.2 *Output values*

Result	
Description	Indicates if the termination request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of success • False: in case of failure

ErrorDescription	
Description	Reason for termination request failure, therefore only needed in case of failure.
XML Type	String
Java Type	String

Multiplicity	One time per request
Mandatory	Yes, in case of error
Details	String containing detailed error information

3.6.7 Path monitoring service (push)

The Path monitoring service provides status change information for every LSP in the controlled domain. In case of changes the event driven mechanism will send an update on the path status containing the new status and the identification number (Path ID) to all registered and authenticated systems.

3.6.7.1 *Output values: update message*

Status Code	
Description	Integer which indicates the status of the LSP
XML Type	Int
Java Type	Int
Multiplicity	One time per message
Mandatory	Yes
Details	This field will contain codes for possible states of LSPs in the network, the allowed values are: • 0: LSP is up • 1: LSP is down • 2: LSP rerouted

LSP handle	
Description	The ID referring to the LSP with status change
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes
Details	Character representation of LSP identifier

3.6.8 Path monitoring service (pull)

On request this service provides status information about the specified LSP.

3.6.8.1 *Input parameters*

LSP handle	
Description	The ID referring to the LSP which should be checked
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	No
Details	Character representation of LSP identifier

3.6.8.2 *Output values*

Result	
Description	Indicates if the LSP is up or down
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of LSP is up • False: in case of LSP is down

Path	
Description	A list of intermediate network nodes and links defining the path from Source TNA to Destination TNA.
XML Type	List of pairs (IPv4 address, Int)
Java Type	List of pairs (IPv4 address, Int)
Multiplicity	One time per request

Mandatory	No, only used in the case of advance reservation
Details	A network node is specified by its loopback address (IPv4-Format). The integer specifies the LinkId of the outgoing link from the corresponding node. The LinkId of the node belonging to the Destination TNA is “0”, as there is no further link.

ErrorDescription	
Description	Reason why the LSP has gone down or has been rerouted in case it happen.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes, in case of error
Details	If an LSP does not exist any longer, this will be indicated by LSP down and ErrorDescription = “Deleted”

Optionally this service my also retrieve information for all LSPs, if the LSP handle is specified as an empty string. In this case on additional output parameter LSP handle is needed. The output would then be a list of structures containing the values (LSP handle, Result, Path, ErrorDescription)

3.6.9 Path discovery service

This service provides an interface to retrieve any established point to point connection in the controlled network. It is basically needed to refresh the NRPS view on the network in case of memory losses through system failures or maintenance reboots.

3.6.9.1 *Input parameters*

The service request is sent without any parameters.

3.6.9.2 *Output values*

Error/Return Code	
Description	Indicates if the path discovery request has been successful
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: in case of LSP is up • False: in case of LSP is down

Path List	
Description	A list of all LSPs in the network is returned.
XML Type	list of string
Java Type	list of string
Multiplicity	One time per request
Mandatory	Yes
Details	An LSP is defined by its LSP handle.

ErrorDescription	
Description	Reason for path discovery failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes, in case of error
Details	String containing detailed error information

3.6.10 Topology service (pull)

The purpose of this service towards a NRPS is to provide access to the OSPF-TE database (LSA information) containing all link and network device topology information within the controlled domain. Filters can be defined to concentrate the output on certain elements of the network.

3.6.10.1 *Input Parameters*

Type	
Description	Specifies type of the link state attributes the NRPS requires information about.
XML Type	Int
Java Type	Int
Multiplicity	One time per request
Mandatory	No
Details	Allowed values are: • 1: return node information • 2: return link information • 4: return TNA information The values may be added, e.g. 7 means return all information

Node address	
Description	IPv4 address of the node the NRPS requires information about. Only LSAs belonging to the node are returned. If not specified LSA of all nodes in the network will be forwarded.
XML Type	IPv4 address
Java Type	IPv4 address
Multiplicity	One time per request
Mandatory	No
Details	IPv4 address of the node for which detailed information is requested.

3.6.10.2 *Output values*

Nodes	
Description	Information about nodes.
XML Type	List of Pair(IPv4 address, string)
Java Type	List of Pair(IPv4 address, string)
Multiplicity	One time per request
Mandatory	Yes
Details	The IPv4 address specifies the loopback address of a node. The string specifies the type of the node. Allowed values are: • "TDM" for Time-Devision Multiplex (SDH) • "LSC" for Lambda Switch Capable (all optical) • "L2SC" for Layer-2 Switch Capable (Ethernet) • "FSC" for Fiber-Switch Capable For hybrid switches a list of values can be returned.

TNAs	
Description	Information about TNAs.
XML Type	List of Triples(TNA address, IPv4 address, list of int)
Java Type	List of Triples(TNA address, IPv4 address, list of int)
Multiplicity	One time per request
Mandatory	Yes
Details	The IPv4 address specifies the node to which this TNA address belongs. The list of integers specifies the possible bandwidths in mbps supported by the interface, which is given by the TNA address. The bandwidth list is needed in case of SDH to support Ethernet services with different bandwidths. In case of All Optical Switches the bandwidth value indicates the bandwidth of the transponder defined by the TNA address.

Links	
Description	Information about Links.
XML Type	List of Triples(IPv4 address, IPv4 address, structure)
Java Type	List of Triples(IPv4 address, IPv4 address, structure)
Multiplicity	One time per request
Mandatory	Yes
Details	The IPv4 addresses specify the nodes which this link connects. The structure specifies detailed information about the link. Mandatory are the fields LinkId and the maximal bandwidth in mbps.

Error/Return Code	
Description	Indicates if the topology request has been successful.
XML Type	Boolean
Java Type	Boolean
Multiplicity	One time per request
Mandatory	Yes
Details	The allowed values are: • True: the topology request has been successful • False: the topology request failed

ErrorDescription	
Description	Reason for topology service failure, therefore only needed in case of failure.
XML Type	String
Java Type	String
Multiplicity	One time per request
Mandatory	Yes, in case of error
Details	String containing detailed error information

3.6.11 Topology update service (push)

The topology update service provides all registered and authenticated systems an event driven status update on the topology of the controlled network domain.

3.6.11.1 *Input parameters*

None

3.6.11.2 *Output values: update message*

The following objects defined in the topology service (pull) are returned: Nodes, TNAs and Links.

3.7 NSP – G²MPLS CP – GN2 GÉANT2 JRA3 Interface

During Phase 2 of the project it is planned to extend the work done during Phase 1 to make our system interoperable with the outcome of WP2 and other European and non European projects. In particular WP1 has already contacted WP2 and GN2-GÉANT2 JRA3 representatives to start discussing possible architectures and interfaces as well as the EnLIGHTened and G-Lambda projects. However due to the current stage of the developments inside WP2 and GÉANT2 JRA3 it hasn't been possible to have an interface specification ready for this deliverable.

As can be seen in **Figure 3.8**, at this point of the design the most convenient layer to create interfaces to provide interoperability with other systems is the NSP. It is planned to interchange routing and signalling information through new interfaces implemented at the NSP, however the details of these interfaces are still under discussion.

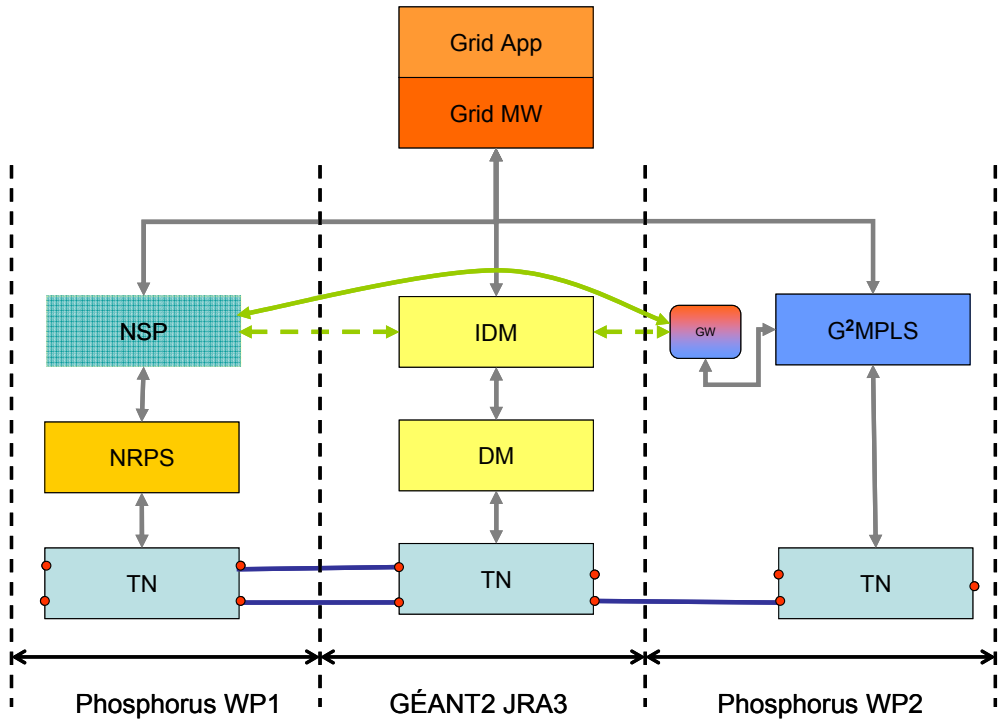

Figure 3.8: Interconnection possibilities of Phase 2 architecture

4 Flow diagrams

In this section some high level flow diagrams of our system are going to be presented. Due to the fact that this document is intended to describe the behaviour of a system able to make interoperable different NRPS, the functions done at the Network Service Plane will be described. It is assumed that the functions implemented at the NRPS Adapter level and at the NRPS will be described in detail by the different teams providing NRPSs on the next deliverables.

4.1 Notification Service

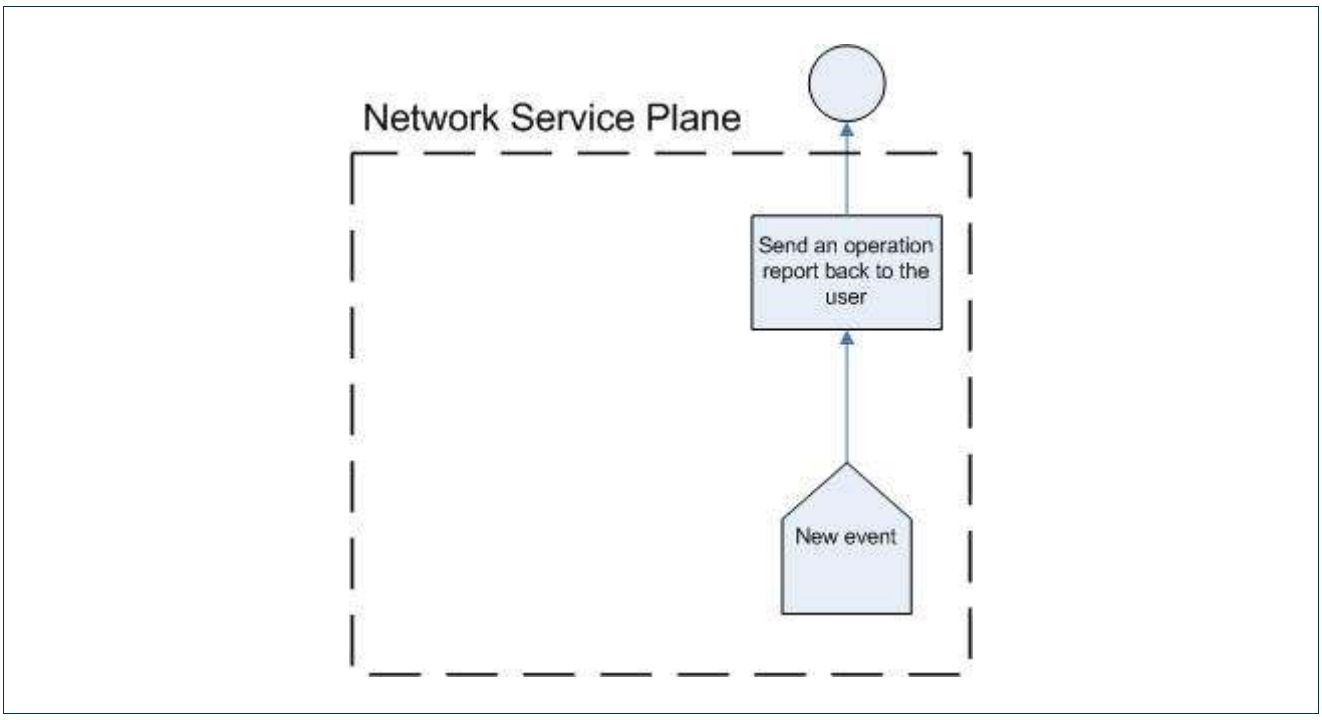

Figure 4.1: High level flow diagram for a Notification Service.

4.2 Availability Request

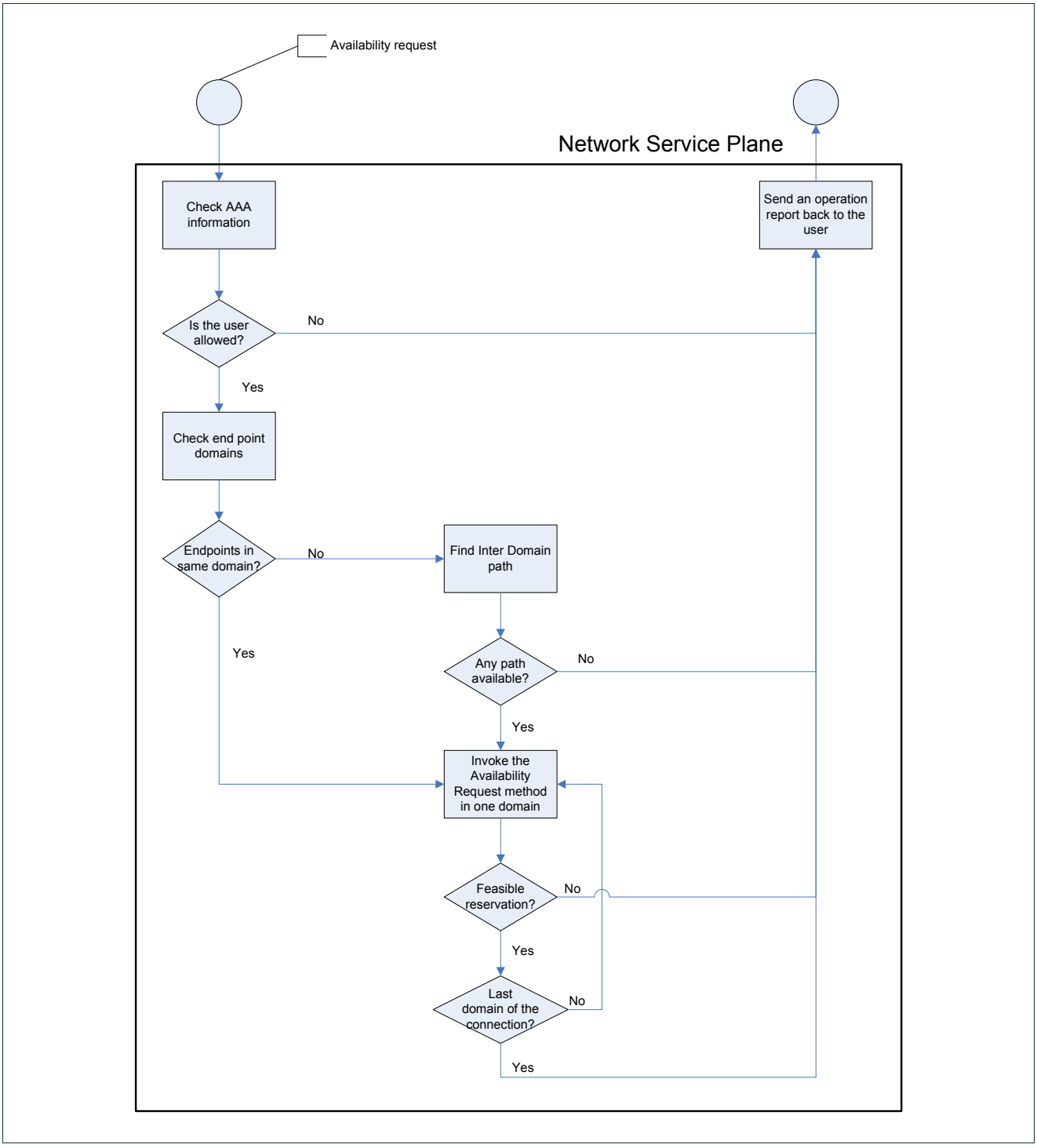

Figure 4.2: High level flow diagram for an Availability Request operation.

4.3 Reservation Request

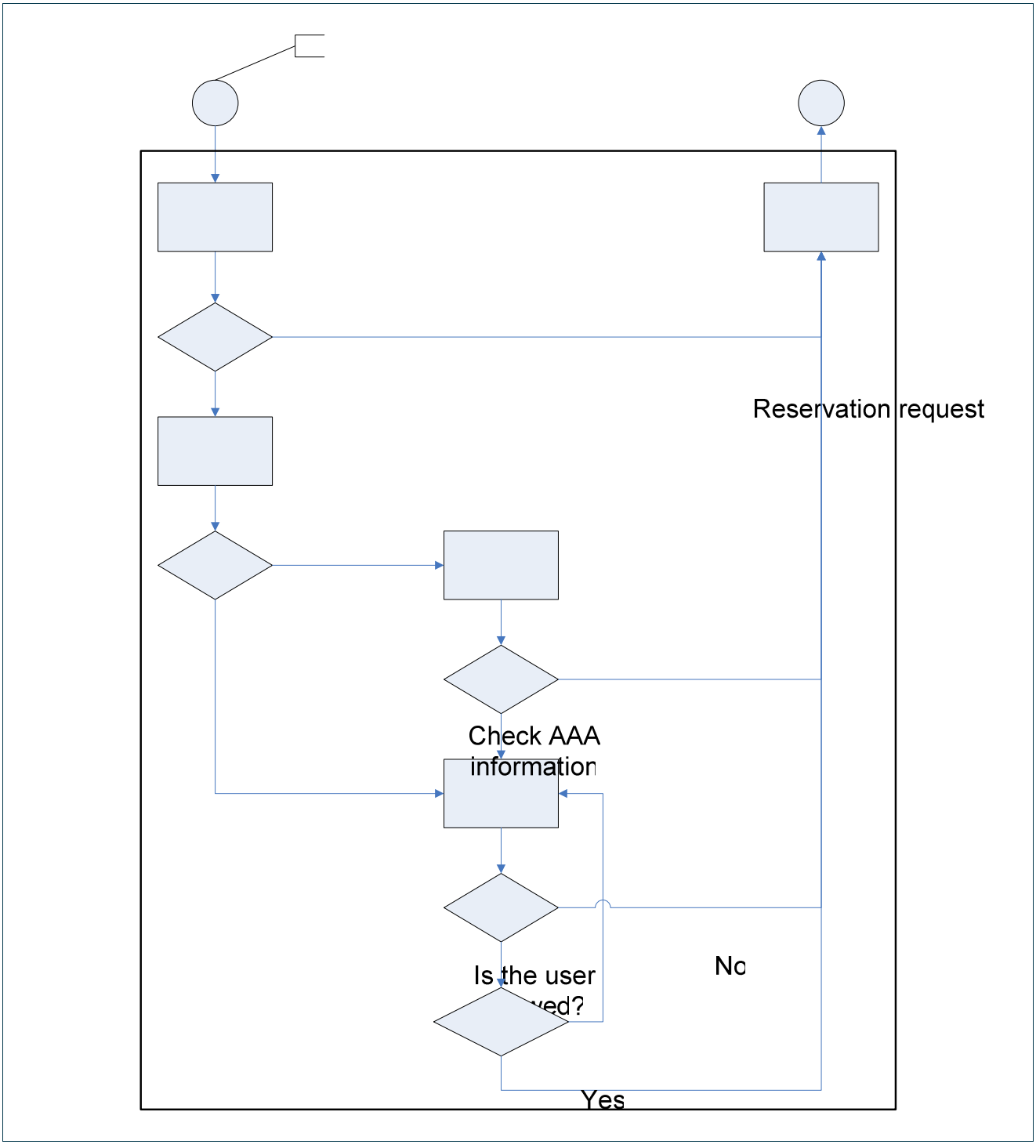

Figure 4.3: High level flow diagram for a Reservation Request operation.

Check end point domains

Endpoints in No Find Inter D

4.4 Cancel Reservation

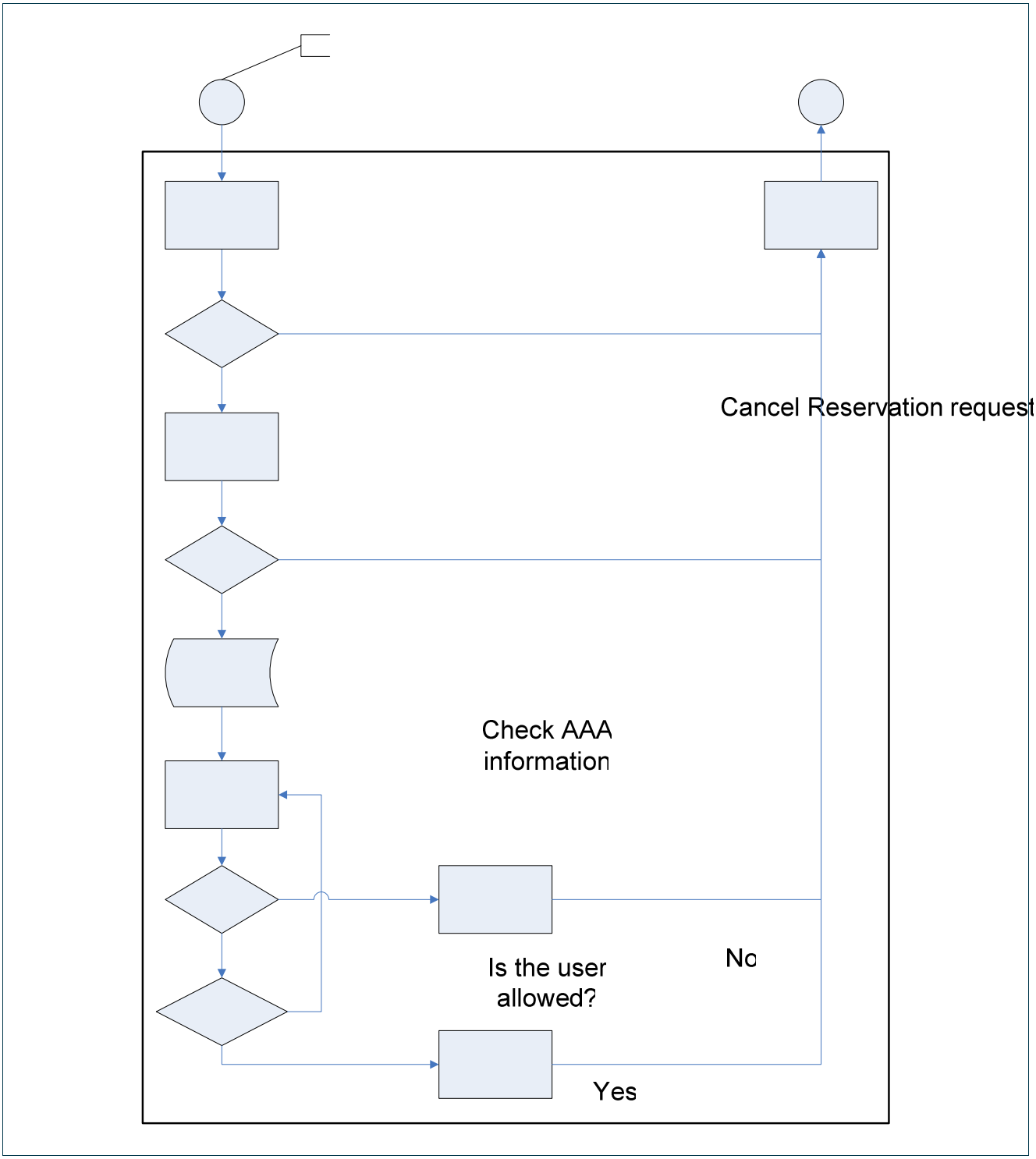

Figure 4.4: High level flow diagram for a Cancel Reservation operation.

4.5 Status Request

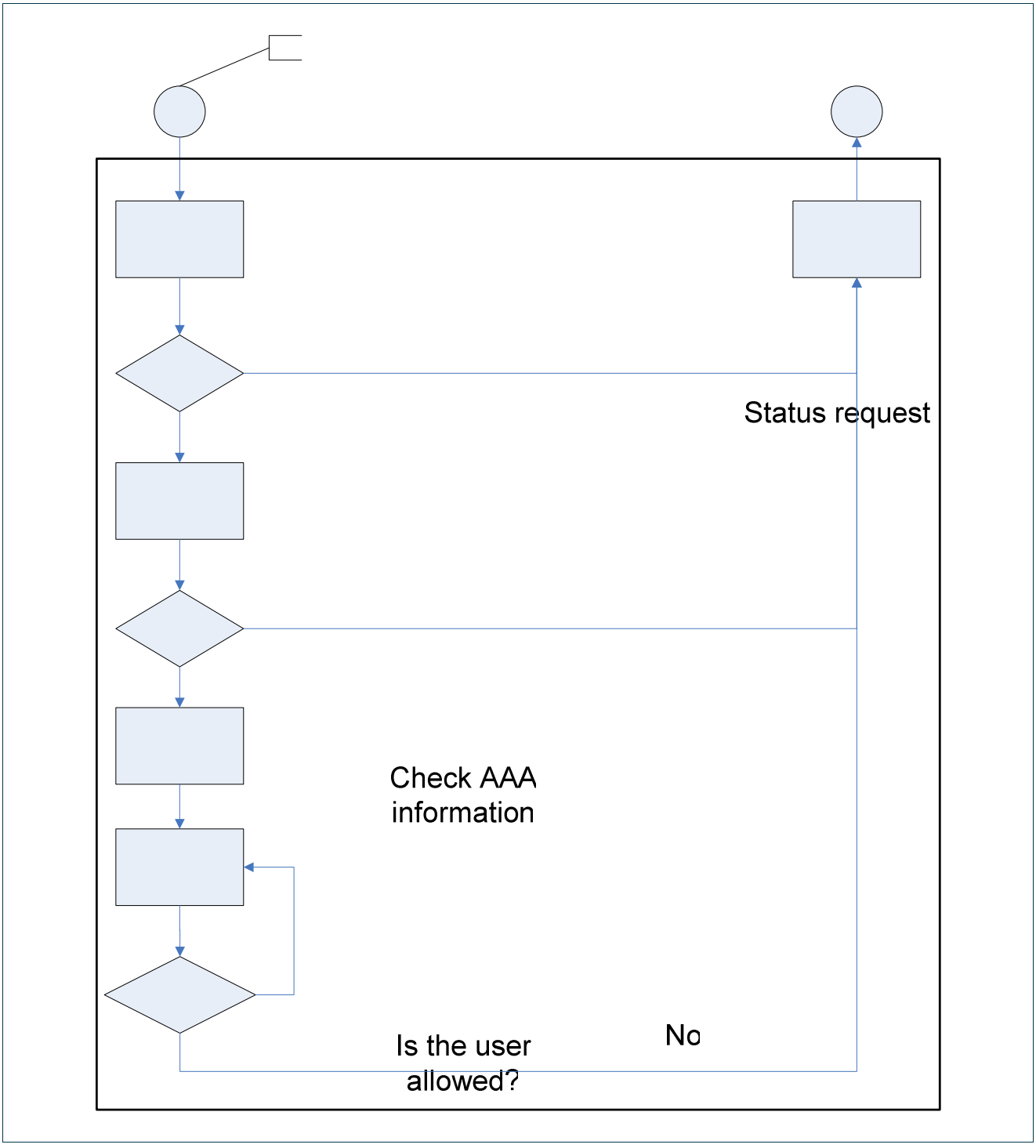

Figure 4.5: High level flow diagram for a Status Request operation.

4.6 Bind Request

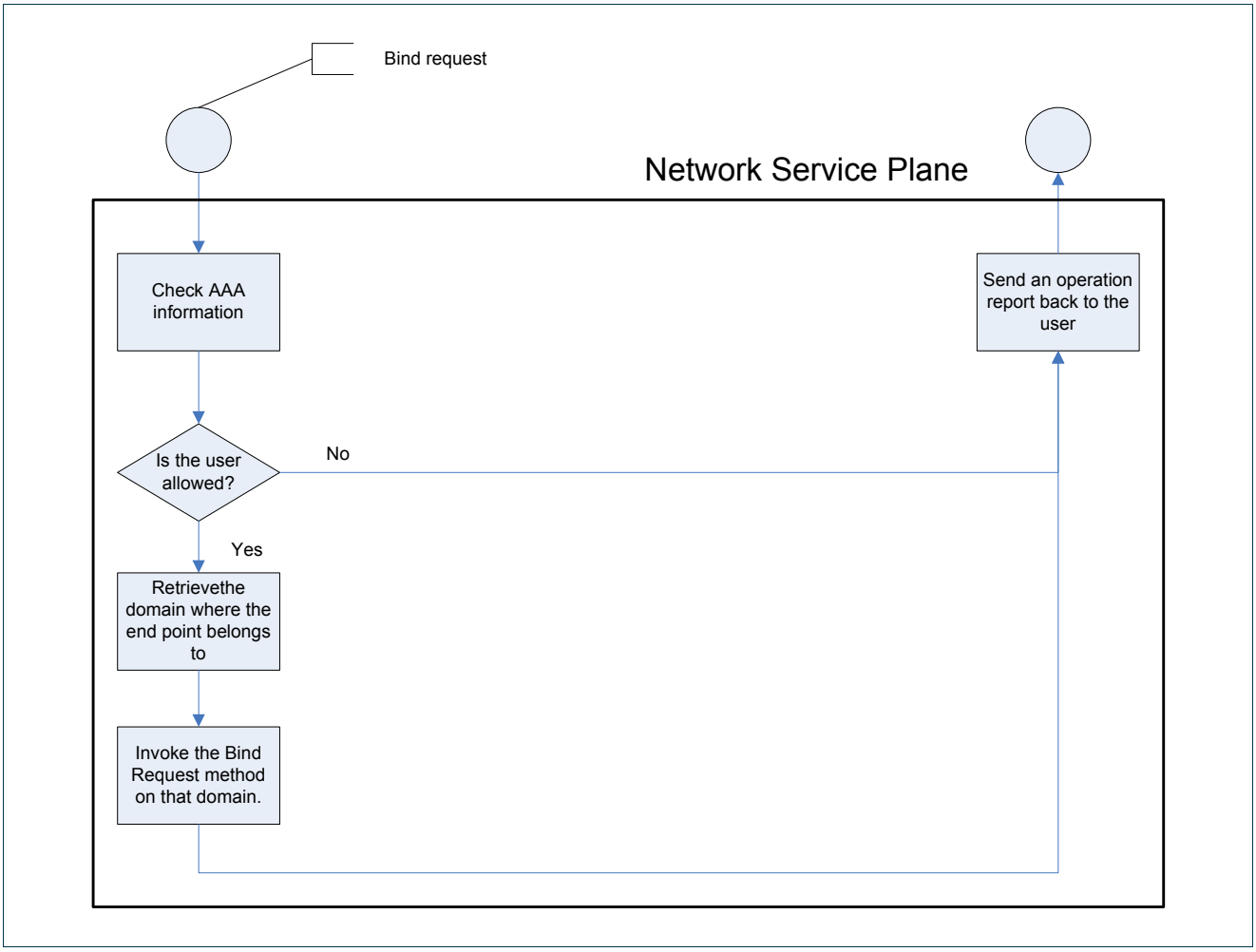

Figure 4.6: High level flow diagram for a Bind Request operation.

4.7 Activation Request

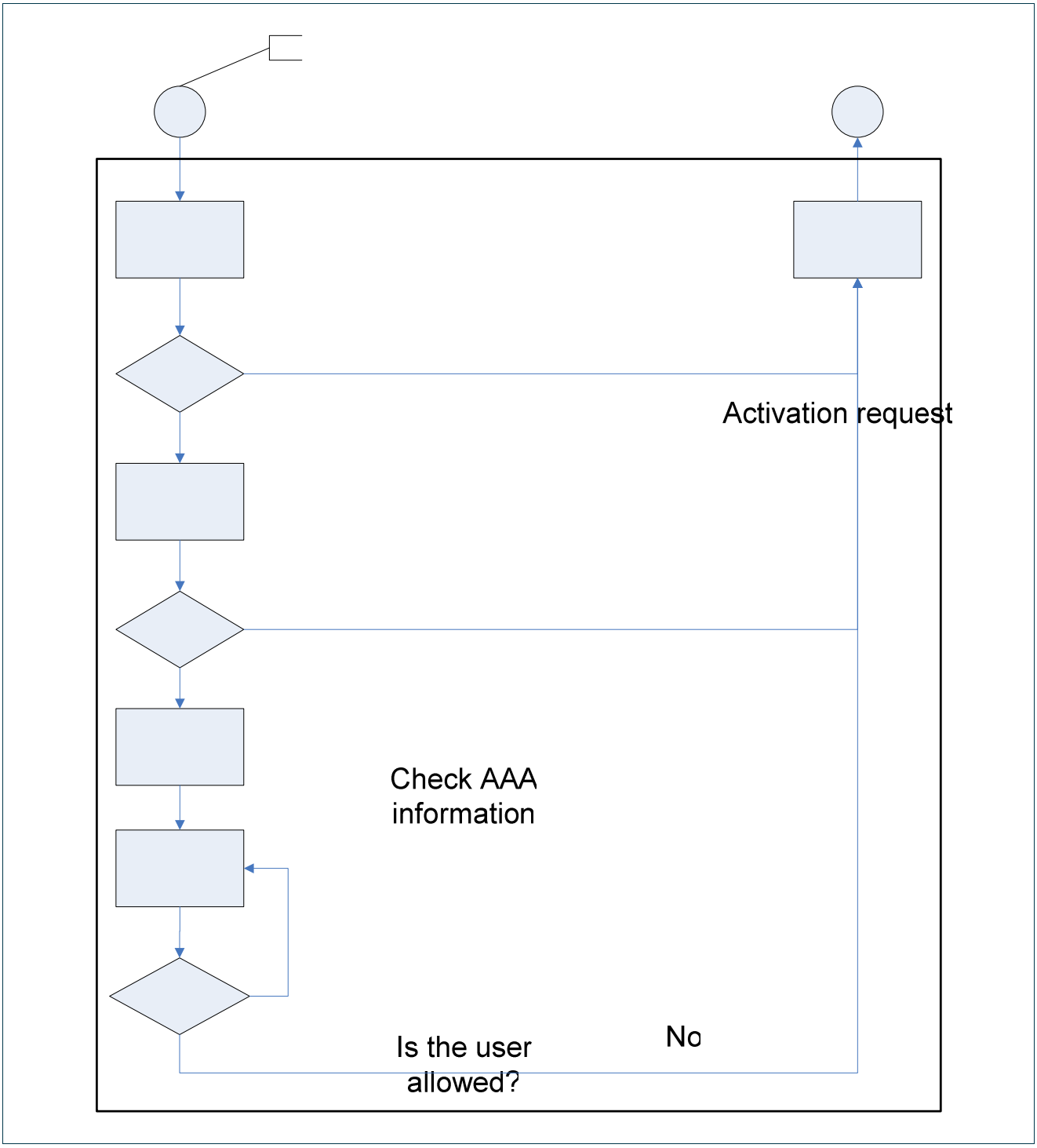

Figure 4.7: High level flow diagram for an Activation Request operation.

4.8 Complete Job

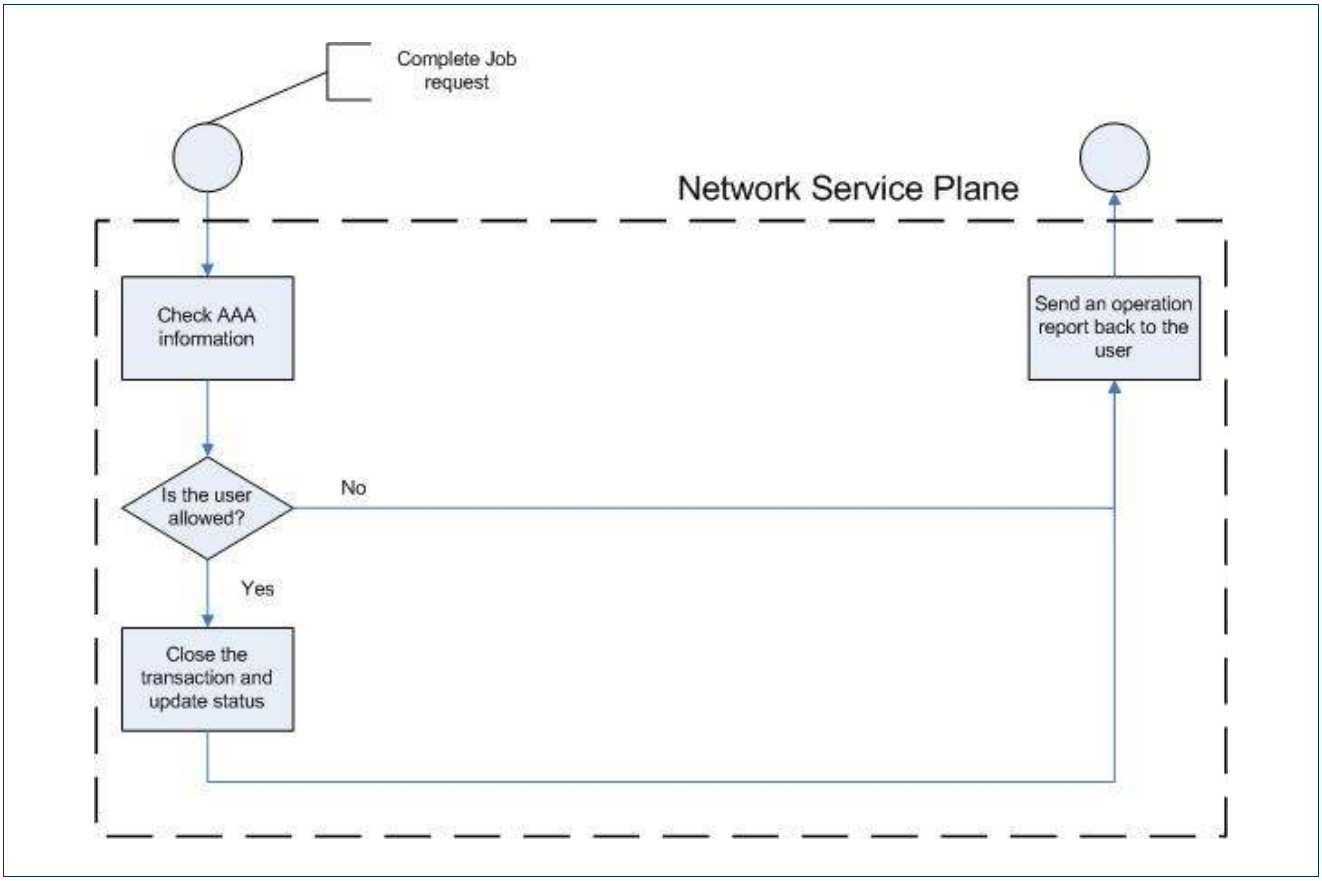

Figure 4.8: High level flow diagram for a Complete Job operation.

4.9 Cancel Job

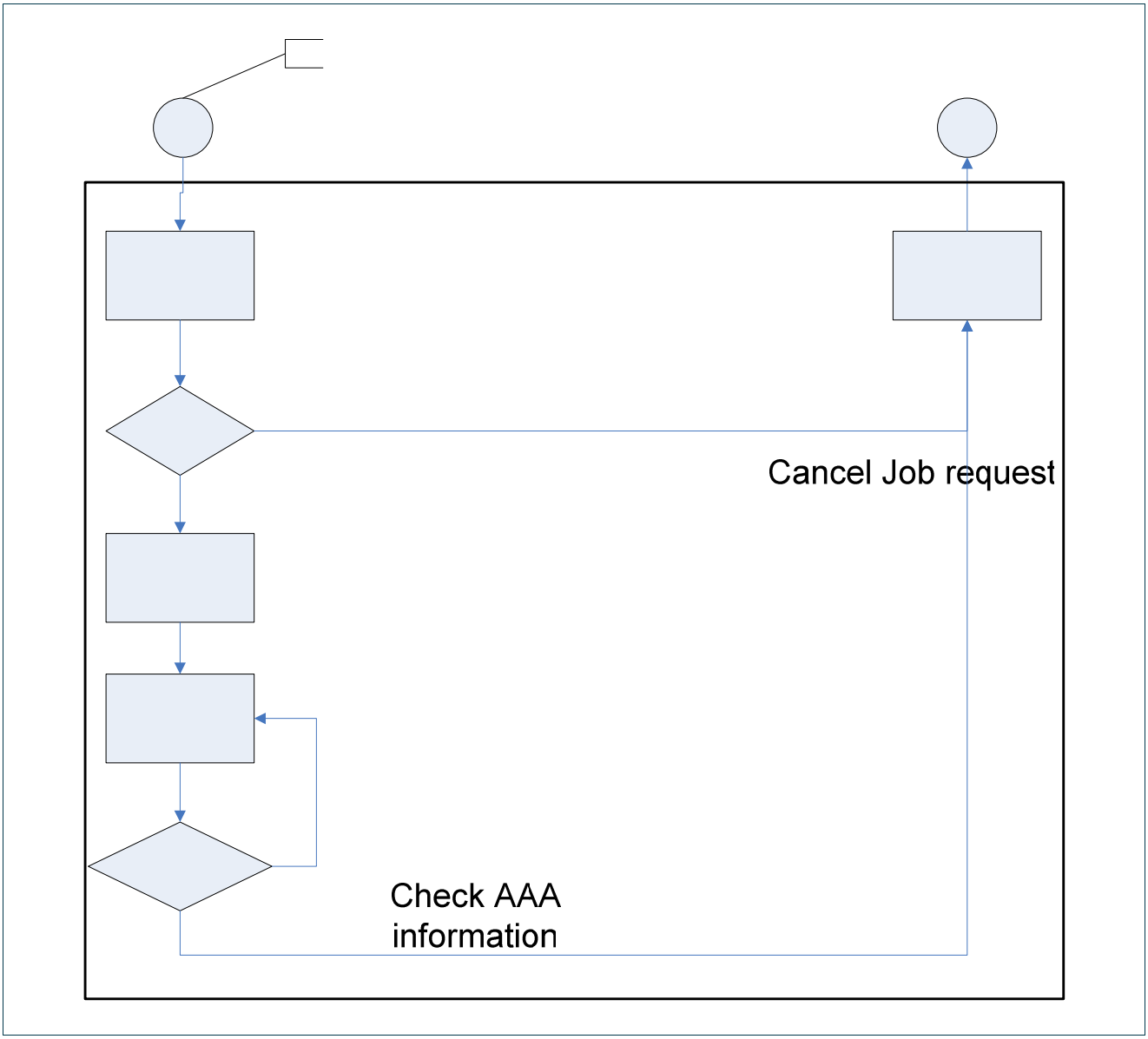

Figure 4.9: High level flow diagram for a Cancel Job operation.

Is the user allowed?

No

Retrieve Domains involved in the transaction

5 Conclusions

The requirements and specifications gathered during this deliverable will enhance the NRPS provided to the project by different partners, since these set of new interfaces specified allow their interoperability through the Network Service Plane (NSP). The northbound interface defined within the NSP will provide single-step on-demand services across multidomain networks, with advance reservation functionalities involving network resources.

This WP will develop a new Network Service Plane and a NRPS Adapter that will provide interoperability functionalities between the different NRPSs, the Grid middleware and a GMPLS Control Plane.

Another aim of the specifications defined is the interoperability with the JRA3 developments done within the GÉANT2 project, however and as interdomain interfaces from JRA3 are still under development, it is expected that during Phase 2 of the project the draft interface already proposed will be enhanced. Moreover, in order to better integrate WP1 developments with the G^{2}MPLS and the JRA3 distributed architecture, an enhanced Peer-to-Peer approach has been considered.

6 References

[ARGON]	- ARGON – Allocation and Reservation in Grid-enabled Optic Networks“ http://www.viola-testbed.de/content/index.php?id=reports - Project home page: http://www.viola-testbed.de - B2.3.2 (2006) "Program Documentation of the Reservation System User Interface" http://www.viola-testbed.de/content/index.php?id=reports
[DRAC]	http://www.nortel.com/solutions/optical/collateral/nn110181.pdf
[EnLIGHTened]	Project home page: http://enlightenedcomputing.org/
[GN2-JRA3]	“Deliverable DJ.3.3.1:GÉANT2 Bandwidth on Demand Framework and General Architecture” http://www.geant2.net/upload/pdf/GN2-05-208v7_DJ3-3-1_GEANT2_Initial_Bandwidth_on_Demand_Framework_and_Architecture.pdf
[G-Lambda]	Project home page: http://www.g-lambda.net/
[GT4]	Project home page: http://www.globus.org/toolkit/
[UCLPv2]	Project home page: http://www.uclp.ca/
[UniCore]	Project home page: http://www.unicore.eu/

7 Acronyms

[AAA]	Authorization, Authentication and Accounting
[ARGON]	Allocation and Reservation in Grid-enabled Optic Networks
[BoD]	Bandwidth on Demand
[CP]	Control Plane
[DB]	Data Base
[DM]	Domain Manager
[DRAC]	Dynamic Resource Allocation Controller
[GLIF]	Global Lambda Integrated Facility
[GMPLS]	Generalized Multi Protocol Label Switching
[IDM]	InterDomain Manager
[ID]	Identifier
[IP]	Internet Protocol
[I-NNI]	Interior NNI
[JRA]	Joint Research Activity
[LSA]	Link State Attribute
[LSP]	Label Switched Path
[MAC]	Medium Access Control
[MS]	MetaScheduler
[NNI]	Network-Network Interface
[NRPS]	Network Resource Provisioning System
[NSP]	Network Service Plane
[NSAP]	Network Service Access Point
[OSPF]	Open Shortest Path First
[QoS]	Quality of Service
[RSVP]	ReSerVation Protocol
[SDH]	Synchronous Digital Hierarchy
[TE]	Traffic Engineering
[TXN]	Transaction
[TNA]	Transport Network Address
[UCLPv2]	User Controlled LightPaths version 2
[URL]	Uniform Resource Locator
[VLAN]	Virtual Local Area Network
[VPLS]	Virtual Private LAN Services
[WP]	Work Package
[WS]	Web Service

Appendix A Technologies and addressing

A.1 The testbed

One of the objectives of the Phosphorus project is to create a heterogeneous European test-bed composed of several local test-beds. This requires some specific attention to the way the test-beds used in WP1 are interconnected.

The networks used in Phosphorus consist of backbone or test networks controlled by NRPSs, and local networks that are connected to NREN backbone networks that are in general not controlled by NRPSs. The Phosphorus project does not have dark fibre to link the different local test-beds. Therefore, GÉANT2 will provide the project dedicated GbEth links between the different local test-beds. A similar approach has been used with GLIF, which is mostly used for the overseas connectivity. As a result, from the WP1 perspective, there is no control over the configuration of the physical devices present on the links joining different local test-beds.

Regardless of the technology internally used by GÉANT2 or GLIF to provide the connectivity, all links will end in dedicated GbEth ports. In order to improve the flexibility of our scenario, a set of VLANs will be pre-provisioned on every interdomain link that will end on a Layer 2 switch at each edge of the link. This way, using single GbEth links between domains, it will be possible to emulate several dedicated links by multiplexing and demultiplexing VLANs into dedicated GbEth ports.

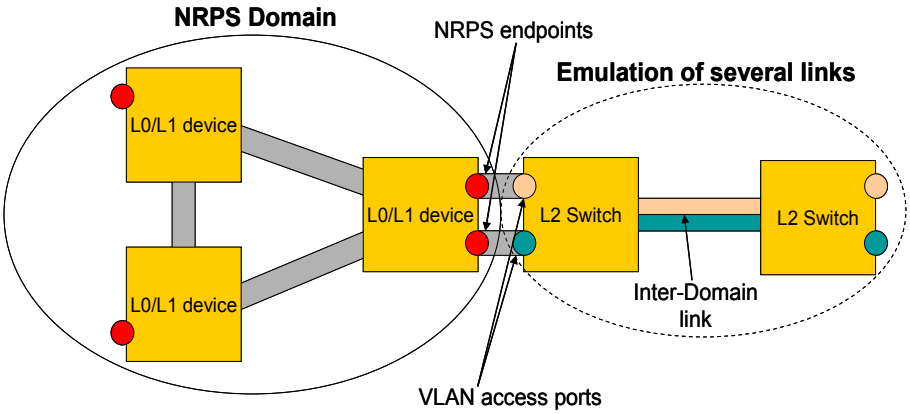

Figure A.1: Use of VLANs to emulate several links.

This means that any technology supported by the NRPS present in that domain can be used as far as dedicated GbEth ports are used for the interdomain links. This way, the system will be able to provide end to end connections starting or ending in any domain.

Another important issue to clarify is where the boundaries of the NRPS are, this is which physical devices are going to be configured by an NRPS and which ones are going to be manually configured or configured by a third party software. The system developed inside Phosphorus WP1 will only set up a set of end to end connections inside the NRPS domains. Therefore any configuration needed on the client side network is left to the user. As can be seen in the figure below the client packet networks are not configured by any NRPS.

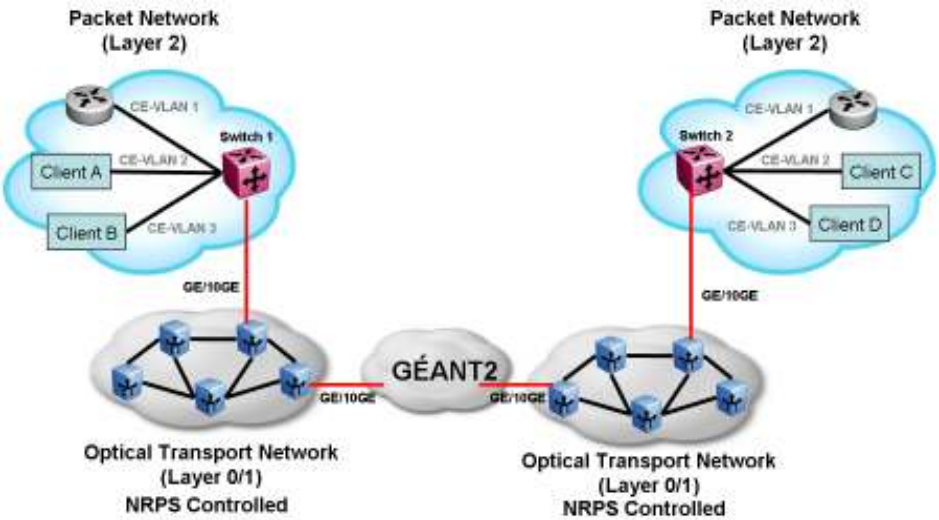

Figure A.2: Phosphorus network general architecture.

One of the improvements that is going to be studied and eventually implemented during Phase 2 of the project is the use of VLANs to emulate direct transparent connections between domains. This way, the NRPSs would be able to negotiate which VLAN is going to be used for a connection in the interdomain links. After the VLAN negotiation, the NRPS domains would proceed to configure the L2 equipment accordingly. It is important to note that not all domains will be able to support this feature due to the fact that not all NRPS are L2 capable and because the L2 equipment installed in the different domains might not be compatible with the current implementation of the NRPSs.

The networks interconnected in Phosphorus consist of backbone networks controlled by NRPSs and local networks that are connected to the backbone networks and that are in general not controlled by NRPSs.

The NRPSs can have different notions of their “NRPS endpoints” (or “provider endpoints”) and it is not the task of the Network Service Plane to interpret these endpoints, so NRPS endpoints are identified by strings at this layer.

Endpoints from the applications’ point of view are IP addresses or IP address ranges in local networks. Since for these “customer endpoints”, the specific IP addresses of the cluster nodes that will be assigned for a job may not be known until shortly before the start of a job, endpoints from the Middleware perspective are simply site names.

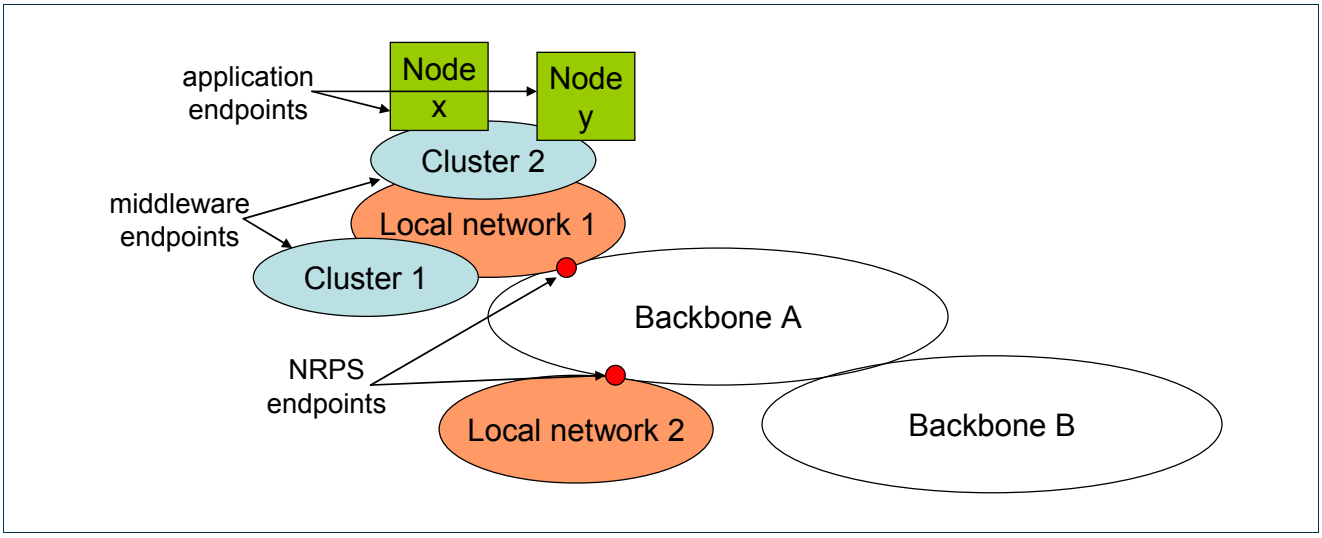

Figure A.3: Application, middleware, and NRPS endpoints

The mapping between the different types of endpoints (sketched in **Figure A.3**) has to be realized in the Network Service Plane. To achieve this, the endpoints configured via the administrative interface should be defined as follows:

- Endpoint name/identifier (e.g. “UniBonn-2”, used at the Application & Middleware layer).
- NRPS endpoint identifier pool (e.g. TNA addresses 192.168.37.225-192.168.37.254).

The Network Service Plane can look up the NRPS endpoint identifier corresponding to the endpoint names given in a request from the middleware and establish a connection using a specific TNA address (i.e. corresponding to the interface represented by the red bullet connected to “Local network 1” in **Figure A.3**).

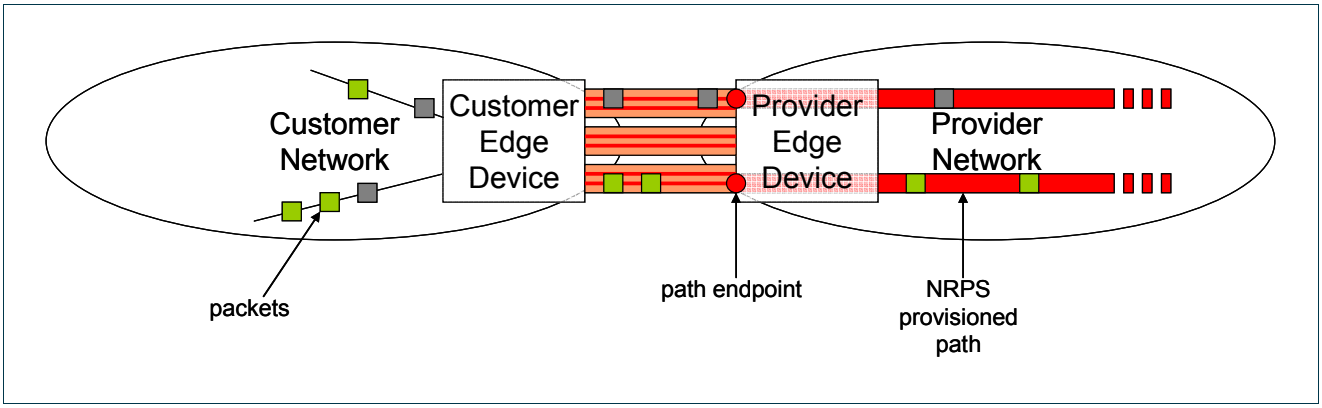

Figure A.4: Customer edge / provider edge scenario

In a provider edge / customer edge scenario (see **Figure A.4**), where the endpoint of a path provided by the NRPS is a logical channel on the customer side of a border device, and all traffic on this channel is put onto the corresponding path. Note that in general, there can be several logical channels on a single link, so that in this context, the information about an endpoint as defined in Section 2.2.1.3 might have to be extended to identify such a path endpoint: E.g., if provider edge device is a pure optical cross-connect, then a logical channel on a physical link would be a specific wavelength, or if the device is an SDH cross-connect, a logical channel would be a specific container.

The “bind” operation is used to create the “binding” between specific IP addresses and a specific path endpoint, i.e. the mapping between customer and provider endpoints. The binding has to be configured in the local network and is triggered by a Bind Request (cf. Section 3.3.5) that is handed either to the NRPS or to the corresponding local network by the NRPS Adapter. This allows for the flexibility to have NRPSs that take care of the configuration up to the customer endpoints themselves as well as NRPSs that do not. Therefore, the NRPS Adapter may need to know where such a local configuration service is located for each of the endpoints.