034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.2.7

Grid-GMPLS network interfaces specification

Due date of deliverable: 2008-02-29
Actual submission date: 2008-02-29
Document code: Phosphorus-WP2-D2.7

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable: **University of Essex (UESSEX)**

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

Abstract

This report details the Grid-GMPLS G.OUNI, G.E-NNI and G.I-NNI Network Interfaces that enable the concept of Grid Network Services (GNS). The document also describes the PHOSPHORUS control plane architectures (overlay and integrated) with regards to interfaces and an overview of the procedures and services supported on each of them in terms of Grid and network services, Grid-based semantics, supported protocols and specific protocol extensions needed.

List of Contributors

George Zervas	UESSEX	Artur Binczewski	PSNC
Eduard Escalona	UESSEX	Bartosz Belter	PSNC
Dimitra Simeonidou	UESSEX	Radoslaw Krzywania	PSNC
Reza Nejabati	UESSEX	Damian Parniewicz	PSNC
Nicola Ciulli	NXW	Maciej Strozyk	PSNC
Gino Carrozzo	NXW	Anna Tzanakaki	AIT
Francesco Salvestrini	NXW	Georgios Markidis	AIT
Giodi Giorgi	NXW	Jaafar Elmirghani	ULEEDS
Oliver Wäldrich	FHG-SCAI	Taisir El Gorashi	ULEEDS

Table of Contents

List of Figures

List of Tables

List of Listings

0 Executive Summary

This document is the final specification document of the Grid-GMPLS Network Interfaces (i.e. G-OUNI, G.E-NNI, G.I-NNI). It provides all information included on the preliminary document (submitted on M12) that was detailing G.OUNI in terms of Grid and network services, Grid-based semantics, supported protocols and specific protocol extensions. Apart from some limited extensions to the G.OUNI section this final specification document reports also on G.E-NNI and G.I-NNI interfaces.

In Section 1 the objectives of this document on G^{2}MPLS Control Plane Network Interfaces are stated, as well as the scope of the deliverable in WP2 framework.

In Section 2 the terminology and abbreviations relevant to the G^{2}MPLS Network Interfaces are presented.

Section 3 provides a brief summary of PHOSPHORUS G^{2}MPLS Control Plane architectures (Overlay and Integrated) with focus on network interfaces.

In Section 4 Grid Optical User Network Interface (G.OUNI) functionalities, services, supported protocols and related extensions are presented. The role of G.OUNI in Grid Network environment is described with main reference to the OGF-G.OUNI informational draft that the team is discussing in the framework of OGF GHPN RG. Description of Grid and network services as well as the protocol extensions required to realise them are reported.

In Section 7 Grid External Network-Network Interface (G.E-NNI) functionalities, services, protocol extensions and message flows are presented. G.E-NNI reference point is defined to exist between control domains. G.E-NNI is based on E-NNI and is been extended to support both network services as well as Grid network services.

Section 6 abstractly reports on Grid Internal Network-Network Interface (G.I-NNI), since the main services supported have been covered on previous sections.

Section 7 finally concludes the deliverable.

1 Objectives and Scope

This is the final release of the Grid-GMPLS network interfaces specification document. In provides the detailed description of all the G^{2}MPLS reference points, the G.OUNI interface reported on the preliminary version (M12), together with G.E-NNI and G.I-NNI.

The advent of Grid Computing and Optical Networks has necessitated the development of interoperable procedures for requesting and establishing dynamic network (via GMPLS Control Plane) and non-network services (via Grid Middleware) between clients, applications and computational resources (e.g. CPUs, storage, etc.), all connected by the transport network. The development of such procedures requires the definition of interfaces at the G^{2}MPLS NCP level able to support connectivity services through the multi-domain transport network for Grid end-points, signalling protocols used to invoke the Grid and network services, and auto-discovery procedures to aid signalling. All these procedures are required to facilitate on demand as well as in-advance Grid and network services over G^{2}MPLS.

The document here describes the PHOSPHORUS control plane architectures (Overlay and Integrated) with regards to G^{2}MPLS reference points. A high level description of the roles, procedures, and services supported on each of the interfaces is provided. Starting with the G.OUNI, a description of its role to the broader Grid Network environment and how this is reflected on an Open Grid Forum Standardisation Informational Draft is provided. The need for a generic G.OUNI interface to serve any type of Grid Middleware and any type of Network provisioning system is also reported.

The aim of this document is to provide a description of Grid and network services that should be supported over G.OUNI, G.E-NNI and G.I-NNI and report on the protocol extensions required to realise the G^{2}MPLS Control Plane. A high level description of Grid and network services offered over G.OUNI, G.E-NNI and G.I-NNI is provided. During this stage, a hierarchical description of existing services is mentioned as baseline and prerequisites to comply with already standardised interfaces (e.g. [OIF-UNI1.0R2-COMM], [OIF-E-NNI-Sig-1.0],

[OIF-E-NNI-Rtr-1.0]) towards network and Grid extensions to support the PHOSPHORUS overlay and integrated approach accordingly. Following the structure of the OIF-UNI reports, G.OUNI service invocation configurations, signalling configuration, addressing, Grid Neighbour discovery and Grid Service Discovery is provided. The role of atomic and complex G.OUNI services required supporting resource reservation and co-allocation accordingly are identified based on WS-Agreement specifications. This identifies the messages that have to be translated to G^{2}MPLS protocols for the Control Plane to support such services. A first attempt of G.OUNI Policy and Security information is provided in terms of a proposed Grid AAA-AuthZ solution for policy based access control and enforcement at G.OUNI. The document provides the G.OUNI abstract message description for signalling, routing and discovery procedures. The protocol extension section begins with RSVP. The main scope here is to identify the RSVP extensions necessary to satisfy requirements for G.OUNI signalling mechanisms. As a base for further Grid extensions, according to the deliverable [G2MPLS-ARCH], the UNI-RSVP-TE signalling protocol was chosen [OIF-UNI1.0R2-COMM, OIF-UNI1.0R2-RSVP]. To make transformation from JSDL elements into RSVP objects possible data types of all mandatory elements are described and mapped. The document, then, abstractly reports on LMP as an alternative approach to support Grid resource capabilities and availabilities but mainly focuses on OSPF extensions as the main PHOSPHORUS development method to represent Grid resources on G^{2}MPLS control plane. Grid resource description extension can be based on GLUE schema [GLUE] that is abstract modelling for Grid site resources [G2MPLS-ARCH]. Thus, Grid resource description objects and properties as well as Grid resource availability calendar based on GLUE schema is provided. Then, Grid related OSPF extensions can be applied to GMPLS routing protocol using new types of the OSPF Opaque LSA [IETF-RFC2370]. Low level description of protocol extensions are provided on [G2MPLS-EXT].

In order to support deployment of a G^{2}MPLS control plane into heterogeneous Grid Network environment, it is then essential to support the concept of control domains, and in particular, the specification of the signalling and routing information exchanged between such domains as defined in [OIF-E-NNI-Sig-1.0, OIF-E-NNI-Rtr-1.0]. However, extensions to standardised protocols and procedures are defined and described in order to support GNS services among different domains. The G.E-NNI reference point is defined to exist between G^{2}MPLS control domains. G.E-NNI is instantiated by signalling and routing protocols and as such it becomes a G.E-NNI signal and routing interface. The signalling and routing extensions to support GNS services are reported through detailed description of protocol extensions and message flows. The compatibility with G.OUNI interface is also identified. Finally, G.I-NNI is abstractly described since all services and functions supported by it are covered by the G.OUNI and G.E-NNI sections.

2 Terminology

Keyword	Source	Definition
Grid middleware (MW)	**OGF**	Grid technology (a.k.a. middleware) is employed to facilitate formalizing and complying with the Grid context associated with an application execution. Middleware is computer software that connects software components or applications. It is used most often to support complex, distributed applications. It includes web servers, application servers, content management systems, and similar tools that support application development and delivery. Middleware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture.
GLUE schema	**[GLUE]**	An information model that provides a description of core Grid resources at the conceptual level by abstracting real world resources into constructs that can be represented in computer systems (e.g. objects, properties, behaviour, and relationships). The GLUE schema is not tied to any particular implementation and can be profitably used to exchange information among different knowledge domains.
Job Submission Description Language (JSDL)	**[OGF-GFD81], [JSDL]**	A language for describing job submissions, including details of their required execution environments. See https://forge.gridforum.org/projects/jsdl-wg for more information.

Keyword	Source	Definition
Grid Network Service (GNS)	**[OGF-GNS]**	A network service (e.g. management of QoS classes, policy enforcement points, topology data, network usage metrics, AAA, etc.) with roles and/or interfaces that are deemed to be specific to a Grid infrastructure is a Grid Network Service [OGF-GNS]. Network Services belong to the class of the base resources of the OGSA Architecture [OGSA]. Base resources are those physical or logical resources that are supported entities out of the context of the OGSA. Examples of such entities include CPUs and memory in the physical case and licenses, contents and OS processes in the logical case.
Grid Resource	**[OGF-GFD81]**	In OGSA, a resource is an entity that is useful in a Grid environment. The term usually encompasses entities that are pooled (e.g. hosts, software licenses, IP addresses) or that provide a given capacity (e.g. disks, networks, memory, databases). However, entities such as processes, print jobs, database query results and virtual organizations may also be represented and handled as resources.
Grid service	**[OGF-GFD81]**	In general use, a Grid service is a Web service that is designed to operate in a Grid environment, and meets the requirements of the Grid(s) in which it participates.
Job	**[OGF-GFD81]**	A user-defined task that is scheduled to be carried out by an execution subsystem. In OGSA-EMS, a job is modelled as a manageable resource, has an endpoint reference, and is managed by a job manager.
Network Control Plane (NCP)	**[ASON-ARCH, ASON-DEF]**	The network Control Plane performs the call control and connection control functions. Through signalling, the Control Plane sets up and releases connections, and may restore a connection in case of a failure. The Control Plane also performs other functions in support of call and connection control, such as routing information dissemination.

Keyword	Source	Definition
TE-link	**[IETF-RFC4201, IETF-RFC4202]**	A traffic engineering (TE) link is a logical construct that represents a way to group/map information about certain physical resources (and their properties) that interconnect LSRs with information that is used by Constrained SPF (for the purpose of path computation) and by GMPLS signalling.
Vsite	**UNICORE architecture**	A Vsite identifies a particular set of Grid resources at a UNICORE site (Usite) and is controlled by a Network Job Supervisor (NJS). Vsites may consist of a single supercomputer or a cluster. If more than one resource is operated by an organization there can be one Vsite for each resource inside one Usite.
Authentication Authorization Accounting	**WP4 D4.1**	A term used to refer to a framework for intelligently controlling access to computer resources, enforcing policies, auditing usage, and providing the information necessary to bill for services. These combined functions are considered important for effective network management and security. Authentication is the process of identifying a user or an access subject, based on identity credentials which examples are username and password, digital certificates, one-time-tokens, etc. Authorization refers to the confirmation that a user/subject who is requesting services is a valid user of the resources or services requested. Accounting is the process of keeping track of a user's activity while accessing the resources or services.
WS-Agreement	**[OGF-GFD.107]**	Web Services Agreement Specification (WS-Agreement), a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties.

2.1 Abbreviations

AAA – Authorization Authentication and Accounting

ASON – Automatically switched optical network

ASTN – Automatic Switched Transport Network

ATM – Asynchronous Transfer Mode

COPS – Common Open Policy Service

CP – Control Plane

CRP – Complex Resource Provisioning

G.OUNI – Grid Optical User Network Interface

G.OUNI-C – G.OUNI - Client

G.OUNI-N – G.OUNI - Network

G^2MPLS – Grid-enabled GMPLS

GCD – Grid Calendar Database

GLUE – Grid Laboratory Uniform Environment

GMPLS – Generalized Multiprotocol Label Switching

GNS – Grid Network Services

GRD – Grid Resource Database

GSI – Grid Security Infrastructure

HO – Home Organisation

IETF – Internet Engineering Task Force

IPCC – IP Control Channel

ITU-T – International Telecommunication Union - Telecommunication Standardization Sector

JSDL – Job Submission Description Language

LMP – Link Management Protocol

NCP – Network Control Plane

OGF – Open Grid Forum

OHRM – Obligation Handling Reference Model

OIF – Optical Internetworking Forum

OSPF – Open shortest Path First

PAP – Policy Authority Point

PBAC – Policy Based Access Control

PDP – Policy Decision Point

PEP – Policy Enforcement Point

QoS – Quality of Service

RSVP – Resource Reservation Protocol

SDH – Synchronous Digital Hierarchy

SDT – Service Description Term

SLA – Service Level Agreement

SLO – Service Level Objective

SONET – Synchronous Optical Networking

STS – SONET Basic Transmission Rate

TBN – Token Based Networking

TED – Traffic Engineering Database

TLV – Tag Length Value

TNE – Transport Network Element

TVS – Token Validation Service

VC – Virtual Concatenation

VL – Virtual Laboratory

VO – Virtual Organisation

WS – Web Service

WSRF – Web Service Description Framework

XACML – eXtensible Access Control Markup Language

3 PHOSPHORUS G^{2}MPLS Control Plane: Architecture and Interfaces

G^{2}MPLS Control Plane sets analogous reference points (Figure 3-1) with respect to the ASON/GMPLS, with evolved network interfaces capable of managing and advertising the semantic of both Grid and network resources. Network interfaces in the scope of this document are:

- **G.OUNI**: Interface between Grid site/user and the G^{2}MPLS NCP,
- **G.I-NNI**: Interface between G^{2}MPLS adjacent nodes,
- **G.E-NNI**: Interface between different NCP domains.

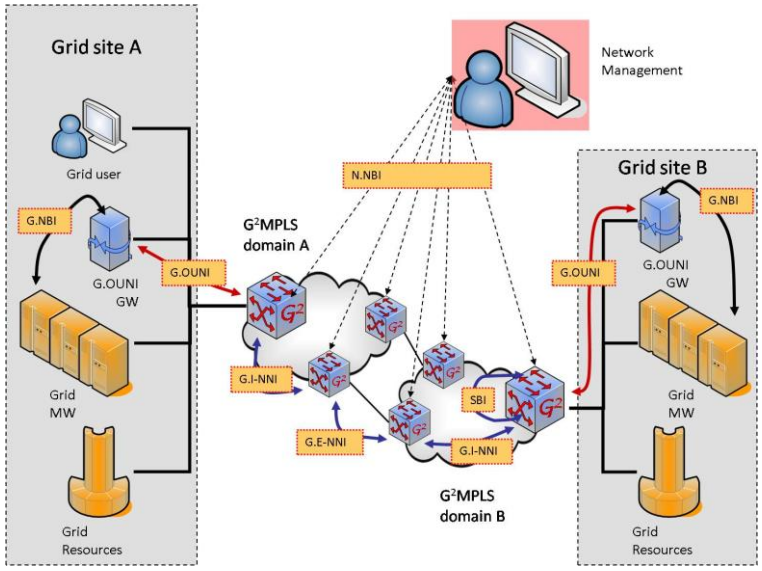

Figure 3-1: G²MPLS Network reference points.

The roles, activities, procedures and services of these interfaces depend on the network layering models adopted for the G²MPLS Control Plane. In G²MPLS architecture, two control plane models have been identified as mentioned in companion documents, deliverables [G2MPLS-ARCH] and [G2MPLS-DEP]. They are:

- G²MPLS Overlay model
- G²MPLS Integrated model

These models refer principally to the positioning between the Grid Service Layer and the Network Control Plane and require different capabilities of the G²MPLS NCP.

In G²MPLS Overlay model, the Grid layer has both Grid and network routing knowledge (ref. Figure 3-2). G²MPLS provides automatic configuration just for the network service part; moreover, it acts as an information bearer of network and Grid resources. Therefore, in this context G²MPLS basically implements the ASON Switched Connections (SC) and operates as a slave of the Grid layer (the Grid scheduler, in particular), which is the overall responsible for initiation and coordination of the (advance) reservation process through the participating Grid sites and the network. In G²MPLS Overlay, the role of the network interfaces is mainly scoped to the Network Service creation, but they also piggyback opaquely end-to-end Grid information concerning resource availabilities, site capabilities (routing) and job description data (signalling).

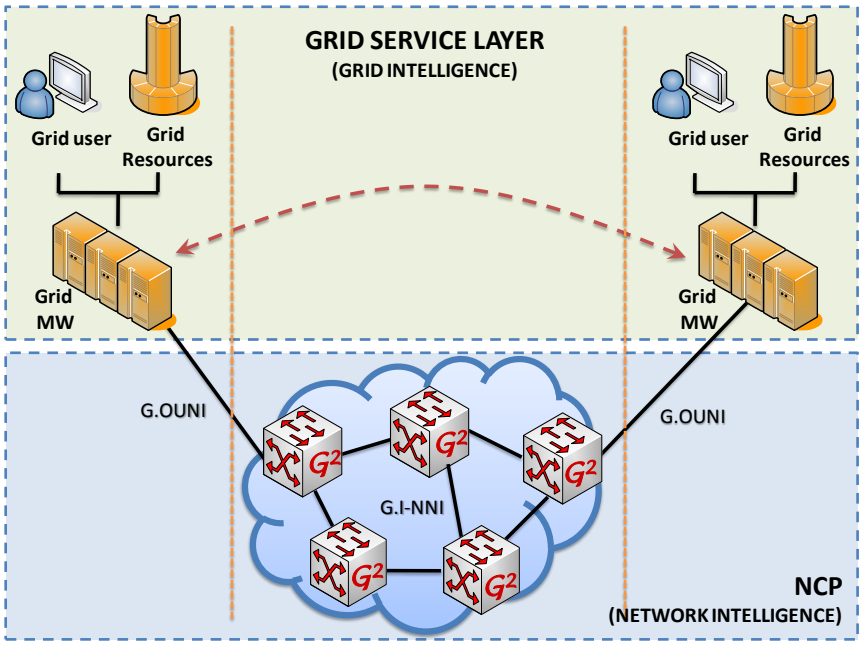

Figure 3-2: G^{2}MPLS Overlay Model

In the G^{2}MPLS Integrated model, most of the co-allocation functionalities are moved to the Network Control Plane (ref. Figure 3-3). G^{2}MPLS is responsible for scheduling and configuring all the job parts, those related to the Grid sites and those related to the network. The Grid scheduler functionality is still needed to coordinate workflow services, because G^{2}MPLS NCP is capable of managing just the workflow elementary unit, i.e. the Grid job. In this model, the role of the network interfaces is scoped to Grid Network Service creation, which implies that Grid information concerning Grid resource availabilities (routing) and job description data (signalling) become transparent at those interfaces in which a decision process needs to be provided: these are the G.OUNI, by which a G^{2}MPLS is entered, and the G.E-NNI, by which the border between domains is traversed.

Abstract descriptions of the G^{2}MPLS Network Interfaces have already been defined in deliverables [G2MPLS-ARCH] and [G2MPLS-DEP]. The next sections of this deliverable basically deal with the extensions required for enabling Grid awareness over G.OUNI interface.

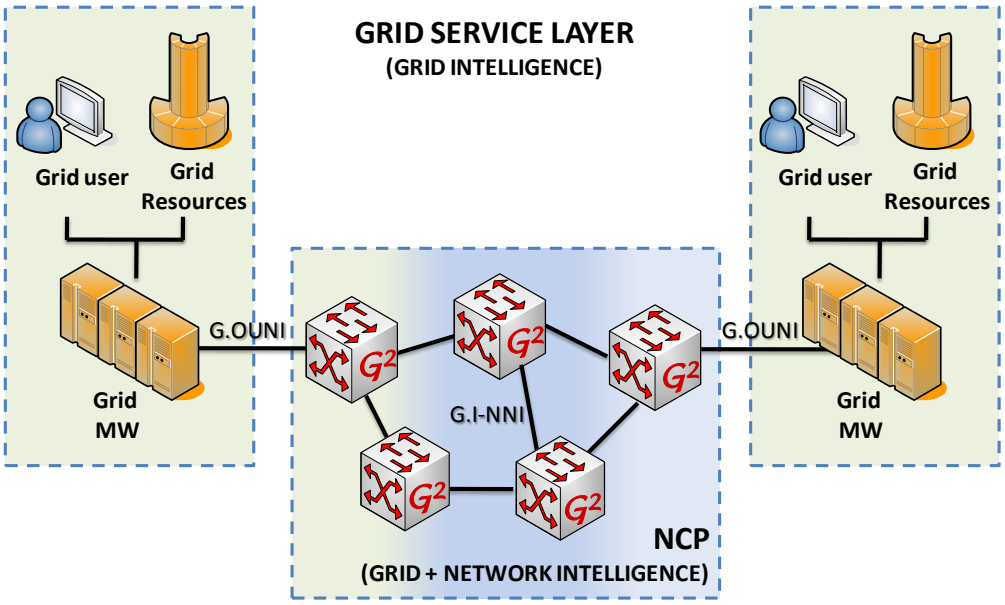

Figure 3-3: G^{2}MPLS Integrated Model

4 Grid Optical User Network Interface (G.OUNI) interface and functionalities

This chapter presents the G.OUNI interface that interconnects the Grid layer to Grid-enabled GMPLS (G^2MPLS) at the Control Plane. Different types of services such as resource discovery, characterization, allocation, and management services (middleware and connectivity services) need to be supported. All of the above issues must be addressed by protocols and mechanisms that build an architectural model. This chapter provides definition, architecture and protocol extensions of the G.OUNI to support PHOSPHORUS Grid Network architectures described in [G2MPLS-ARCH]. This work introduced an informational draft currently under discussion in the Open Grid Forum (OGF) for a generalised Grid User Network Interface (G.OUNI) able to serve any Grid Network architectural model.

4.1 G.OUNI definition and role in Grid Network Environment

The technological evolution shaped the promise for a new technological era and an emergent roadmap to the Grid Networking infrastructure. This drives the need of a number of distinct layered architectural models across geographical organizational boundaries, heterogeneous environments with different policies, service provisioning systems, control and transport planes as well as security standards. In order to take advantage of Grid resources (computers, instruments, sensors, data resources), network resources (bandwidth, wavelengths, switches, ports), as well as storage resources, in a joint and seamless way, Grid and network provisioning systems must be interfaced via a generic G.OUNI. The PHOSPHORUS project has taken the initiative to create an informational G.OUNI draft on Open Grid Forum (OGF) under the Grid High Performance Networking (GHPN) research group. This draft aims to provide a standard set of functionalities of a Grid Optical User Network Interface (G.OUNI) to serve distributed heterogeneous dynamic Grid network environments. The G.OUNI will act as a Grid network service control interface between Grid end points (e.g. users, applications,

resources) as well as middleware with network service provisioning systems (i.e. GMPLS, G^{2}MPLS, NRPS, OBS) and thus broaden and generalise the PHOSPHORUS G.OUNI approach. The goal of this work is to describe G.OUNI requirements driven from GNS use cases (i.e. PHOSPHORUS, Enlightened, G-Lambda, etc.) and in turn provide specific G.OUNI capabilities to meet these requirements. The various functionalities required to support Grid services and applications will also be determined. Key to the realization of this standardization vision is to embrace the OGF and OASIS standards and provide extensions to already defined UNI standards (OIF, IETF, ITU).

In the OGF draft point of view, Grid Optical User Network Interface (G.OUNI) comprises a number of procedures to facilitate on demand as well as in-advance access to Grid services/resources by interfacing Grid end points and any Grid middleware with any type of network resource provisioning system. G.OUNI is conceived also to interoperate with current GMPLS transport network in a limited downgraded configuration, by acting as a standard O-UNI. Interoperable procedures between Grid users/resources (e.g. storage, processor, memory) and optical network for agreement negotiation and Grid service activation have to be developed. The G.OUNI reference point described in OGF-G.OUNI draft acts as the interface between Grid End Points and the Grid Network Service Provisioning Systems as showed in Figure 4-1.

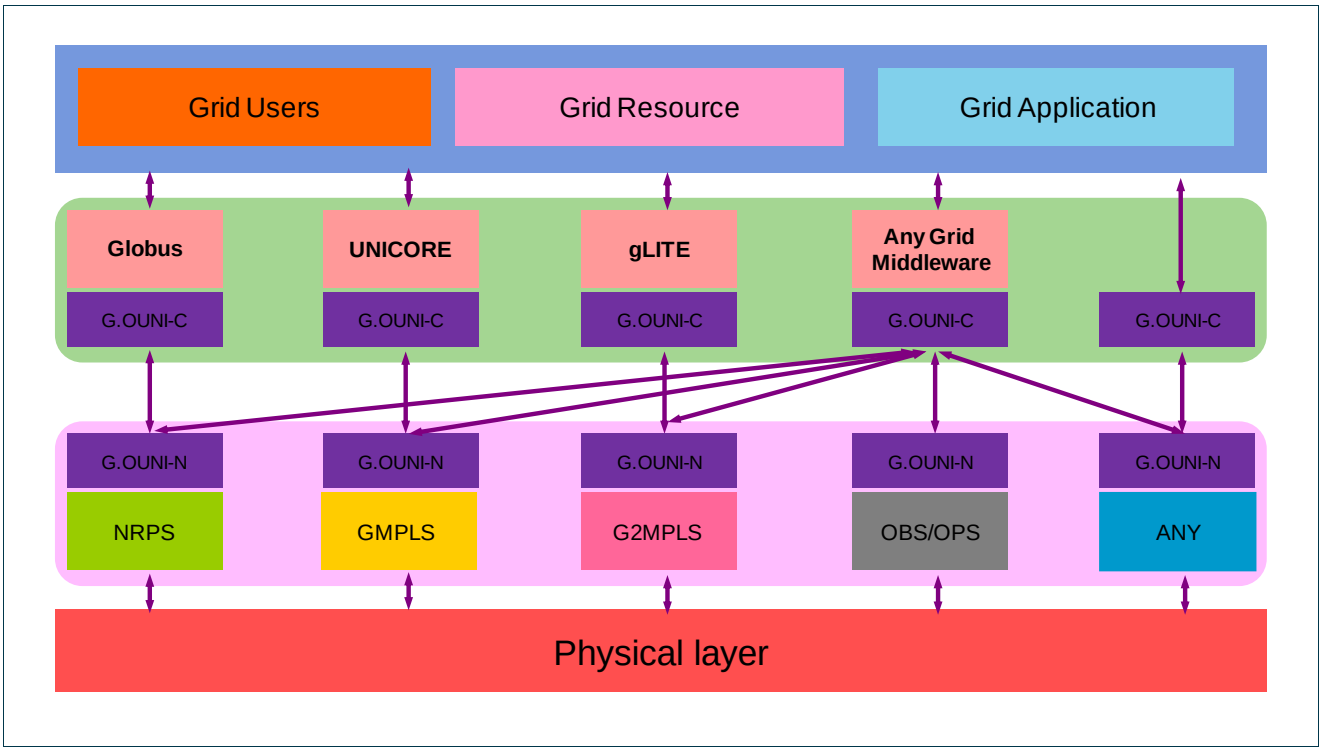

Figure 4-1: Grid User Network Interface with Grid End points as well as Grid middleware with Network Provisioning Systems [OGF-G.OUNI]

The rest of the chapter describe specific functionalities, services and protocol extensions that will be supported by PHOSPHORUS G.OUNI to interface specific Grid middleware (e.g. UNICORE, Globus) deployed in PHOSPHORUS and G^{2}MPLS. Furthermore, in addition to high-level description of protocol extensions (such as RSVP, OSPF, LMP), low-level detailed description of them is provided on [G2MPLS-EXT].

4.2 G.OUNI activities and roles in PHOSPHORUS

The advent of Grid Computing and Optical Networks has necessitated the development of interoperable procedures for requesting and establishing dynamic network (via GMPLS Control Plane) and non-network services (via Grid Middleware) between clients, applications and computational resources (e.g. CPUs, storage, etc.), all connected by the transport network. The development of such procedures requires the definition of an enhanced User to Network interface at the G^{2}MPLS NCP level able to support:

- connectivity services through the transport network for Grid end-points
- signalling protocols used to invoke the Grid and network services
- auto-discovery procedures to aid signalling.

All these procedures are required to facilitate on demand as well as in-advance Grid and network services over G^{2}MPLS. The above mentioned description constitutes the Grid Optical User Network Interface (G.OUNI).

G.OUNI activities listed and described below represent the functionalities supported by G.OUNI in a very abstract way.

Network Activities

1. Connection establishment
2. Connection deletion
3. Status exchange
4. Network auto-discovery
5. Use (traffic)

Grid Network Activities

6. GNS establishment can have subcategories based on type of service (e.g. publish, discovery, reservation, co-allocation of resources)
7. GNS deletion
8. GNS status exchange
9. GNS auto-discovery

10. Grid Use

The specification of the first five activities under the Network service cluster (1-5) is provided by OIF UNI 2.0 standardization document [OIF-UNI2.0-COMM]. The result of GNS establishment is the creation of Grid Service such as publication, reservation or co-allocation of resources to/from G^{2}MPLS NCP. The result of GNS deletion is the destruction removal of a service. The result of the GNS status exchange is the discovery of Grid service status. The GNS auto-discovery between Grid sites (client/application/resources) and the network consists in the publication of the services available from both network and Grid sites and the transaction of Grid and/or Network topology among Grid and Network layers.

For each activity, there is Grid Layer – Network Layer role. In all cases, G.OUNI-N agent provides the Network Layer role and G.OUNI-C the Grid Layer role. The first five activities are specified in OIF UNI 2.0 as User – Server roles (instead of Grid Layer – Network Layer roles). G^{2}MPLS enhances Activity 1 by implementing advance reservation as an extra feature of connection establishment (Table 4-1).

Network Activities				
1	**2**	**3**	**4**	**5**
Grid Layer	Grid Layer	Grid Layer	Grid Layer	Grid Layer
⇕	⇕	⇕	⇕	⇕
Connection Establishment	Connection Deletion	Status Exchange	Auto Discovery	Use
⇕	⇕	⇕	⇕	⇕
Network Layer	Network Layer	Network Layer	Network Layer	Network Layer

Table 4-1: G.OUNI Network Activities

For the activities starting from 6 to 10, client and server have two different roles based on the architectural model used (overlay or integrated) and thus are separately described below (Table 4-2).

In the overlay model the Grid Network activities (6-10) are fully controlled and maintained by the Grid layer as described in [G2MPLS-DEP], [G2MPLS-ARCH] and G.OUNI is the interface where all Grid messages are encapsulated in the signalling protocol to carry Grid information to the other end of the network. Activity 9 is though used from the client (Network) side to User (Grid layer) side to provide Network topological information to Grid Scheduler which in turn can send detailed connection requests towards G^{2}MPLS.

In the integrated model many of the functionalities for advance reservation and co-allocation of resources are moved to the G^{2}MPLS Network Control Plane although some of Grid scheduling functionalities remain in the

Grid layer. Thus, a number of transactions between Grid and Network Layer will be required to compile the one-step co-allocation of Grid and Network resource model as portrayed below.

Grid Service Activities				
6	**7**	**8**	**9**	**10**
Grid Layer	Grid Layer	Grid Layer	Grid Layer	Grid Layer
⇕	⇕	⇕	⇕	⇕
Grid Service Establishment	Grid Service Deletion	Grid Service Status	Grid Service Auto-Discovery	Grid Use
⇕	⇕	⇕	⇕	⇕
Network Layer	Network Layer	Network Layer	Network Layer	Network Layer

Table 4-2: G.OUNI Grid Network Activities

The Grid Layer and Network Layer roles for the G.OUNI-C and G.OUNI-N for these functions are described in sections 4.6 (Grid Neighbour Discovery) and 4.7 (Grid Service Discovery). Services related to G.OUNI activities are depicted in next section.

4.3 Services offered over the G.OUNI

The services offered over G.OUNI are divided into two main categories: the network services and the Grid services. The network services are based on the ones already standardized in OIF UNI documents [1.0, UNI-01.0-R2, 2.0] plus some extensions, which are required for PHOSPHORUS and are described below. The network services basically refer to on-demand and in-advance connections, given specific parameters such as bandwidth, ingress and egress access points. On the other hand, Grid Network Services include also reservation and co-allocation as well as access to Grid resources such as CPU or storage. An overview of the Network services and Grid Network services supported over the overlay and integrated models are provided in sections 4.3.1 and 4.3.2 accordingly.

4.3.1 Network services offered over G.OUNI

A control plane consists inherently of different functional entities, one of which is concerned with the Reference Point called the User to Network Interface (UNI). The UNI standards and definitions from OIF and ITU-T can be used as a basic platform for the G.OUNI.

The Optical Internetworking Forum (OIF) defines the UNI as the service control interface between user devices and the transport network equipment from different vendors. Signalling over the UNI is used to invoke services that the transport network offers to clients. The purpose of the UNI is to specify interoperable procedures for requesting and establishing dynamic connectivity across heterogeneous networks defined by the Automatic Switched Optical Network (ASON) [OIF-UNI1.0R2-COMM]. The development of such procedures requires the definition of the physical interfaces between clients and the transport network, the connectivity services offered by the transport network, the signalling protocols used to invoke the services, the mechanisms used to transport signalling messages, and the auto-discovery procedures that aid signalling [G2MPLS-arch].

OIF UNI 1.0 Services

OIF UNI 1.0 focuses primarily on the ability to create and delete on-demand point-to-point transport network connections according to bandwidth, signal type and routing constraints. These connections can be SONET services of payload bandwidth STS-1 and higher, or SDH services of payload bandwidth VC-3 and higher whereas their properties are negotiated during the connection establishment phase.

The procedures involved through messaging between a UNI-C and a UNI-N entity (i.e. UNI signalling) for the invocation of transport network services are:

- **Signalling actions**
  - *Connection creation*: Permits the creation of a connection under specific network constrains and security procedures.
  - *Connection deletion*: Identifies the disposal of an existing connection.
  - *Connection status enquiry*: Represents the exchange of specific connection attributes allowing the *connection* status *discovery*
- ***Connection types which is defined by:***
  - *Framing (e.g. SONET/SDH)*
  - *Transparency*
    - Section transparency
    - Line transparency
    - Path transparency
  - *Signal Types*
  - *Concatenation*
    - Contiguous concatenation
    - Virtual concatenation

Furthermore the following supported procedures are defined under UNI 1.0:

- ***UNI Neighbour Discovery (Optional)***: Neighbour discovery procedure allows Transport Network Elements (TNEs) to discover their identities as well as the identities of remote ports to which their local ports are connected and can be characterized as an essential procedure required for performing interface mapping between a client and a TNE.
- ***Service Discovery (Optional)***: Service discovery is invoked after neighbour discovery is complete to allow a UNI-C to expose to the transport network the client device capabilities and to acquire from the UNI-N, transport network service information (i.e. the signalling protocols used and UNI versions supported, client port-level service attributes, transparency service support, and network routing diversity support).
- ***Signalling Control Channel Maintenance***: OIF UNI 1.0 supports procedures for maintenance of the control channel under different configuration possibilities. These procedures allow signalling peers to continuously monitor and maintain control channel connectivity for UNI signalling.

UNI 2.0 Extensions

OIF work has been continued towards UNI 2.0, under which a connection can be a SONET service of bandwidth VT1.5 and higher, or SDH service of bandwidth VC-11 and higher, an Ethernet service, or a G.709 service and the properties of the connection are defined by the attributes specified during connection establishment. The following features are added in UNI 2.0 [OIF-UNI2.0-COMM]:

- Separation of Call and Connection Controllers
- Multi- and Dual- Homing for Diverse Routing
- Non-Disruptive Connection Modification
- 1: N Signalled Protection
- Sub STS-1 Rate Connections
- Transport of Ethernet Services
- Transport of G.709 Interfaces
- Enhanced Security

OIF UNI 2.0 is officially a working document in OIF Architecture and Signalling Working Group, but it has a consolidated position towards becoming a de-facto Implementation Agreement, because of the wide vendor support and the many interoperability events demonstrated worldwide in conferences (e.g. Supercomm) and research projects (e.g. IST-FP6-MUPBED). The Phosphorus project has no formal liaison with OIF on UNI2.0, but some partners contributed background know-how on the issue deriving from the participation in other research projects (e.g. PSNC also partner of the IST-FP6-MUPBED consortium).

G.OUNI

The Grid enabled optical network needs to cope with various types of demanding users and applications and thus is necessary to support different types of control plane architectures (overlay, integrated), service

invocation scenarios (direct, indirect). In this scenario, G.OUNI is responsible for providing on-demand access to Grid services by encompassing the interoperable procedures between the Grid users and the optical network. To facilitate the support of Grid capabilities and to address the wide variety of requirements G.OUNI must be implemented combining the various UNI standards plus some additional functionality. Towards this direction new signalling and transport mechanisms in addition to the above mentioned must be developed. In these the signalling will be responsible for requesting, establishing and maintaining connectivity between Grid users and Grid resources while the data transport mechanism will provide traffic mapping between the Grid service and the optical transport network [OGF-GDF36].

The introduced G.OUNI functionalities can be grouped in the following categories:

- Control Plane
  - *Network Topology Enquiry and Restoration*

    G.OUNI must be able to provide network topology information to Grid users/services and implement various protection and restoration schemes to deal with the diverse nature of transport network failures.
  - *Network Resource Availability*

    All available network resources (i.e. amount of bandwidth, links, cross-connects etc.) should be discovered by the G.OUNI and facilitated to Grid services.
  - *Network Resource Capability*

    For supporting different bandwidth demands for Grid users and services G.OUNI should introduce flexible allocation mechanisms providing multiple wavelengths, single wavelength or sub-wavelength bandwidth.
  - *Network Advance Reservation*

    Through automatic service discovery G.OUNI will provide users the capability to automatically schedule, provision and set-up lightpaths across the network, thus facilitating network advance reservations. WS-Agreements are able to describe advance reservation and more info is provided on Section 4.9 and section 4.11.4.2 reflects RSVP extensions towards advance reservations.
- Transport Plane
  - *Traffic classification and shaping*

    G.OUNI should be able to map the user data traffic into the used transmission entity (e.g. timeslot, wavelength etc.) performing traffic classification and aggregation.
  - *Data plane security*

    A security mechanism able to support security credentials and policy information provided by any agreement provider will be facilitated by the G.OUNI.

Network services supported by PHOSPHORUS overlay and integrated models are:

- Connection creation
- Connection deletion
- Connection status enquiry
- UNI Neighbour Discovery
- Service Discovery
- Signalling Control Channel Maintenance
- Network Topology Enquiry and Restoration
- Network Resource Availability
- Network Resource Capability
- Network Advance Reservation
- Traffic classification and shaping
- Data plane security

4.3.2 Grid services offered over G.OUNI

In the same way as network services, Grid services are offered to clients over G.OUNI. Grid services mainly allow on-demand access to Grid resources considering reservation, allocation, actual use and release. Moreover, procedures such as discovery of capability and availability of computational resources are also required in order to facilitate Grid resource management. WS-agreements, described next, will be used to request services and map then into the G.OUNI.

4.3.2.1 *Grid services offered by overlay G^{2}MPLS NCP and integrated G^{2}MPLS NCP*

In PHOSPHORUS, services offered over G.OUNI depend on the architectural model chosen. Within the overlay model, G.OUNI basically offers network services described in the previous section, however, information about Grid resource capability needs to be exchanged through the G.OUNI and propagated by the G^{2}MPLS NCP. In this case, co-allocation is performed by the Grid Middleware. On the other hand, under an integrated model, G^{2}MPLS also supports meta-scheduling processing such as co-allocation or indexing, so G.OUNI services should be extended and these operations can be done.

The overlay G^{2}MPLS Control Plane architecture basically offers network services but information about Grid resource capability and availability has to be exchanged between middleware modules. As no IP layer is used for the interconnection of these modules, the information is distributed using the control plane. Therefore, the

G.OUNI includes a new service that allows the flooding of information from one middleware module to the others (Figure 4-2).

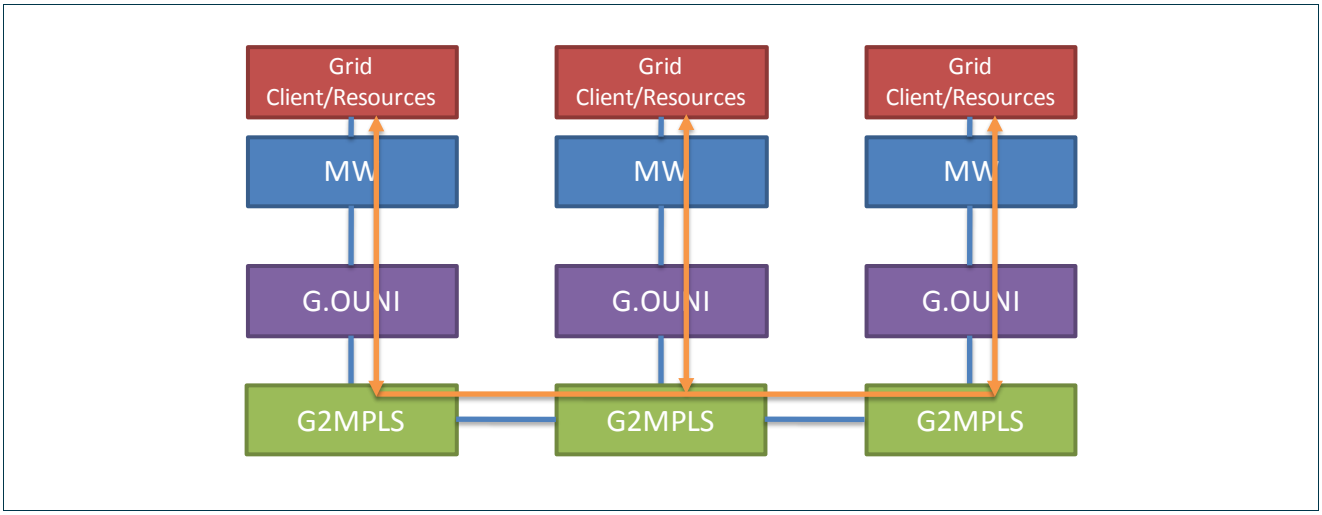

Figure 4-2: Grid information exchange through the G^{2}MPLS NCP

G.OUNI offers:

1. Grid Service Discovery
2. Grid Resource Discovery

The Grid service and resource discovery mechanisms across the G.OUNI comprise the negotiation between Grid client/application and the G^{2}MPLS control plane. The negotiation includes exchange of information describing resource capabilities (e.g. CPU, storage…) and availabilities. This information is then advertised to the network and flooded by the G^{2}MPLS control plane.

The integrated G^{2}MPLS Control Plane architecture considers Grid resources as part of the network. In this case, G.OUNI does not only offer network services such as end-to-end connectivity, but it also supports some meta-scheduling processing previously operated by the Grid Middleware and now provided at the control plane level.

G.OUNI offers:

1. Grid Service Discovery
2. Grid Resource Discovery
3. Grid Advance Reservation Request
4. Grid Advance Reservation Cancellation
5. Grid Resource co-allocation

Besides the services described for the overlay model, the G.OUNI must support the request of advance reservations of Grid resources, in which resources are pre-reserved prior in time and allocated when they are really needed. Moreover, the meta-scheduling processing has to be supported at the G.OUNI. All abovementioned services are listed in Table 4-3.

Grid Services	Overlay	Integrated
Grid Service Discovery	✓	✓
Grid Resource Discovery	✓	✓
Grid Advance Reservation Request	x	✓
Grid Advance Reservation Cancellation	x	✓
Grid Resource co-allocation	x	✓

Table 4-3: Grid Services

4.3.2.2 *Service Provisioning using Service Level Agreements*

Today it is common practice to use Service Level Agreement (SLA) in grid resource management. SLAs provide a flexible way to describe desired services, their properties, rewards (penalties) for successful (unsuccessful) service provisioning, and the guarantees that must be met in order to fulfil service requirements. Furthermore, SLAs provide a domain independent way to describe atomic services such as resource reservation services, as well as complex services like co-allocation services.

The Web Service Agreement specification of the Open Grid Forum (OGF) became a standard way within the grid community to use SLAs. Therefore, the WS-Agreement specification defines a term language to express the different aspects of a service level agreement and a protocol to create SLAs based on templates and to monitor the state of the SLAs and of the given guarantees. Below there is a list and a brief description of the common features that atomic and complex services should provide over G.OUNI. More detailed information regarding these services is provided on section 4.9.

- *Definition of Reservation Times*
- *Restriction and Validation of SLA Offers*
  - *Restriction of the Agreement Offer Structure*
  - *Restriction of Value Facets*
- *Negotiation of agreement templates*
- *Agreement creation*

Definition of Reservation Times

A guarantee expresses a Quality of Service (QoS) level that a service must fulfil. In general guarantees in WS-Agreement are expressed in the form of Guarantee Terms. A guarantee term applies to one or more services in the agreement. The services of the agreement are described by their service description terms. Compute services that support advance reservations may expose this functionality by using SLAs. Services that support advance reservation of resources expose a special guarantee term in the agreement template.

Restriction and Validation of SLA Offers

Besides describing a service and the associated guarantees, it is also important to define the structure and the value facets of a valid SLA offer. This applies to the service description terms and their content as well as to the guarantee terms. Such a restriction can be included in a SLA template by using *Creation Constraints*.

Restriction of the Agreement Structure

Services defined by an SLA are described by service description terms using a certain description language. These description languages usually aim to cover the purposes of a certain domain. Therefore, they are normally quite flexible in the way they can be used. For instance the job description language JSDL allows specifying the total number of CPUs required for a compute job by an upper bound, a lower bound, an exact value and a value range. Usually only a subset of these options should be used. E.g. a user may only specify the exact value of the needed CPUs. Therefore Creation Constraints are a way to restrict the structure of valid agreement offers.

Restriction of Value Facets

Besides restricting the structure of a SLA offer, there is also the need to restrict the value facets of the offer. A typical example is a compute service, where the maximum number of available resources (e.g. compute nodes) needs to be restricted to 128 and the number of individual available CPUs to 2.

Negotiation of agreement templates

In a co-allocation process scheduling services need resource availability information of the involved systems in order to efficiently calculate a common reservation time. In the easiest way the availability information is provided in an agreement template, e.g. by including appropriate creation constraint. Alternatively, availability information can be provided by a separate service. However, for a specific resource provider or specific types of resources availability information may only be partially or not even available at all. Network resource management systems are a good example for systems, where availability information may not be expressed in a convenient way. In order to co-allocate this type of resources, negotiation mechanisms are required.

Agreement creation

After the negotiation of an agreement template, a client wants to create an according agreement. Here a transaction problem arises for co-allocation scenarios. A grid scheduler needs to create a set of SLAs with different resource providers in order to provide co-allocated resource. Therefore, the scheduler first negotiates a set of templates with the providers and identifies the possible reservation times of the required resources. Since the negotiated templates are not binding, a situation may occur in which one of the service provider cannot deliver the negotiated service. To overcome this problem two approaches exist:

1. to use transactions to create the SLAs, or
2. to create each SLA within one step.

These two approaches will be explained in detail in section 4.9.

Atomic G.OUNI Services (Resource Reservation Service)

Resource reservation services are atomic services. They provide the basic functionality for complex services such as co-allocation services, e.g. they enable complex services to reserve resources in a certain timeframe. These resources are then available at the reservation time. In that way scheduling services can coordinate multiple resources in order to create higher-level services. Atomic services typically encapsulate low-level scheduler, local scheduler, or local resource management systems. They are located close to a resource, and control for example a cluster, a supercomputer, or a network of workstations, Atomic compute services can usually provide guarantees like execute a job at a specific time, finish a job execution until a certain time, execute a job exclusively on a resource, run a job until completion (no preemption).

Complex G.OUNI Services (Co-allocation Services)

Complex services as grid scheduler, super scheduler, and resource broker work on top of lower level scheduler. They use grid functionality to discover resources that meet job requirements and orchestrate resources in order to provide non-trivial quality of service. A service that co-allocates compute resources at different sites is an example for a complex compute service.

It is obvious, that atomic services provide the basic functionality for complex services, and that the functionalities of both, atomic and complex services, differ a lot. However, service level agreements may be employed in order to provide atomic services as well as complex services.

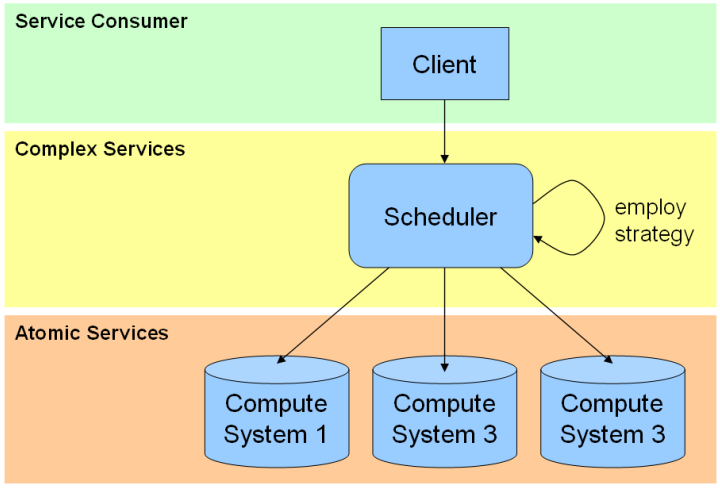

Figure 4-3: SLA Layered Model

In Figure 4-3 a simple abstraction of grid infrastructure is given. A grid scheduling system manages a set of compute systems (e.g. 3 different clusters) in a certain domain. It uses the atomic SLA to coordinate the resources from the different systems in time. This is done transparently by the scheduling system behalf of a user.

If a client wants to submit a complex compute job, it queries an SLA template from the grid scheduling system. An application consisting of several modules is a sample of such a complex compute job. Each component of the application may be executed on a different compute system, and a certain network bandwidth between the compute systems may be required by the application in order to run optimally. The components of the application communicated with each other. This means that the resources of the different compute systems must be co-allocated. This also applies to the network resources, which must be available at the application runtime. Furthermore, the required bandwidth may depend on the number of compute resources available to the application. Moreover, not every possible combination of resources may result in an optimal application performance.

Therefore, the grid scheduling system provides the user with a set of templates that comprise sensible application configurations. The user can choose one template and modify its content according to his/her requirements. It is up to the grid scheduling system which degree of freedom to change the content of the SLA template it gives to the user. Some systems may allow a user to change the content of a template according to the physical constraints of the underlying systems. Other systems, which are more restrictive, provide the user with a set of templates. The user may then choose one of the provided templates, which serves its purposes

best. In such templates the user may only specify a very limited number of properties, e.g. the location of input files for the compute job. The mechanisms to constraint the structure of an agreement and its valid value facets were already described for atomic services and apply also for complex services.

4.4 G.OUNI service invocation configurations

As the G.OUNI is used to trigger Grid and network transactions, it must support the invocation of both transport network services and grid resources services. Under the G.OUNI, services may be invoked in two models: direct and indirect invocation [OIF-UNI1.0R2-COMM]. In the direct invocation model, the G.OUNI-C functionality is implemented in the client itself while under the indirect invocation model an entity called the *Proxy G.O*UNI-C performs G.OUNI functions on behalf of one or more clients. The invocation of grid and network resources services will be supported by a component called GOUNI-N Agent which is located on all the edge network elements of the GMPLS cloud.

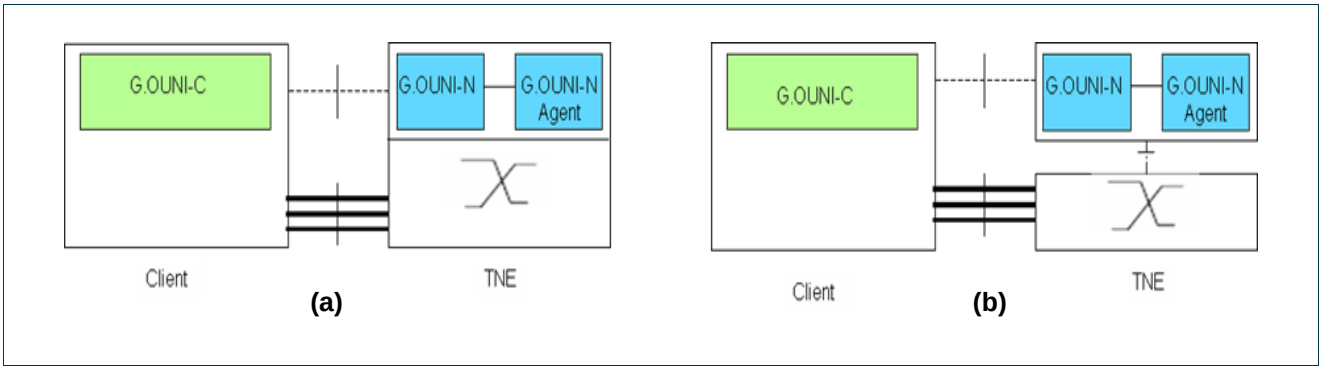

Figure 4-4: Direct Service Invocation Configurations

Different configurations of direct and indirect invocation can be obtained depending on the implementation of the G.OUNI-N functionality. The G.OUNI-N functionality can be implemented in the TNE as shown in Figure 4-4.a and Figure 4-5.a or as a proxy as shown in Figure 4-4.b and Figure 4-5.b.

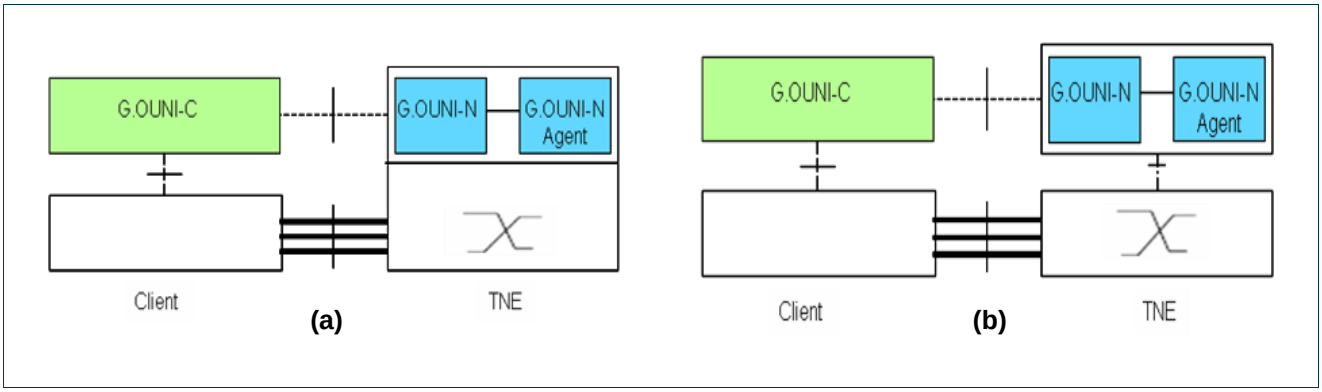

Figure 4-5: Indirect Service Invocation Configurations

In D.2.1 a number of use cases for the G^{2}MPLS architecture were provided to indicate invocation services offered by the G.OUNI. The terms *direct* and *indirect invocation* are used in a different way than that identified in [OIF-UNI1.0R2-COMM] and represent different application use cases. Under the *direct invocation use case* the user configures and requests the execution of a job that involves Grid resources located at his local Grid site (Vsite A) and a remote Grid (Vsite B). During this direct invocation model a G.OUNI is required to build a Grid Network Service between those two sites as depicted in Figure 4-6. An overview of the role of G.OUNI in the indirect use case is provided on the table below.

Steps	
1.	The G.OUNI schedules and configures the local Grid resources via LRMS on Vsite A.
2.	The G.OUNI sends a GNS transaction request to the G.OUNI-N on the peering G^2.LER.
3.	The egress G^2.LER sends the GNS transaction request to the peering G.OUNI gateway.
4.	The G.OUNI gateway on site B requests its LRMS in the middleware to schedule and configure the local Grid resources.
5.	The LRMS at Vsite B sends job request confirmation to the G.OUNI gateway.

Table 4-4: G.OUNI transactions for PHOSPHORUS direct use case

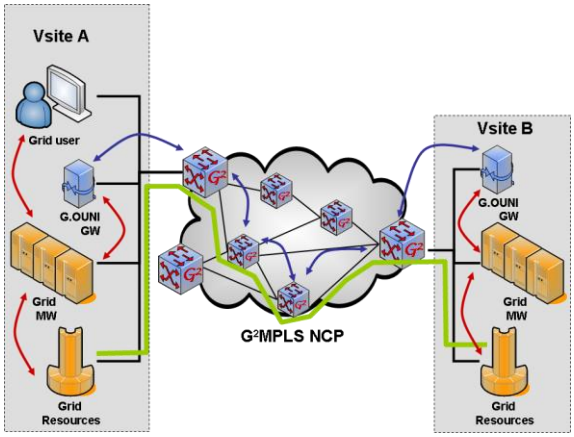

Figure 4-6: PHOSPHORUS direct invocation use case ([G2MPLS-ARCH])

The indirect invocation involves requesting Grid resources all located remotely from the user. It follows similar steps and the ones that involve G.OUNI are reflected below.

Steps	
1.	The Grid broker requests the local G.OUNI gateway to transact GNS between Vsite A and Vsite C.
2.	The G.OUNI gateway sends a GNS transaction request to the G.OUNI-N (acting as a proxy) on the peering G^2.LER.
3.	The G^2.LER peering with Vsite A sends the GNS transaction request to the peering G.OUNI gateway.
4.	The G.OUNI gateway at Vsite A requests the LRMS in the middleware to schedule and configure the local Grid resources for Vsite A.
5.	The LRMS at Vsite A sends job request confirmation to the G.OUNI gateway.
6.	The G.OUNI gateway sends a GNS transaction response to the G.OUNI-N on the peering G^2.LER.
7.	The egress G^2.LER sends the GNS transaction request to the peering G.OUNI gateway.
8.	The G.OUNI gateway requests the LRMS in the middleware to schedule and configure the local Grid resources for Vsite B.
9.	The LRMS at Vsite B sends job request confirmation to the G.OUNI gateway.
10.	The G.OUNI gateway sends a GNS transaction response to the G.OUNI-N on the peering egress G^2.LER.
11.	The G^2.LER peering with Vsite A sends a GNS transaction response to the G.OUNI gateway at Vsite A.

12.	The GNS transaction response is sent back by the G.OUNI gateway at Vsite C to the local Grid scheduler.

Table 4-5: G.OUNI transactions for PHOSPHORUS indirect use case

Complete description of both direct and indirect use case is provided in [G2MPLS-ARCH]. However, the OIF UNI invocation models can be always applied for all of the identified use-cases in D.2.1.

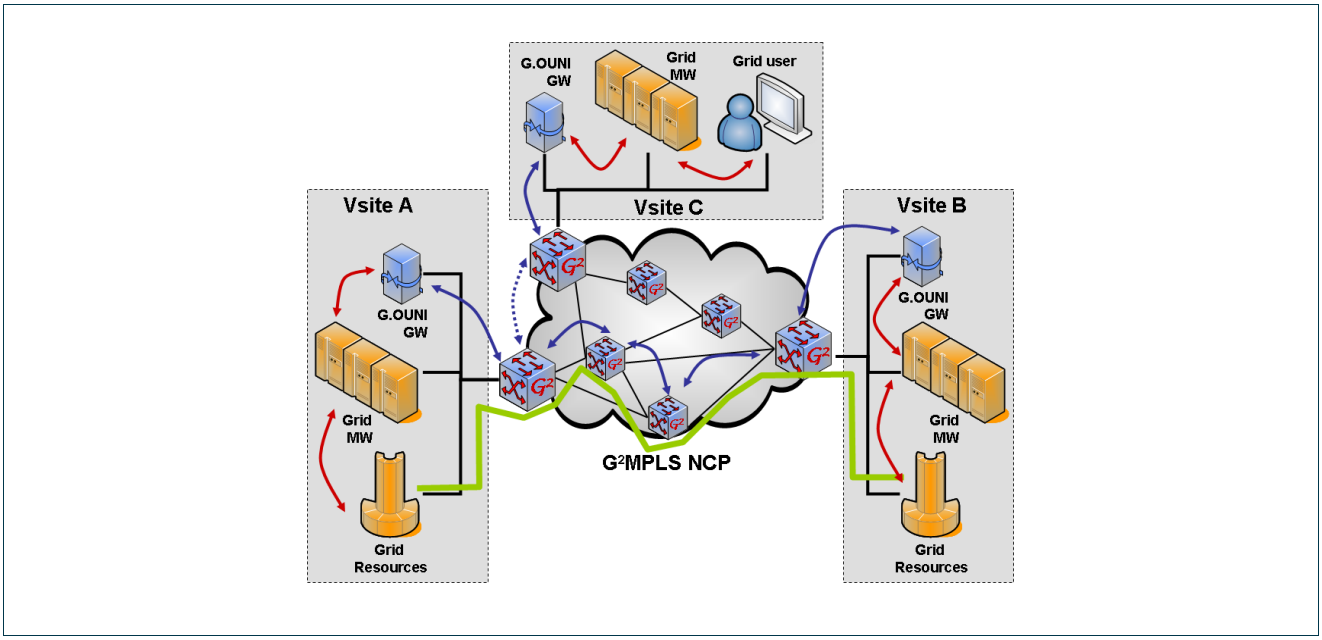

Figure 4-7: PHOSPHORUS in-direct invocation use case ([G2MPLS-ARCH])

4.5 G.OUNI signalling configurations

G.OUNI supports both in-fibre and out-of-fibre Control Channel configurations between the G.OUNI-C and the G.OUNI-N. Under the in-fibre configuration the IPCC is embedded in the data carrying optical link between the client and the TNE. This configuration applies only to the direct service invocation model depicted in Figure 4-4.a. The out-of-fibre configuration involves a separate dedicated communication link for signalling messages between the client and the TNE. This type of signalling applies to all the invocation configurations shown in Figure 4-4 and Figure 4-5.

The G.OUNI supports signalling over OTN (G.709-based) and Ethernet connections in addition to SONET/SDH (supported by the OIF-UNI1.0R2-COMM). However The UNI 2.0, which is still in the OIF acceptance stage [OIF-UNI2.0-COMM, OIF-UNI2.0-RSVP], is planning to include these connections.

4.6 Addressing (Grid and network entities)

Multiple address spaces are needed to support the identification and reachability of different entities within the transport network and the Grid resources. The following address spaces are involved in identifying entities under the G.OUNI:

- Internal transport network addresses.
- G.OUNI-N and G.OUNI-C Node Identifier (Node ID).
- IPCC Identifier (CCID).
- Transport Network Assigned (TNA) address.
- Grid End-points addresses.
    - Grid Client Site addresses
    - Grid Resources addresses

The first four address spaces identify the transport network as detailed in [OIF-UNI1.0R2-COMM]. The Grid end address spaces are defined under the G.OUNI to identify the client side and the Grid resources attached to it. These addresses follow the addressing schemes used in client networks (e.g. IP, ATM, etc). Grid resources addresses are directly exchanged in signalling between the G.OUNI-N and the G.OUNI-C as part of the Grid resources capability and availability discovery as will be discussed in sections 4.7, 4.8. This address spaces are shown in Figure 4-8.

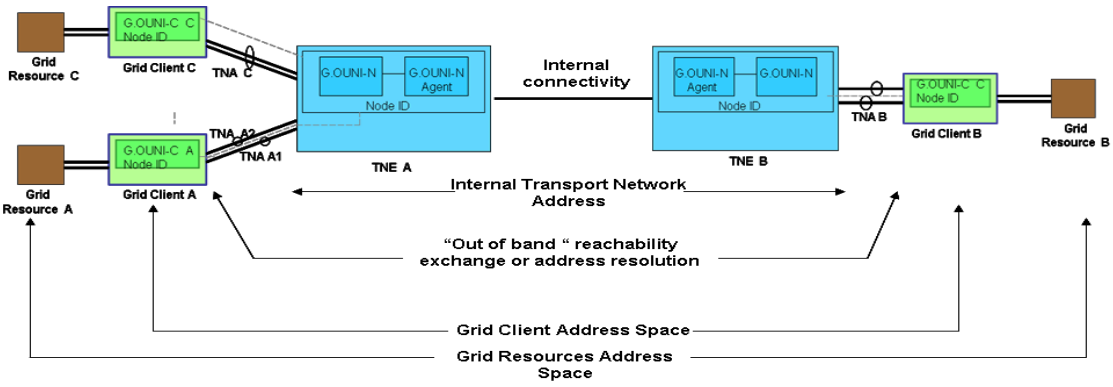

Figure 4-8: Address Spaces for G.OUNI

4.7 Grid Neighbour Discovery

The Grid Neighbour discovery procedures defined in this section allow TNEs and directly attached Grid Client devices to determine the identities and capabilities of each other and the identities of remote ports to which their local ports are connected. The Grid Neighbour discovery procedure is optional for the network resources in G^{2}MPLS NCP and mandatory for the Grid resources. If the Grid Neighbour discovery is not implemented in the Grid Client and the TNE, then the neighbour and remote identifiers (ports, resources etc.) must be manually configured in the corresponding G.OUNI-C and G.OUNI-N. The implementation of the neighbour discovery as described in this section avoids the need for manual configuration of neighbour Grid Users/Resources connectivity to G^{2}MPLS network. It, thus, provides a means of automatically detecting the status of Grid resource/application capabilities, as well as inconsistencies in physical wiring.

The neighbour discovery procedure is vital for establishing a dynamic interface between client devices and the network side**.** Under the G^{2}MPLS, the neighbour discovery procedures should include, in addition to local ports identification and IPCC maintenance [OIF-UNI1.0R2-COMM], discovery of the capabilities of the Grid resources at Grid sites attached to the G.OUNI-C. In the G.OUNI the functional entity responsible for advertising the capabilities of the grid resources is called the Discovery Agent. However, in the case this procedure is not implemented, the information provided by it will be configured by the service discovery procedure as it is impractical to configure it manually in such a very dynamic environment.

The PHOSPHORUS approach will use a full OSPF based mechanism in the case that IPCCs are statically configured for grid resources capability advertising. The local ports identification and IPCC maintenance

procedures defined in [OIF-UNI1.0R2-COMM] are typically based on the LMP which is also described in Section 11.

4.8 Grid Service Discovery

Under G^{2}MPLS implementation G.OUNI must support mechanisms for the indication and negotiation of both network and Grid resources as part of the service discovery procedures. These procedures are invoked after neighbour discovery is complete. It consists of a sequence of message exchanges between the G.OUNI-C and the G.OUNI-N. The following service attributes are supported by the G.OUNI:

- Signalling protocols and G.OUNI version supported
- Client port-level service attribute
- Transparency service support
- Network routing diversity support
- Grid services/resources capability
- Grid resources availability

The first four attributes are already supported and described on the OIF UNI1.0. Grid resources capability advertising is an essential part of service discovery under the G.OUNI. It allows the GNS-Agent to discover the Grid resources attached to the clients. The G.OUNI-C initiates the capability advertising process by sending a message to the G.OUNI-N. This message includes the Grid client node address and the Grid resources addresses of all the resources attached to the G.OUNI-C and their specific capabilities (e.g. types of CPU, storage, OS, etc.). The G.OUNI-N replies by sending a response message acknowledging the reception of the information. On the other hand, Grid resources availability allows the advertisement of the state and amount of the Grid resources available in a Grid site.

4.9 Description of WS-Agreement Services

WS-Agreement specification defines a term language to express the different aspects of a service level agreement and a protocol to create SLAs based on templates and to monitor the state of the SLAs and of the given guarantees. The Open Grid Forum (OGF) standardised the Web Service Agreement specification in order to use SLAs. Atomic and complex services that where initially described in section 4.3.2.2 are thoroughly described below.

Atomic G.OUNI Services (Resource Reservation Service)

An atomic service usually deals only with the provisioning of one resource type, e.g. compute resources. In WS-Agreement services are described by Service Description Terms (SDTs). A SDT has a name and additionally specifies a service name. The name uniquely identifies the SDT and the service name identifies the service the SDT describes. In principle the WS-Agreement specification allows to specify multiple SDTs for one service, but atomic services are usually described by only one SDT.

Service description terms are domain independent. This means that a domain specific definition is needed to describe the service that will be delivered. For compute services such a definition already exists in the form of the JSDL specification [JSDL-SPEC]. JSDL provides the means to specify the application to execute, the required resources, the location of input and output files, and to identify a compute job.

Applications are defined in the Application section of the JSDL document, and are identified by their name and their version. They can also have an optional description. In general applications must be installed and configured at the target systems, so that they can be invoked by the grid middleware.

The Resources section of the JSDL document specifies the required resources for a certain job. This section includes e.g. the required operating system, the candidate hosts, the required CPU architecture and speed, the required individual CPU time, and the required amount of resources. These values and their constraints (e.g. operating system, CPU architecture and speed, and maximal resources count) are usually configured within the grid middleware. Valid JSDL documents must not violate these constraints. An example of a SDT is shown in Listing 4-1.

At this point it is important to note that the sample SDT contains a JSDL document. For a different resource the JSDL document can be replaced with another resource description language. For example an atomic service that schedules and controls the access to a limited set of software licenses would define a specific license in a SDT by using a domain specific description language. By including a reservation guarantee, the software license can be reserved at a certain time. Therefore, SLAs can be used for coordinating compute resources and licenses. Alternatively, a network service that reserves a certain bandwidth between two endpoints could be specified by using an applicable description language.

```
<wsag:ServiceDescriptionTerm wsag:Name="COMPUTE_SERVICE"
        wsag:ServiceName=" RESERVATION_SERVICE ">
        <jsdl:JobDefinition>
                <jsdl:JobDescription>
                        <jsdl:Application>
                                <jsdl:ApplicationName>Application_1</jsdl:ApplicationName>
                        </jsdl:Application>
```

```
                    <jsdl:Resources>
                        <jsdl:OperatingSystem>
                            <jsdl:OperatingSystemType>
                                <jsdl:OperatingSystemName>
                                    LINUX
                                </jsdl:OperatingSystemName>
                            </jsdl:OperatingSystemType>
                            <jsdl:OperatingSystemVersion>
                                9.2
                            </jsdl:OperatingSystemVersion>
                        </jsdl:OperatingSystem>
                        <jsdl:CPUArchitecture>
                            <jsdl:CPUArchitectureName>x86</jsdl:CPUArchitectureName>
                        </jsdl:CPUArchitecture>
                        <jsdl:IndividualCPUSpeed>
                            <jsdl:Exact>2.0E9</jsdl:Exact>
                        </jsdl:IndividualCPUSpeed>
                        <jsdl:IndividualCPUCount>
                            <jsdl:Exact>2.0</jsdl:Exact>
                        </jsdl:IndividualCPUCount>
                        <jsdl:TotalResourceCount>
                            <jsdl:Exact>1.0</jsdl:Exact>
                        </jsdl:TotalResourceCount>
                    </jsdl:Resources>
                </jsdl:JobDescription>
            </jsdl:JobDefinition>
</wsag:ServiceDescriptionTerm>
```

Listing 4-1: Description of a Compute Service

Complex G.OUNI Services (Co-allocation Services)

The main difference between atomic services and complex services is that atomic services mainly comprise one resource (e.g. only one resource type at a certain site is required for the service provisioning); while complex services potentially consist of multiple components, e.g. different types of resources or resources at different sites are required for the service. A service that co-allocates compute resources at different sites is an example for a complex compute service (Figure 4-3).

Listing 4-2 shows an example of the service description terms and guarantees of a complex service. In this example an application is defined, which consists of two components A and B. These components are

executed on different systems. Additionally, a guarantee on the available network bandwidth is given. The compute and network resources required to execute the application were co-allocated by the grid scheduler by orchestrating atomic service level agreements defined in the previous sections. These orchestrated systems are exposed by the grid scheduler in the form of a complex service.

```
<wsag:ServiceDescriptionTerm wsag:Name="COMPONENT_A"
        wsag:ServiceName="APPLICATION_1">
        <jsdl:JobDefinition>
                <jsdl:JobDescription>
                        <jsdl:Application>
                                <jsdl:ApplicationName>Component_1</jsdl:ApplicationName>
                        </jsdl:Application>
                        <jsdl:Resources>
                                <jsdl:TotalResourceCount>
                                        <jsdl:Exact>1.0</jsdl:Exact>
                                </jsdl:TotalResourceCount>
                        </jsdl:Resources>
                </jsdl:JobDescription>
        </jsdl:JobDefinition>
</wsag:ServiceDescriptionTerm>

<wsag:ServiceDescriptionTerm wsag:Name="COMPONENT_B"
        wsag:ServiceName="APPLICATION_1">
        <jsdl:JobDefinition>
                <jsdl:JobDescription>
                        <jsdl:Application>
                                <jsdl:ApplicationName>Component_2</jsdl:ApplicationName>
                        </jsdl:Application>
                        <jsdl:Resources>
                                <jsdl:TotalResourceCount>
                                        <jsdl:Exact>2.0</jsdl:Exact>
                                </jsdl:TotalResourceCount>
                        </jsdl:Resources>
                </jsdl:JobDescription>
        </jsdl:JobDefinition>
</wsag:ServiceDescriptionTerm>

<wsag:GuaranteeTerm wsag:Name="NETWORK_GUARANTEE_1">
        <wsag:ServiceScope wsag:ServiceName="APPLICATION_1"/>
```

```
        <wsag:ServiceLevelObjective>
                <wsag:CustomServiceLevel xsi:type="kpi:GuaranteeAttributesType"
                        xmlns:kpi="http://schemas.PHOSPHORUS.eu">
                        <kpi:NetworkGuarantee>
                                <kpi:bandwidth>5 GBit</kpi:bandwidth>
                                <kpi:bidirectional>true</kpi:bidirectional>
                                <kpi:source>COMPONENT_A</kpi:source>
                                <kpi:destination>COMPONENT_B</kpi:destination>
                        </kpi:NetworkGuarantee>
                </wsag:CustomServiceLevel>
        </wsag:ServiceLevelObjective>
        <wsag:BusinessValueList/>
</wsag:GuaranteeTerm>
```

Listing 4-2: Complex Compute Service

Definition of Reservation Times

One objective for creating agreements is to provide guarantees associated with a service. A guarantee expresses a Quality of Service (QoS) level that a service must fulfil. In general guarantees in WS-Agreement are expressed in the form of Guarantee Terms. A guarantee term applies to one or more services in the agreement. The services of the agreement are described by their service description terms. One guarantee term can apply to one or more services, which are referenced in the Service Scope of the guarantee term. The guaranteed quality of a service is defined in the Service Level Objective (SLO) of the guarantee term. The service level objective contains a key performance indicator with a custom service level definition. Additionally a guarantee term can include a list of business values, which specifies the rewards that are granted when the service guarantees were met successfully, respectively the penalties when the guarantees were violated.

The grid compute jobs are in general executed in a best effort fashion. This means jobs respectively applications are in general simply passed to a batch queuing system to execute at the next possible time the required resources are available. This implies that the execution time of a job is normally not known in advance. In order to enable more complex services, such as co-allocation services, compute services need to expose resource reservation functionality. Resource reservation functionality can be used for compute services to e.g. control the execution time of an application and therefore coordinate multiple resources. This means a client application is able to specify the exact execution time of a compute job.

Compute services that support advance reservations may expose this functionality by using SLAs. Services that support advance reservation of resources expose a special guarantee term in the agreement template. The name of this guarantee term is ADVANCE_RESERVATION_GUARANTEE. This guarantee term is used to

specify the start time of a certain service. The service that a guarantee term relates to is defined in the *ServiceScope* object. A guarantee term can refer to one or more services. In the given example the guarantee relates to a service named RESERVATION_SERVICE. This service consists of exactly one compute job. That means that the sample SLA represents in fact an advance reservation compute job.

```
<wsag:GuaranteeTerm wsag:Name="ADVANCE_RESERVATION_GUARANTEE">
        <wsag:ServiceScope wsag:ServiceName="RESERVATION_SERVICE" />
        <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                        <wsag:KPIName>ADVANCE_RESERVATION</wsag:KPIName>
                        <wsag:CustomServiceLevel>
                                <ext:ReservationTime>
                                        2007-03-15T11:53:41.921+01:00
                                </ext:ReservationTime>
                        </wsag:CustomServiceLevel>
                </wsag:KPITarget>
        </wsag:ServiceLevelObjective>
        <wsag:BusinessValueList />
</wsag:GuaranteeTerm>
```

Listing 4-3: Advance Reservation Guarantee Term

Restriction and Validation of SLA Offers

Besides describing a service and the associated guarantees, it is also important to define the structure and the value facets of a valid SLA offer. This applies to the service description terms and their content as well as to the guarantee terms. Such a restriction can be included in a SLA template by using *Creation Constraints*. Each creation constraint refers to a set of elements in the agreement template. The elements are identified by a XQuery expression. A creation constraint can restrict the structure of the specific elements, or it can restrict their value facets.

Creation constraints are used by the agreement initiator (e.g. client) in order to create valid agreement offers. Through the usage of creation constraints the agreement initiator is able to determine what agreement offers an agreement responder is willing to accept. Depending on the information the agreement responder is willing to expose, this information can be very detailed. In turn, agreement responders (e.g. resource provider) use creation constraints to validate agreement offers. When an agreement provider receives an agreement offer, it looks up the template that was used to create the offer. Then each item in the offer referenced by a creation

constraint is validated against the constraint defined in the template. Only if all creation constraints are validated successfully the SLA is created.

Restriction of the Agreement Structure

Services defined by an SLA are described by service description terms using a certain description language. These description languages usually aim to cover the purposes of a certain domain. Therefore, they are normally quite flexible in the way they can be used. For instance the job description language JSDL allows specifying the total number of CPUs required for a compute job by an upper bound, a lower bound, an exact value and a value range. Therefore, inconsistent job definitions can be created using *pure JSDL* for the interface definitions. These inconsistencies are resolved by restricting complex types by using creation constraints.

By restricting complex types within an agreement, the content of the SLA is defined very accurately. In fact new, more concrete types are derived from the standard ones (e.g. JSDL). These types are still standard compliant, but are much more restrictive. In the given example (Listing 4-4) the usage of the element *TotalCPUCount* from the type *RangeValue_Type* of the JSDL standard is restricted in a way that only the *Exact* element must occur exactly one time in the element *TotalCPUCount.* By default the elements *UpperBoundedRange* and *LowerBoundedRange* may occur 0 to 1 time in *TotalCPUCount*, and the elements *Exact* and *Range* may occur 0 to n times.

```
<wsag:Item>
        <wsag:Location>//jsdl:JobDefinition/jsdl:JobDescription/jsdl:Resources/jsdl:TotalCPUCount
        </wsag:Location>
        <wsag:ItemConstraint>
                <xs:sequence>
                        <xs:element name="UpperBoundedRange" type="jsdl:Boundary_Type" minOccurs="0"
maxOccurs="0"/>
                        <xs:element name="LowerBoundedRange" type="jsdl:Boundary_Type" minOccurs="0"
maxOccurs="0"/>
                        <xs:element name="Exact" type="jsdl:Exact_Type" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="Range" type="jsdl:Range_Type" minOccurs="0" maxOccurs="0"/>
                </xs:sequence>
        </wsag:ItemConstraint>
</wsag:Item>
```

Listing 4-4: Restriction of the Agreement Structure

Restriction of Value Facets

Besides restricting the structure of an SLA offer, there is also the need to restrict the value facets of the offer. A typical example is a compute service, where the maximum number of available resources (e.g. compute nodes) needs to be restricted to 128 and the number of individual available CPUs to 2. These constraints result e.g. from the physical restrictions of the available resources.

Listing 4-5 shows a simple creation constraint, which restricts the minimum and maximum number of total available CPUs of a compute system.

```
<wsag:Item>
        <wsag:Location>//jsdl:JobDefinition/jsdl:JobDescription/jsdl:Resources/jsdl:TotalCPUCount/jsdl:Exact
        </wsag:Location>
        <wsag:ItemConstraint>
                <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                        <xs:restriction base="xs:double">
                                <xs:minInclusive value="1" />
                                <xs:maxInclusive value="128" />
                        </xs:restriction>
                </xs:simpleType>
        </wsag:ItemConstraint>
</wsag:Item>
```

Listing 4-5: Restriction of the Available Resources

Creation constraints may not only have one continuous value facet, but multiple. An example for this is the restriction of valid start times for a service with a given quality. Such a service could be e.g. a compute service, where a certain amount of compute nodes have to be provided for a certain time. This means the possible start times of the service must be restricted to a specific value facet. In other words the value facets of a guarantee term must be constraint. Listing 4-6 illustrates such a constraint.

```
<wsag:Item>
        <wsag:Location>//wsag:GuaranteeTerm/wsag:ServiceLevelObjective/wsag:KPITarget/wsag:CustomServiceLevel
                /ext:ReservationTime
        </wsag:Location>
        <wsag:ItemConstraint>
                <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
                        <xs:union>
                                <xs:simpleType>
                                        <xs:restriction base="xs:dateTime">
                                                <xs:minInclusive value="2007-08-10T17:00:00+02:00" />
```

```
                                        <xs:maxInclusive value="2007-08-10T123:00:00+02:00" />
                                    </xs:restriction>
                                </xs:simpleType>
                                <xs:simpleType>
                                    <xs:restriction base="xs:dateTime">
                                        <xs:minInclusive value="2007-08-10T23:00:00+02:00" />
                                        <xs:maxInclusive value="2007-12-24T18:00:00+02:00" />
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:union>
                        </xs:simpleType>
                    </wsag:ItemConstraint>
</wsag:Item>
```

Listing 4-6: Restriction of Valid Start Times

Mapping of reservation messages to the WS-Agreement protocol

According to deliverable [G2MPLS-ARCH] the overlay and the integrated approach use a two step mechanism to create reservations in order to co-allocate resources. Therefore, these two steps comprise are of the pre-reservation of resources and the commitment of a reservation. The overlay approach additionally comprises a negotiation phase, which takes place before the actual reservation of the resources. In this phase the content of the reservation, which will be created, is negotiated. Since the negotiation of agreements is not yet specified within the WS-Agreement specification, we need to extend the current standard with negotiation capabilities. In the next chapter the negotiation process and the required WS-Agreement extensions are described.

Negotiation of agreement templates

Before in a co-allocation scenario resources are actually reserved by creating SLAs, usually a negotiation process takes place in which feasible time frames are identified, where the required resources are available. In order to start the negotiation process a client queries a set of SLA templates from a resource provider via the WSRF *GetResourceProperties* method as defined in the WS-Agreement specification. From these templates, the client chooses the most suitable one as a starting point for the negotiation process. This template defines the context of the subsequent iterations. All subsequent offers must refer to this agreement template in order to enable the resource provider to validate the creation constraints of the original template.

After the client has chosen an agreement template, it is used to derive new template from it. The new template must contain a reference to the originating template within its context. Furthermore, the client may change the content of the new created template, namely the content of the service description terms, the service property terms, the guarantee terms, and the creation constraints. These changes must be done according to the

creation constraints defined in the original template. The creation constraints within the new created template must be more restrictive than the constraints within the originating template in order to assure that the negotiation process converges. They provide hints for the resource provider, within which limits the client is willing to create an agreement.

After the initiator created the new agreement template according to its requirements, the template is send to the responder via a negotiateTemplate message. When the responder has received a negotiateTemplate message, it must first check the validity of the refined template. This step includes (i) retrieving the original agreement template that was used to create the input document, (ii) validating the structure of the input document with respect to the originating template, and (iii) validating the changes of the content in the refined template with respect to the creation constraints defined in the originating template. Once this is done, the resource provider now checks whether the provided requested service could be provided as is or not. In the first case it just returns the agreement template to the client, indicating that an offer based on that template will potentially be accepted. In the latter case the provider creates one or more counter offers. If the service cannot be provided at all, no counter offer is returned. After the client receives the counter offers from the resource provider, it checks whether one meets its requirements. If there is no suitable template, the client either stops the negotiation process, or starts again from the beginning. If there is an applicable template, the client validates whether there is need for an additional negotiation step or not. If yes, it uses the selected template and proceeds with step negotiating another template; otherwise the selected one is used to create a new SLA.

Listing 4-7 defines the required extensions for the negotiation capabilities:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:wsa="http://www.w3.org/2005/08/addressing"
        xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
        targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement">

        <wsdl:types>
                <xs:schema
                        targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement"
                        xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
                        xmlns:wsa="http://www.w3.org/2005/08/addressing"
                        elementFormDefault="qualified" attributeFormDefault="qualified">

                        <xs:element name="negotiateTemplateInput"
                                type="wsag:NegotiateTemplateInputType" />

                        <xs:element name="negotiateTemplateResponse"
                                type="wsag:NegotiateTemplateOutputType" />

                        <xs:complexType name="NegotiateTemplateInputType">
                                <xs:sequence>
```

```
                                        <xs:element ref="wsag:AgreementOffer" />
                                        <xs:element name="NoncriticalExtension"
                                                type="wsag:NoncriticalExtensionType" minOccurs="0"
                                                maxOccurs="unbounded" />
                                        <xs:any namespace="##other" processContents="lax"
                                                minOccurs="0" maxOccurs="unbounded" />
                                </xs:sequence>
                        </xs:complexType>

                        <xs:complexType name="NegotiateTemplateOutputType">
                                <xs:sequence>
                                        <xs:element ref="wsag:AgreementOffer" />
                                        <xs:element name="NoncriticalExtension"
                                                type="wsag:NoncriticalExtensionType" minOccurs="0"
                                                maxOccurs="unbounded" />
                                        <xs:any namespace="##other" processContents="lax"
                                                minOccurs="0" maxOccurs="unbounded" />
                                </xs:sequence>
                        </xs:complexType>

                </xs:schema>
        </wsdl:types>

        <wsdl:message name="negotiateTemplateInputMessage">
                <wsdl:part name="parameters"
                        element="wsag:negotiateTemplateInput" />
        </wsdl:message>

        <wsdl:message name="negotiateTemplateOuputMessage">
                <wsdl:part name="parameters"
                        element="wsag:negotiateTemplateResponse" />
        </wsdl:message>

        <wsdl:message name="negotiateTemplateFaultMessage">
                <wsdl:part name="fault" element="wsag:ContinuingFault" />
        </wsdl:message>

        <wsdl:portType name="AgreementFactory">
                <wsdl:operation name="negotiateTemplate">
```

```
                        <wsdl:input message="wsag:negotiateTemplateInputMessage"
                                wsa:Action="http://schemas.ggf.org/graap/2007/03/ws-
agreement/NegotiateTemplateRequest" />
                        <wsdl:output message="wsag:negotiateTemplateOuputMessage"
                                wsa:Action="http://schemas.ggf.org/graap/2007/03/ws-
agreement/NegotiateTemplateResponse" />
                        <wsdl:fault name="ResourceUnknownFault"
                                message="wsrf-rw:ResourceUnknownFault" />
                        <wsdl:fault name="ContinuingFault"
                                message="wsag:negotiateTemplateFaultMessage" />
                </wsdl:operation>
        </wsdl:portType>

</wsdl:definitions>
```

Listing 4-7: WS-Agreement Negotiation Extension

Agreement creation

When multiple resources are co-allocated, the different resources are reserved by creating SLAs which guarantee that the requested resources are available at a specific time. In principle two different approaches exist to create these SLAs:

1. use transactions to create the SLAs, or
2. create each SLA within one step.

When using transactions to co-allocate SLAs (e.g. by a two-phase-commit protocol), the SLAs are created in two steps. First a prepare message is send to the provider, by which the required resources are pre-reserved for a certain time frame. The prepared SLAs must be committed during this time frame. In order to co-allocate SLAs, first all SLAs are prepared, the commit messages are sent to the resource providers. By that the SLAs are finally created and the co-allocated resources are finally reserved. Since the current WS-Agreement specification foresees the creation agreements in only one step, an extension of the existing standard would be needed for this approach, which needs to be defined.

Another way of solving the problem is to create agreements within one phase and to allow users to terminate the agreement within a defined time period without penalty. When the agreement is created, the required resources are reserved. From a functional point of view, this reservation is comparable with the pre-reservation of resources of the two-phase-commit approach. The main difference between these approaches is that in the first one a client has to explicitly commit a reservation, while in the latter one this is already done implicitly when creating the agreement. Therefore, creating an agreement within one step has two obvious advantages, it uses

the current WS-Agreement standard and it keeps the protocol stack simple. The message definitions to create agreements can be found in the Agreement Factory Port Type definition of the WS-Agreement specification [WSAG-SPEC]. The required messages are shown in Listing 4-8.

```
<wsdl:message name="CreateAgreementInputMessage">
    <wsdl:part name="parameters"
                element="wsag:CreateAgreementInput" />
    </wsdl:message>

<wsdl:message name="CreateAgreementOuputMessage">
    <wsdl:part name="parameters"
                element="wsag:CreateAgreementResponse" />
</wsdl:message>

<wsdl:message name="CreateAgreementFaultMessage">
    <wsdl:part name="fault"
                element="wsag:ContinuingFault"/>
</wsdl:message>
```

Listing 4-8: Agreement Creation Messages

To terminate agreements the terminate operation of the Agreement port type [WSAG-SPEC] is used. The required messages are shown in Listing 4-9.

```
<wsdl:message name="TerminateInputMessage">
    <wsdl:part name="parameters" element="wsag:TerminateInput" />
</wsdl:message>

<wsdl:message name="TerminateOuputMessage">
    <wsdl:part name="parameters" element="wsag:TerminateResponse" />
</wsdl:message>
```

Listing 4-9: Message Definitions for Terminating Agreements

To monitor the created agreements the GetResourceProperty operation of the WSRF framework is used [WSRP-SPEC]. The messages related to the GetResourceProperty operation are shown in Listing 4-10.

```
<wsdl:message name="GetResourcePropertyRequest">
    <wsdl:part name="GetResourcePropertyRequest"
```

```
            element="wsrf-rp:GetResourceProperty" />
</wsdl:message>

<wsdl:message name="GetResourcePropertyResponse">
    <wsdl:part name="GetResourcePropertyResponse"
                    element="wsrf-rp:GetResourcePropertyResponse" />
</wsdl:message>

<wsdl:message name="InvalidResourcePropertyQNameFault">
    <wsdl:part name="InvalidResourcePropertyQNameFault"
                    element="wsrf-rp:InvalidResourcePropertyQNameFault" />
</wsdl:message>

<wsdl:message name="ResourceUnknownFault">
    <wsdl:part name="ResourceUnknownFault"
                    element="wsrf-rw:ResourceUnknownFault" />
</wsdl:message>

<wsdl:message name="ResourceUnavailableFault">
    <wsdl:part name="ResourceUnavailableFault"
                    element="wsrf-rw:ResourceUnavailableFault" />
</wsdl:message>
```

Listing 4-10: GetResourceProperty Message Definitions

4.10 G.OUNI policy and security

4.10.1 Policy Based Access Control and Policy Enforcement in G.OUNI

G.OUNI specification discussed in this document suggests a number protocols and mechanisms that will support Network and Grid services over the G.OUNI. Besides the generic UNI transport network services, such as connection creation, connection deletion, and connection status enquiry, the G.OUNI also allows offering Grid services that can be both network service over Grid middleware/interface and co-allocated Grid services.

Both network and Grid services are subject to policy based access control that includes both Authentication (AuthN) and Authorisation (AuthZ). AuthN is typically defined by user client software and AuthZ is more application specific and allows using application specific/defined policy. It is also required that AuthZ can be

done only for authenticated user. The following analysis and design considerations are provided for the policy based access control.

For the G^{2}MPLS that uses Grid middleware for creating Grid enabled NCP, AuthN/AuthZ services can be applied at both layers Grid middleware and G.OUNI. Grid Security Infrastructure (GSI) and Grid middleware can provide extended functionality for secure user admission and job submission. However, it also suggests existence of local/internal to the resource, access control service, in our case Network Element with the G.OUNI front-end providing at least a relevant policy enforcement mechanism that enforce upper layer Grid middleware AuthZ decisions at the resource level. Example can be a case when Grid middleware AuthZ service authorise a user with some VO related attributes (like project and group) and allows him/her a specific group or privilege level account available at the network resource. Next, this assigned attribute or privilege level need to be checked at the G.OUNI to provide a user requested/allowed level/quality of service.

Although the type of policy and used access control mechanisms can be different at the Grid middleware and the G.OUNI the overall G^{2}MPLS security and access control infrastructure should provide consistent access control and policy enforcement and management. All these issues are subject to research and development in the WP4 AAA package.

Below discussion is focussed on the G.OUNI access control services that should address the following issues:

- provide consistent access control according to local policy;
- enforce upper layer access control decisions;
- allow call-outs to site or domain central AuthZ service.

4.10.2 Policy and Security considerations in the OIF UNI Specification

OIF User Network Interface (UNI) 1.0 release 2 Signalling Specification [OIF-UNI1.0R2-COMM] provides UNI policy and security consideration. The UNI specification requires that the transport network must provide appropriate mechanisms to ensure accurate and authorised usage of network resources and client accountability. Access control and accounting should be governed according to the policy. Policy typically contains a collection of rules and/or conditions according to which the resources can be accessed or used. It is also suggested that policy based access control and usage should provide necessary information for accounting network related user activity.

As described in the UNI specification, policy rules should define conditions on parameters such as source and destination address, priorities, bilateral agreements between service providers, time or cost constrains, etc., -

which are all transport network related. Provided example suggests treating user groups as user related attributes in a form like this: “Approve all requests on behalf of a given user group received from a given UNI-C agent, if identity of the requestor can be verified”.

However, for Grid based collaborative applications in research and high-tech industry network access is typically bound to the project based user and resource association such as Virtual Organisation (VO) or Virtual Laboratory (VL) and depend on user attributes. User attributes are typically maintained by user Home Organisation (HO) or VO attribute service such as VOMS. It is also important that network resources are provided depending on the type of application that will use them. When integrated with the Grid resources or services access control, there is a need to make Grid and network resource access control policies compliant or compliant and compatible and possibly use the same access control tools and policy format.

UNI specification suggests using simple Policy Based Access Control (PBAC) model originated from the ITU-T Access Control Framework (X.812-ACF) defined in the standard T-REC-X.812-199511 [ITU-T-X.812] and actually compliant with the Generic AAA AuthZ Framework [IETF-RFC2903, IETF-RFC2904]. The suggested PBAC model consist of two major modules a Policy Enforcement Point (PEP), typically located at the Network Node/Element (such as an OXC), and a Policy Decision Point (PDP) that may be placed together with the PEP or separately depending on the complexity and frequency of events that require policy decisions. PDP may use different local and external policy repositories often referred as Policy Authority Point (PAP). UNI specification also suggests using COPS protocol as policy decision enforcement mechanism if an external PDP is used.

The UNI specification describes few example policies applicable to optical network connection provisioning: time-of-day based provisioning; identity and credit verification for connection requestor; and usage based accounting. Although these example can be considered as typical and basic, their implementation require more complex functionality and infrastructure when just simple PEP-PDP interaction. This is especially related to providing support to access control session/state management [DEMCHENKO], COPS policy enforcement mechanism [IETF-RFC2748], and usage control [ZHANG].

4.10.3 Proposed GAAA-AuthZ solution for policy based access control and enforcement at UNI

Recent developments in the GAAA-AuthZ framework for Complex Resource Provisioning (CRP) [DEMCHENKO] and Token Based Networking (TBN) [GOMMANS] provide a number of solutions and mechanisms to resolve the problems mentioned above, in particular:

- AuthZ session management to support complex AuthZ decision and multiple resource access, including multiple resources belonging to different administrative and security domains.
- AuthZ tickets with extended functionality to support AuthZ session management, delegation and obligated policy decisions.
- Authorisation and reservation tokens as policy enforcement mechanisms that can be used in the G^{2}MPLS Control plane and in-band.
- Policy Obligations handling model to support usable/accountable resource access/usage and additionally global and local user account mapping widely used in Grid based applications and supercomputing.

When considered for supporting Grid resources access over G.OUNI, AuthZ service may be required to support also AuthZ session termination as a part of the resource release process/sequence.

Although the above listed functionalities can be implemented under extended PEP or PDP functionality, such approach would significantly limit the AuthZ service flexibility and potentially affect the interoperability of different implementations as the discussed functionalities require agreement on a number of protocol, messaging format and attribute semantics.

The proposed in the GAAA-AuthZ framework solution is based on using such structural components and solutions as Token Validation Service (TVS), Obligation Handling Reference Model (OHRM), and XACML policy profile for OLPP.

Figure 4-9 illustrates the major GAAA-AuthZ modules and how they interact when evaluating Service Request.

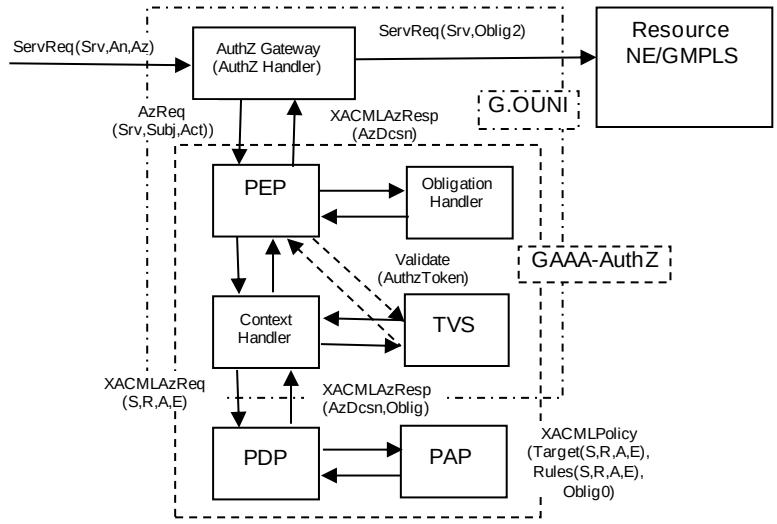

Figure 4-9: GAAA-AuthZ components providing Service Request evaluation

Authorisation service components GAAA-AuthZ are called from the G.OUNI via AuthZ gateway that intercepts service request ServiceRequest (ServiceId, AuthN, AuthZ) containing service name (and variables if necessary) and AuthN and AuthZ attributes. The AuthZ Gateway (that can be just interceptor process calling from the service or application) extracts necessary information and sends the AuthZ request AuthzRequest (ServiceId, Subject, AuthZ), containing service name ServiceId, requestor's identification Subject and AuthZ attributes AuthZ, to the PEP. The major PEP's task is to convert the AuthZ request's semantics into the PDP request which semantics are defined by the used policy. When using XACML policy and PDP, PEP will send the XACML AuthZ request to the PDP in the format (Subject, Resource, Attributes, (Environment)). If in the general case the XACML policy contains Obligations, they are returned in the XACMLAzResponse (AuthzDecision, Obligations). PEP calls the Obligation Handler to process the Obligations which are defined as actions to be taken on the policy decision or in conjunction with the Service access (like account mapping, quota enforcing, logging, or accounting).

If the Service request contains the AuthZ token that may reference a local or global reservation ID, or just identify the AuthZ session in which context the request is sent, token validation is performed by the Token Validation Service (TVS). TVS is typically called from the PEP and returns confirmation on whether the token is valid. Separating TVS as a different function or service allows creating a flexible token and/or a ticket policy enforcement infrastructure for on-demand network resource provisioning.

The GAAA-AuthZ architecture for multidomain Optical Network Resource Provisioning is being developed in the framework of the PHOSPHORUS WP4 as GAAA Toolkit [AAA-ARCH].

4.11 G.OUNI abstract messages and procedures

The G.OUNI signalling and routing messages are described in this section. These messages are denoted "abstract" since the actual realisation depends on the protocol used. Sections 4.11.4.2, 4.13 and 4.14 describe G^{2}MPLS extensions for RSVP, LMP and OSPF accordingly. In the description below, the terms "initiating G.OUNI-C" and "terminating G.OUNI-C" are used to identify the entities at the two end of a GNS service that initiate and terminate a given action. The initiating G.OUNI-C for a given signalling or routing action does not necessarily need to be the G.OUNI-C that originally requested the GNS service. For instance in the PHOSPHORUS indirect model presented in section 4.4, the initiating G.OUNI-C for a GNS service and connection is different from the initiating G.OUNI-C for the original GNS service and connection request.

The following abstract messages are currently defined.

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated	Message Direction
G.OUNI standardised OIF messages				
1.	NS Create Request	S	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
2.	NS Create Response	S	O/I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N
3.	NS Create Confirmation	S	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
4.	NS Delete Request	S	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
5.	NS Delete Response	S	O/I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N
6.	NS Status Enquiry	S	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
7.	NS Status Response	S	O/I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated	Message Direction
8.	NS Notification	S	O/I	G.OUNI-N ➔ G.OUNI-C
Grid Network Service (GNS) abstract messages required to support G^{2}MPLS NCP				
9.	Grid service capability	R	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
10.	Grid resource availability	R	O/I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
11.	Network resource availability	R	O/I	G.OUNI-N ➔ G.OUNI-C
12.	Network topology information	R	O/I	G.OUNI-N ➔ G.OUNI-C
13.	Network end-point assigned addresses (TNAs)	R	O/I	G.OUNI-N ➔ G.OUNI-C
14.	GNS capabilities	R	I	G.OUNI-N ➔ G.OUNI-C
15.	Grid resource allocation request	S	I	G.OUNI-N ➔ G.OUNI-C
16.	GNS Create Request	S	I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
17.	GNS Create Response	S	I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N
18.	GNS Create Confirmation	S	I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
19.	GNS Delete Request	S	I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C
20.	GNS Delete Response	S	I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N
21.	GNS Status Enquiry	S	I	G.OUNI-C ➔ G.OUNI-N G.OUNI-N ➔ G.OUNI-C

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated	Message Direction
22.	GNS Status Response	S	I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N
23.	GNS Notification	S	I	G.OUNI-N ➔ G.OUNI-C G.OUNI-C ➔ G.OUNI-N

Table 4-6: G.OUNI Messages

Table 4-6 depicts messages that can be either signalling messages (1-8 and 12-20) which have to do with the Grid and network reservation establishment, or routing messages (9-14) which have to do with the Grid and Network service discovery. Messages 1-13 are supported by both overlay and integrated model, whereas messages 14-23 are supported only on the integrated model. This implies that the G^{2}MPLS integrated model should be backward compatible with the G^{2}MPLS overlay model in terms of the G.OUNI messages as interoperability between domains supporting those two different models can take place.

A list of G.OUNI attributes is associated with each G.OUNI message; some are mandatory for a given signalling message and some optional. All these are described in the following sections. Furthermore, the G^{2}MPLS NCP may not support all of the requested attributes and in this case appropriate indications to the client in response messages should be sent. The manner in which the G.OUNI abstract messages are mapped to actions within the G^{2}MPLS NCP and transport network are outside the scope of this deliverable. The same stands for the signalling protocol used within the transport network to realise such actions.

4.11.1 Network Service (NS) abstract signalling messages

The first eight abstract messages listed on Table 4-6 represent the standardised signalling messages described in [OIF-UNI1.0R2-COMM] where the more generic term “Network Service (NS)” appears there as “Connection”. To avoid repetition of already standardised messages only a brief description is provided here. Attributes of the messages are described in [OIF-UNI1.0R2-COMM].

1. The *NS create request* is used by the Client (Grid MW) to request a connection between specified source and destination clients (Grid user/applications/resources). The *NS Create Request* is sent from:
    a. The initiating G.OUNI-C to G.OUNI-N to request the creation of a connection;
    b. The G.OUNI-N to the terminating G.OUNI-C to indicate an incoming connection request.

2. The *NS create response* is used to acknowledge the establishment of the NS (connection) to the G.OUNI-C that initiated the NS request. The corresponding client may then start transmission of data on the established connection. The *NS create Response* is sent from*:*

    a. The terminating G.OUNI-C to G.OUNI-N to accept or reject an incoming NS create request;
    b. The G.OUNI-N to the initiating G.OUNI-C to indicate the success or failure of the NS requested previously.

3. The *NS create confirmation* is used to acknowledge the establishment of the NS (connection) to the G.OUNI-C that terminated the connection create request and after that the corresponding client may start transmission of data on the established connection. The *NS create confirmation* is sent from:

    a. The initiating G.OUNI-C to G.OUNI-N to acknowledge completion of the connection establishment;
    b. The G.OUNI-N to the terminating G.OUNI-C to indicate that the connection has been successfully created and that the corresponding client may start transmitting data over the connection.

4. The *NS delete request* is used to indicate the deletion of a connection. If the NS (connection) deletion is being initiated by a G.OUNI-C, then this G.OUNI-C should maintain the connection control stat and the corresponding client should maintain the data plane until after the NS deletion has been acknowledged. This avoids alarms being generated by the client at the other end of the connection. The G.OUNI-C terminating the NS deletion request may delete the NS state upon receipt of the NS delete request. The network may also delete an NS (a forced deletion) due to internal network failures or G.OUNI-N timeout of a deletion response. In that case a NS delete response is not required and the G.OUNI-N may terminate all NS control state and the data path. The *NS delete request* is sent from:

    a. The initiating G.OUNI-C to G.OUNI-N to delete a NS;
    b. The G.OUNI-N to the terminating G.OUNI-C to indicate deletion by other end;
    c. The G.OUNI-N to a G.OUNI-C to indicate the deletion of a connection by the network.

5. The *NS delete response* message signals the completion of the NS deletion procedure. The *NS delete response* is sent from:

    a. The terminating G.OUNI-C to G.OUNI-N to acknowledge an incoming NS delete request;
    b. The G.OUNI-N to the initiating G.OUNI-C to indicate the successful deletion of the NS as requested.

6. The *NS status enquiry* is used to query the state and attributes of a given NS. The *NS status enquiry* is sent from:

    a. The initiating G.OUNI-C to G.OUNI-N to enquire about the status and/or attributes of the NSs owned by G.OUNI-C or from G.OUNI-N to G.OUNI-C.

7. The *NS status response* returns the status of the specified NS and associated attributes. The *NS status response* is sent from*:*

    a. The G.OUNI-N to the G.OUNI-C and vice versa to indicate the status of NS attributes as requested before.

8. The *NS notification* message is sent autonomously by a G.OUNI-N to either G.OUNI-C to indicate a change in the status of the connection*.*

4.11.2 Grid Network Service (GNS) abstract messages

After describing the messages already supported by [OIF-UNI1.0R2-COMM] which are the base line for the GMPLS NCP new abstract messages are required to support Grid Network Services (GNS) on the PHOSPHORUS G^{2}MPLS overlay and integrated model. The attributes carried in these messages are briefly described here since sections 4.12 on RSVP-TE extensions and 4.13 on OSPF extensions give the whole picture. Some of the messages are described as routing and some others as signalling since they perform different functions. In addition, these messages may be supported by either both overlay and integrated models or only on integrated one.

9. The *Grid Service Capability* indicates the capabilities of the grid services attached to the G.OUNI either by services supported by the Grid MW or applications (e.g. meta-scheduling, indexing, data mining data storage, visualization, etc.) and resource capabilities (e.g. CPU, Storage, etc.). GLUE Schema describes the service and resource attributes that will be mapped to G.OUNI messages and are presented in section 4.13. The message is sent only once from G.OUNI-C to G.OUNI-N. On the overlay model this message is carried over G^{2}MPLS NCP from one Grid MW to others. In the integrated model though these messages are sent from G.OUNI-C to G.OUNI-N towards G^{2}MPLS NCP. The way that G^{2}MPLS NCP manipulates them is not within the scope of this deliverable. The *Grid Service Capability* is sent from:

    a. The G.OUNI-C to G.OUNI-N to indicate the service/resource capabilities of the Grid end point attached to G.OUNI-C (supported both in overlay and integrated model);
    b. The G.OUNI-N to G.OUNI-C indicate the service/resource capabilities of the Grid end point attached to initiated G.OUNI-C to a remote Grid MW (supported in integrated model).

10. The *Grid Resource Availability* is used to describe the availability of the Grid resources to G.OUNI-C (e.g. CPU, Storage, Memory, OS, etc.). The attributes are based on GLUE and mapped to G.OUNI protocol extensions. *Grid resource availability* may provide information about Grid site resources availability calendars (see section 4.11.4.2). A resource calendar, can be a time ordered list of pairs, where a pair contains values:

- a time when resource availability is changing,
- a new resource availability value.

The *Grid Resource Availability* is sent from:

a. The G.OUNI-C to G.OUNI-N to indicate the resource availability of the Grid site attached to G.OUNI-C (supported both in overlay and integrated model);
b. The G.OUNI-N to G.OUNI-C indicate the service capabilities of the Grid site attached to initiated G.OUNI-C to a remote Grid MW (supported in integrated model).

11. The *Network Resource Availability* is used to provide information of network resource availabilities to Grid MW and make it aware of the current recourse status. This way the Grid MW has the ability to calculate an end-to-end path and can be provided to the Grid MW that can support such function. The *Network Resource Availability* is sent from:

    a. The G.OUNI-N to G.OUNI-C indicates the current status of network availability to G.OUNI-C towards Grid MW.

12. The *Network Topology Information* is used to provide limited and virtualised topological information of transport network environment. G^{2}MPLS NCP has the authority to provide any topological information to Grid MW and this can be based upon SLA agreements. The *Network Topology Information* is sent from:

    a. The G.OUNI-N to G.OUNI-C indicates the current status of network availability to G.OUNI-C towards Grid MW.

13. The *Network End-point Assigned Addresses* (TNAs) are used to provide network end point information from G.OUNI-N to G.OUNI-C. This information is the minimum level of information that must be passed from G^{2}MPLS to Grid MW. *The Network End-point Assigned Addresses* (TNAs) message is sent from:

    a. The G.OUNI-N to G.OUNI-C indicates the transport end-point information to G.OUNI-C towards Grid MW.

14. The *GNS Capabilities* determine the Grid Network services supported by G^{2}MPLS under the integrated model. GNS services such as protocol extensions to support JSDL or GLUE schema or even co-allocation of Grid and Network resources, Grid and Network Resource Discovery and Reservation can be passed from G.OUNI-N to G.OUNI-C. This way the Grid MW is informed of the services supported by G^{2}MPLS and can act accordingly. The *GNS Capabilities* message is sent from:

    a. The G.OUNI-N to G.OUNI-C indicates the services supported by G^{2}MPLS NCP to G.OUNI-C towards Grid MW.

15. The *Grid Resource Allocation Request* is forwarded from G.OUNI-N to G.OUNI-C towards the remote Grid site in order to allocate Grid resources as an outcome of discovery and reservation of Grid and Network resources of G^{2}MPLS NCP under the integrated model. The *Grid Resource Allocation Request* message is sent from:

    a. The G.OUNI-N to G.OUNI-C indicates the services supported by G^{2}MPLS NCP to G.OUNI-C towards Grid MW.

16. The *GNS Create Request* is used by the Grid MW to request a Grid Network Service from the G^{2}MPLS NCP under integrated model. The message is carried from G.OUNI-C to G.OUNI-N toward G^{2}MPLS NCP. It carries JSDL attributes as described in detail in section 4.11.4.2. The GNS create request is sent from

    a. The initiating G.OUNI-C to G.OUNI-N to request the creation of a GNS;
    b. The G.OUNI-N to G.OUNI-C to indicate an incoming GNS request towards a remote Grid site.

17. The *GNS Create Response* is used to acknowledge the establishment of Grid Network Service by the G^{2}MPLS NCP. The corresponding Client may then start transmission of data through the established connection to the allocated Grid resources. The *GNS Create Response* is sent from

    a. The G.OUNI-C to G.OUNI-N to accept or reject an incoming GNS request;
    b. The G.OUNI-N to the initiating G.OUNI-C to indicate the success or failure of the GNS requested previously.

18. The *GNS Create Confirmation* is used to acknowledge the establishment of the GNS response to either G.OUNI-C that is attached to Grid MW that sent the GNS create response (in case of overlay model) or to G.OUNI-N (in case of integrated model). The *GNS Create Confirmation* is sent from:

    a. The terminating G.OUNI-C to G.OUNI-N to acknowledge completion of the GNS establishment;
    b. The G.OUNI-N to the terminating G.OUNI-C to indicate the success of the GNS establishment.

19. The *GNS Delete Request* is used to indicate the deletion of a GNS. The GNS deletion can be initiated by G.OUNI-C in case of failure on Grid MW or Grid site. Then this G.OUNI-C should maintain the connection control state and the corresponding client should maintain the data plane until after the GNS deletion has been acknowledged. The G.OUNI-C terminating the GNS deletion request may delete the GNS state upon receipt of the GNS delete request. The G^{2}MPLS NCP may also delete a NS (a forced deletion) due to internal network failures or G.OUNI-N timeout of a deletion response. The *GNS Delete Request* is sent from:

    a. The initiating or terminating G.OUNI-C to G.OUNI-N to delete a GNS;
    b. The G.OUNI-N to the initiating or terminating G.OUNI-C to indicate deletion by other end;

c. The G.OUNI-N to a G.OUNI-C to indicate the deletion of a GNS by the network and escalate the deletion information to Grid MW or Grid site.

20. The *GNS Delete Response* signals the completion of the GNS deletion procedure both from the G^{2}MPLS NCP and any involved Grid MW and Grid site. The *GNS Delete Response* is sent from:

    a. The terminating G.OUNI-C to G.OUNI-N to acknowledge an incoming GNS delete request;
    b. The G.OUNI-N to the initiating G.OUNI-C to indicate the successful deletion of the GNS as requested.

21. The *GNS Status Enquiry* is used to query the state and attributes of a given GNS. The *GNS Status Enquiry* is sent from:

    a. The initiating G.OUNI-C to G.OUNI-N to enquire about the status and/or attributes of the GNSs owned by G.OUNI-C or from G.OUNI-N to G.OUNI-C.

22. The *GNS Status Response* returns the status of the specified GNS and associated attributes. The *GNS Status Response* is sent from:

    a. The G.OUNI-N to the G.OUNI-C and vice versa to indicate the status of GNS attributes as requested before.

23. The GNS Notification message is sent autonomously by a G.OUNI-N to either G.OUNI-C to indicate a change in the status of the GNS.

In this section, relevant signalling, routing and discovery aspects are considered for supporting Grid extensions.

4.11.3 Signalling

Three main aspects of Grid extensions should be covered for G.OUNI signalling mechanism support:

- Extensions for Grid job/service requests (JSDL Version 1.0, GFD-R.056)
  - Support of advance reservations (service time schedules in the signalling messages)
- Extensions to integrate Grid AAA and Security means with the network related means
- Extensions to support internal Grid service/resource failures (e.g. for job recovery purposes)
  - new extended set of Error Codes and Error Values is required

As a result of JSDL v1.0 Specification [JSDL-SPEC] analysis a set of five core groups of additional Grid elements for G^{2}MPLS signalling protocols was appointed:

- **Job Structure Elements** = {JobDefinition Element, JobDescription Element, Description Element}

- **Job Identity Elements** = {JobIdentification Element, JobName Element, JobAnnotation Element, JobProject Element}
- **Application Elements** = {Application Element, ApplicationName Element, ApplicationVersion Element}
- **Resource Elements** = {Resources Element, CandidateHosts Element, HostName Element, FileSystem Element, MountPoint Element, MountSource Element, DiskSpace Element, FileSystemType Element, ExclusiveExecution Element, OperatingSystem Element, OperatingSystemType Element, OperatingSystemName Element, OperatingSystemVersion Element, CPUArchitecture Element, CPUArchitectureName Element, IndividualCPUSpeed Element, IndividualCPUTime Element, IndividualCPUCount Element, IndividualNetworkBandwidth Element, IndividualPhysicalMemory Element, IndividualVirtualMemory Element, IndividualDiskSpace Element, TotalCPUTime Element, TotalCPUCount Element, TotalPhysicalMemory Element, TotalVirtualMemory Element, TotalDiskSpace Element, TotalResourceCount Element, Additional Resources}
- **Data Staging Elements** = {DataStaging Element, FileName Element, FileSystemName, Element, CreationFlag Element, DeleteOnTermination Element, Source Element, URI Element, Target Element}

Each single element from those five groups must be supported by the signalling protocol to make the G^{2}MPLS compatible with JSDL v1.0 specification. There are also some extensions defined for this core set of elements but they are not mandatory and can be implemented as an option.

According to the specification, each job is described by XML structured document and all JSDL elements can be divided into two sets. In the first set there are elements that define XML hierarchy and structure of the document and the second set contains elements with the essential values and only those elements have to be transported by the signalling protocol.

```
<JobDefinition>
        <JobDescription>
                <JobIdentification >
                        <JobName> xsd:string </JobName>?
                        <Description> xsd:string </Description>?
                        <JobAnnotation> xsd:string </JobAnnotation>*
                        <JobProject> xsd:string </JobProject>*
                </JobIdentification>

                <Application>
                        <ApplicationName> xsd:string </ApplicationName>?
                        <ApplicationVersion> xsd:string </ApplicationVersion>?
                        <Description> xsd:string </Description>?
```

```
</Application>

<Resources>
        <CandidateHosts>
                <HostName> xsd:string </HostName>+
        </CandidateHosts>?
        <FileSystem name=”xsd:NCName”>
                <Description> xsd:string </Description>?
                <MountPoint> xsd:string </MountPoint>?
                <MountSource> xsd:string </MountSource>?
                <DiskSpace> jsdl:RangeValue_Type </DiskSpace>?
                <FileSystemType> jsdl:FileSystemTypeEnumeration </FileSystemType>?
        </FileSystem>*
        <ExclusiveExecution> xsd:boolean </ExclusiveExecution>?
        <OperatingSystem>
                <OperatingSystemType>
                        <OperatingSystemName>
                                jsdl:OperatingSystemTypeEnumeration
                        </OperatingSystemName>
                </OperatingSystemType>?
                <OperatingSystemVersion> xsd:string </OperatingSystemVersion>?
                <Description> xsd:string </Description>?
        </OperatingSystem>?
        <CPUArchitecture>
                <CPUArchitectureName>
                jsdl:ProcessorArchitectureEnumeration
                </CPUArchitectureName>
        </CPUArchitecture>?
        <IndividualCPUSpeed> jsdl:RangeValue_Type </IndividualCPUSpeed>?
        <IndividualCPUTime> jsdl:RangeValue_Type </IndividualCPUTime>?
        <IndividualCPUCount> jsdl:RangeValue_Type </IndividualCPUCount>?
        <IndividualNetworkBandwidth>
        jsdl:RangeValue_Type
        </IndividualNetworkBandwidth>?
        <IndividualPhysicalMemory>
        jsdl:RangeValue_Type
        </IndividualPhysicalMemory>?
        <IndividualVirtualMemory> jsdl:RangeValue_Type </IndividualVirtualMemory>?
        <IndividualDiskSpace> jsdl:RangeValue_Type </IndividualDiskSpace>?
```

```
                    <TotalCPUTime> jsdl:RangeValue_Type </TotalCPUTime>?
                    <TotalCPUCount> jsdl:RangeValue_Type </TotalCPUCount>?
                    <TotalPhysicalMemory> jsdl:RangeValue_Type </TotalPhysicalMemory>?
                    <TotalVirtualMemory> jsdl:RangeValue_Type </TotalVirtualMemory>?
                    <TotalDiskSpace> jsdl:RangeValue_Type </TotalDiskSpace>?
                    <TotalResourceCount> jsdl:RangeValue_Type </TotalResourceCount>?
            </Resources>?

            <DataStaging name=”xsd:NCName”?>
                    <FileName> xsd:string </FileName>
                    <FileSystemName> xsd:NCName </FileSystemName>?
                    <CreationFlag> jsdl:CreationFlagEnumeration </CreationFlag>
                    <DeleteOnTermination> xsd:boolean </DeleteOnTermination>?
                    <Source>
                            <URI> xsd:anyURI </URI>?
                    </Source>?
                    <Target>
                            <URI> xsd:anyURI </URI>?
                    </Target>?
            </DataStaging>*
        </JobDescription>
</JobDefinition>
```

Listing 4-11: Pseudo JSDL XML Schema

Most of the element tags are unique and they have well defined place in the XML structure (Listing 4-11), but there are two tags `<Description>` and `<URI>` that can occur in several places. Tag `<Description>` occurs four times in different parts of the XML schema so four different objects need to be created to make the transformation process easier. Tag `<URI>` occurs two times so two objects instances need to be created. The structure of each of the tags is exactly the same and the name and the identifier defines the position of the particular tag in the XML structure.

4.11.4 Routing & Discovery

In GMPLS, the routing protocol is used to distribute information on the Transport Plane topology and its capabilities within a single GMPLS domain and between different GMPLS domains. Each GMPLS node advertises attributes of local node and all local TE-links [IETF-RFC3630]. All this information is flooded within a

routing area and all nodes in the area synchronize their databases. PHOSPHORUS is proposing an alternative employment of the routing protocol. It can be used for G.OUNI messages transporting between G.OUNI-C and G.OUNI-N:

- Grid Service Capability,
- Grid Resource Availability,
- Network Resource Availability,
- Network Topology Information,
- Network End-point Assigned Addresses.

The G.OUNI Network related information is strictly related to the standard GMPLS routing deployment. However, G2MPLS Control Plane defines some new network's attributes:

- full-optical TE-link properties for enhanced constraint based path computation,
- TE-link bandwidth availability for enabling network resource advance reservation.

The G.OUNI Grid related information entails completely new entities in comparison to the GMPLS routing protocol. The one of most important work of PHOSPHORUS G^{2}MPLS NCP activity is to provide specifications for Grid related information.

4.11.4.1 *Grid Service Capability*

The Grid Service Capability specification is based on GLUE Schema specification [GLUE] which provides an abstract modelling for Grid resources [GLUE]. Due to large number of GLUE resources and attributes that can describe a single Grid site or node, a subset of these entities should be selected for implementation. The importance estimation can be done with usage of Job Submission Description Language [JSDL-SPEC] which defines a resource that jobs can request. For each grid resource set of attributes a dedicated object has to be created.

The [GLUE] paper specifies the several groups of Grid resources:

- Core Entities
  - Site
  - Service
- Computing Resources
  - Cluster
  - Computing element
  - Sub-cluster

  - Software
  - Host
- Storage Resources
  - Storage Element
  - Storage Area

[GLUE] defines the relationships between all the above components and exact resource types assigned to particular groups. From the perspective of PHOSPHORUS project the most important groups and candidate for initial implementation are Core Entities (mostly Site) and Computing Resource group. At this stage of the project additional validation should be performed to determine the value of elements from the Storage Resources group for G^{2}MPLS . It is assumed that the Storage Element and Storage Area should be taken into consideration. The target implementation should include all resources from the list above, however [GLUE] is not a final paper and is continuously changing. Therefore strong emphasis should be placed on tracking changes in case of publishing a new version of the document. A design and implementation work should be also focused on the most important resources, which are rather constant and not changing significantly during [GLUE] edition.

Table 4-7 depicts the identified parameters eligible to be mapped into G.OUNI protocols for Grid discovery purposes. More detailed analyse of the GLUE schema and the G^{2}MPLS Grid resource description can be found in [G2MPLS-EXT].

GLUE schema object	GLUE schema property	Description	Count	Data type
Site	Set of resources that are installed and managed by the same organization			
	UniqueID	Unique identifier of this site	1	string
	Name	Human-readable name	1	string
	Latitude	Degree the position of a place north or south of the equator	1	real32
	Longitude	Degree the position of a place east or west of Greenwich, England	1	real32
Service	An abstract, logical view of actual software components that should be formally defined in terms of the information exchanged between provider entity and requester entity			
	UniqueID	Unique identifier of this service	1	string

GLUE schema object	GLUE schema property	Description	Count	Data type
	Type	Service type	1	serviceType_t
	Version	Version of the service	1	string
	Endpoint	Network endpoint for this service	1	uri
	Status	Status of the service; Possible values: “OK”, “Warning”, “Critical”, “Unknown”, “Other”	1	serviceStatus_t
Computing Element	Service that manages jobs and offers them execution environments provided by computing resources			
	UniqueID	Unique identifier for this computing element	1	string
	InfoLRMSType	Type of the underlying local resource manager system	1	lrms_t
	InfoLRMSVersion	Version of the local resource management system	1	string
	InfoHostName	Host name of the machine running this service	1	string
	InfoGatekeeperPort	Gatekeeper port	1	int32
	InfoJobManager	The job manager used by the gatekeeper	1	string
	InfoDataDir	The path of a shared directory available for application data	1	string
	InfoDefaultSE	Unique identifier of the default Storage Element	1	string
	StateStatus	The queue status; Possible values: “Queuering”, “Production”, “Closed”, “Draining”	1	string
	StateRunningJobs	Number of jobs in running state	1	int32

GLUE schema object	GLUE schema property	Description	Count	Data type
	StateWaitingJobs	Number of jobs in waiting state	1	int32
	StateTotalJobs	Number of jobs in any state	1	int32
	StateEstimatedResponseTime	Estimated time to last for a new job	1	int32
	StateWorstResponseTime	Worst time from the job being accepted for execution	1	int32
	Stat.FreeJobSlots	Number of free job slots	1	int32
	PolicyMaxWallClockTime	Maximum wall clock time allowed	1	int32
	PolicyMaxObtainableWallClockTime	Maximum obtainable wall clock time	1	int32
	PolicyMaxCPUTime	Maximum CPU time allowed	1	int32
	PolicyMaxObtainableCPUTime	Maximum obtainable CPU time	1	int32
	PolicyMaxTotalJobs	Maximum allowed number of jobs in the CE	1	int32
	PolicyMaxRunningJobs	Maximum allowed number of jobs in running state in the CE	1	int32
	PolicyMaxWaitingJobs	Maximum number of jobs that can be in waiting state	1	int32
	PolicyPriority	Priority given to jobs in this CE	1	int32
	PolicyAssignedJobSlots	Number of slots for jobs to be in running state	1	int32
	PolicyMaxSlotsPerJobs	Maximum number of slots which could be allocated to a single job	1	int32
	PolicyPreemption	Job preemption enabler	1	boolean
	StateTotalJobs	Number of jobs in any state	1	int32
	StateFreeJobSlots	Number of free job slots	1	int32
SubCluster	Information about an homogenous set of hosts as regards a selected number of host			

GLUE schema object	GLUE schema property	Description	Count	Data type
	attributes			
	UniqueID	Unique identifier of the subcluster	1	string
	PhysicalCPUs	Total number of real CPUs in the subcluster	1	int32
	LogicalCPUs	Effective number of CPUs in the subcluster	1	int32
Software	Information about an installed software packages			
	LocalID	Identifier for the location	1	string
	Version	Version of the software package	1	string
	EnvironmentSetup	Fully qualified script for the setting of the application environment	1	string
Host	Summary description of the hosts part of the subcluster			
	OperatingSystemName	Name of the operating system	1	string
	OperatingSystemRelease	Release of the operating system	1	string
	OperatingSystemVersion	Version of the operating system	1	string
	ProcessorModel	Processor model as defined by the vendor	1	string
	ProcessorVersion	Processor version	1	string
	ProcessorVendor	Name of the processor vendor	1	string
	ProcessorClockSpeed	Clock speed (MHz)	1	int32
	ProcessorInstructionSet	Processor instruction set	1	string
	ProcessorOtherDescription	Other description for the processor	1	string
	MainMemoryRAMSize	Amount of RAM (MByte)	1	int32
	MainMemoryVirtualSize	Amount of Virtual Memory (RAM+Swap) in MB	1	int32

GLUE schema object	GLUE schema property	Description	Count	Data type
Storage Element	Abstraction for a storage resources			
	UniqueID	Unique identifier of the Storage Element	1	string
	Architecture	Underlying architectural system category	1	SEArch_t
	Status	Status of the whole Storage Element	1	SEStatus_t
	TotalOnlineSize	Total size of online storage space (GB)	1	int32
	TotalNearlineSize	Total size of nearline storage (GB)	1	int32
	UsedOnlineSize	Used online storage (GB)	1	int32
	UsedNearlineSize	Used nearline storage (GB)	1	int32
	Access Protocol	Protocols available to access/transport files in/from storage areas	n	accessProt_t
	Control Protocol	Protocol available for the control and/or management of the storage resource	n	ControlProt_t
Storage Area Property	Portion of storage extent to which a uniform set of policies applies			
	Name	Human-friendly name for the area	1	string
	Path	Full path of the root directory for this storage area	1	string
	TotalOnlineSize	Total online storage space (GB)	1	int32
	FreeOnlineSize	Free online storage space (GB)	1	int32
	ReservedOnlineSize	Reserved online storage space (GB)	1	int32

GLUE schema object	GLUE schema property	Description	Count	Data type
	TotalNearlineSize	Total nearline storage space (GB)	1	int32
	FreeNearlineSize	Free nearline storage space (GB)	1	int32
	ReservedNearlineSize	Reserved nearline storage space (GB)	1	int32
	RetentionPolicy	Vals: custodial, output, replica	1	retentionPol_t
	AccessLatency	Vals: online, nearline, offline	1	accessLat_t
	ExpirationMode	Vals: neverExpire, warnWhenExpired, releaseWhenExpired	1	expirationMode_t

Table 4-7: G^{2}MPLS Grid resource description objects and properties

4.11.4.2 *Grid Resource Availability*

The Grid Resource Availability extension must provide information on Grid site resource availability calendars (see Figure 4-10). A resource calendar, which simplest implementation is shown on Figure 4-11 can be a time ordered list of pairs, where a pair contains values:

- time when resource availability is changing,
- new resource availability value.

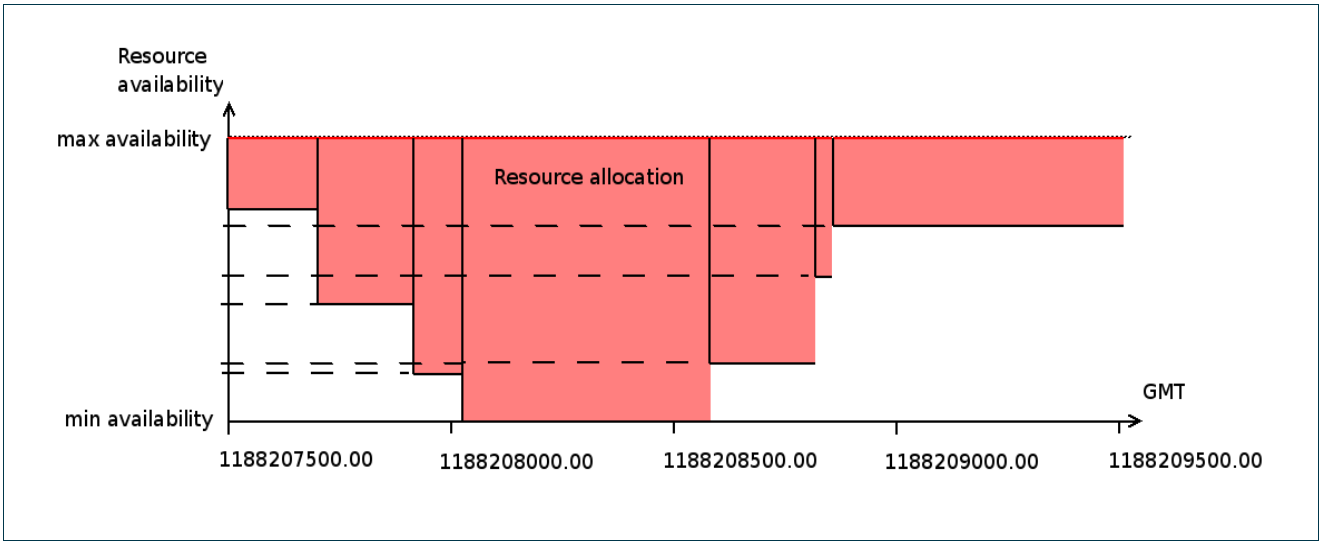

Figure 4-10: Resource availability schedule

The resource calendar for schedule presented on Figure 4-11 has representation:

ResourceCalendar = [(t1, a1), (t2, a2), (t3, a3), (t4, a4), (t5, a5), (t6, a6), (t7, a7)]

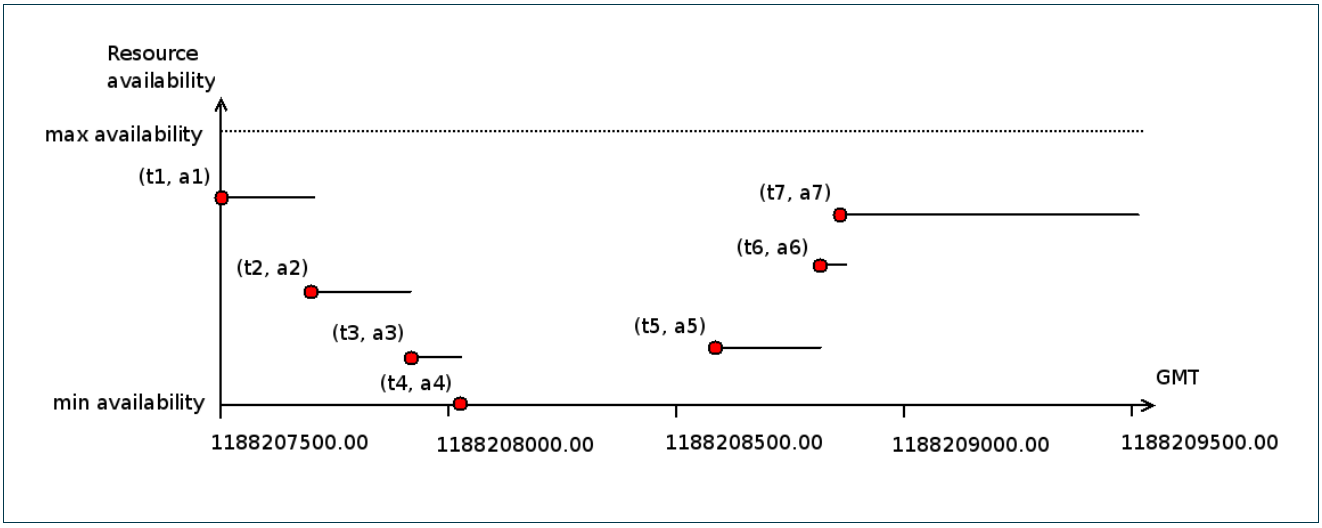

Figure 4-11: Resource availability calendar representation

4.12 RSVP-TE extensions

The main scope of the chapter is to identify the RSVP extensions necessary to satisfy requirements for a Grid Optical UNI signalling mechanisms. As a base for further Grid extensions, according to the deliverable [G2MPLS-ARCH], the OIF UNI RSVP-TE signalling protocol was chosen [OIF-UNI1.0R2-COMM, OIF-UNI1.0R2-RSVP].

JSDL elements types and mappings

To make transformation from JSDL elements into RSVP objects possible data types of all mandatory elements are need to be known and mapped. JSDL uses normative XML Schema simple types and additional it defines four enumeration types and one complex type for description of specific job requirements.

- Simple types – normative XML Schema types can be directly transformed:
  - `xsd:string` → String value
  - `xsd:NCName` → String value
  - `xsd:boolean` → Logical value (Boolean)
  - `xsd:double` → Floating point value (Double)
  - `xsd:anyURI` → String value
- Enumeration types – normative JSDL Names defined in specification; unique integer value will be assigned for each value, for each enumeration type:
  - `jsdl:ProcessorArchitectureEnumeration` - list of processor architecture types [JSDL-SPEC] section 5.2.1
  - `jsdl:FileSystemTypeEnumeration` - list of file system types [JSDL-SPEC] section 5.2.2
  - `jsdl:OperatingSystemTypeEnumeration` – list of operating system types [JSDL-SPEC] section 5.2.3
  - `jsdl:CreationFlagEnumeration` - list of flag types [JSDL-SPEC] section 5.2.4
- Complex type – JSDL defined complex type for all floating point values; new structure have to be defined for this JSDL type:
  - `jsdl:RangeValue_Type` – complex type [JSDL-SPEC] section 5.2.5, defined by the XML structure (Listing 4-12)

```
<UpperBoundedRange exclusiveBound="xsd:boolean"?>
        xsd:double
</UpperBoundedRange>?
<LowerBoundedRange exclusiveBound="xsd:boolean"?>
        xsd:double
```

```
</LowerBoundedRange>?
<Exact epsilon=”xsd:double”?>
        xsd:double
</Exact>*
<Range>
        <LowerBound exclusiveBound="xsd:boolean"?>
                xsd:double
        </LowerBound>
        <UpperBound exclusiveBound="xsd:boolean"?>
                xsd:double
        </UpperBound>
</Range>*
```

Listing 4-12: RangeValue_Type XML structure

```
typedef struct RangeValueType
{
        double upperBoundedRangeValue;
        boolean upperBoundedRangeExclusive;
        double lowerBoundedRangeValue;
        boolean lowerBoundedRangeExclusive;
        int exactSize;
        struct exact
        {
                double exactValue;
                double exactEpsilon;
        } exactsTab[exactSize];
        int rangeSize;
        struct range
        {
                double rangeLowerBoundValue;
                boolean rangeLowerBoundExclusive;
                double rangeUpperBoundValue;
                boolean rangeUpperBoundExclusive;
        } rangesTab[rangesSize]
} RangeValue;
```

Figure 4-12: Definition of the RangeValue type according to the XML jsdl:RangeValue_Type specification

All values of those types have to be encoded into messages according to the RSVP-TE objects specification and the types mapping procedure is required following Table 4-8.

JSDL Element	JSDL element type	RSVP Object name	Data type
<JobName>	xsd:string	JobName	String
<*JobIdentification*> <Description> </*JobIdentification*>	xsd:string	JobIdentificationDescription	String
<JobAnnotation>	xsd:string	JobAnnotation	String
<JobProject>	xsd:string	JobProject	String
<ApplicationName>	xsd:string	ApplicationName	String
<ApplicationVersion>	xsd:string	ApplicationVersion	String
<*Application*> <Description> </*Application*>	xsd:string	ApplicationDescription	String
<HostName>	xsd:string	HostName	String
<FileSystem name=””> *(tag attribute)*	xsd:NCName	FileSystemName	String
<*FileSystem*> <Description> </*FileSystem*>	xsd:string	FileSystemDescription	String
<MountPoint>	xsd:string	MountPoint	String
<MountSource>	xsd:string	MountSource	String
<DiskSpace>	jsdl:RangeValue_Type	DiskSpace	RangeValue
<FileSystemType>	jsdl:FileSystemTypeEnumeration	FileSystemType	Integer
<ExclusiveExecution>	xsd:boolean	ExclusiveExecution	String
<OperatingSystemName>	jsdl:OperatingSystemTy	OperatingSystemName	Integer

JSDL Element	JSDL element type	RSVP Object name	Data type
	peEnumeration		
<OperatingSystemVersion>	xsd:string	OperatingSystemVersion	String
<*OperatingSystem*> <Description> </*OperatingSystem*>	xsd:string	OperatingSystemDescription	String
<CPUArchitectureName>	jsdl:ProcessorArchitectureEnumeration	CPUArchitectureName	Integer
<IndividualCPUSpeed>	jsdl:RangeValue_Type	IndividualCPUSpeed	RangeValue
<IndividualCPUTime>	jsdl:RangeValue_Type	IndividualCPUTime	RangeValue
<IndividualCPUCount>	jsdl:RangeValue_Type	IndividualCPUCount	RangeValue
<IndividualNetworkBandwidth>	jsdl:RangeValue_Type	IndividualNetworkBandwidth	RangeValue
<IndividualPhysicalMemory>	jsdl:RangeValue_Type	IndividualPhysicalMemory	RangeValue
<IndividualVirtualMemory>	jsdl:RangeValue_Type	IndividualVirtualMemory	RangeValue
<IndividualDiskSpace>	jsdl:RangeValue_Type	IndividualDiskSpace	RangeValue
<TotalCPUTime>	jsdl:RangeValue_Type	TotalCPUTime	RangeValue
<TotalCPUCount>	jsdl:RangeValue_Type	TotalCPUCount	RangeValue
<TotalPhysicalMemory>	jsdl:RangeValue_Type	TotalPhysicalMemory	RangeValue
<TotalVirtualMemory>	jsdl:RangeValue_Type	TotalVirtualMemory	RangeValue
<TotalDiskSpace>	jsdl:RangeValue_Type	TotalDiskSpace	RangeValue
<TotalResourceCount>	jsdl:RangeValue_Type	TotalResourceCount	RangeValue
<DataStaging name=””> *(tag attribute)*	xsd:NCName	DataStagingName	String
<FileName>	xsd:string	FileName	String
<FileSystemName>	xsd:NCName	FileSystemName	String

JSDL Element	JSDL element type	RSVP Object name	Data type
<CreationFlag>	jsdl:CreationFlagEnumeration_Type	CreationFlag	Integer
<DeleteOnTermination>	xsd:boolean	DeleteOnTermination	Boolean
<Source> <URI> *</Source>*	xsd:anyURI	SourceURI	String
<Target> <URI> *</Target>*	xsd:anyURI	TargetURI	String

Table 4-8: Table of mandatory JSDL elements and RSVP objects mapping

4.12.1 Basic RSVP Protocol operation and G.OUNI signalling messages

The G.OUNI abstract messages are described in section 4.11. Most of these are directly supported by reusing existing procedures, messages, and objects defined under RSVP_TE [RFC3209] and GMPLS extensions for RSVP-TE [RFC3417, RFC34734] with some new objects described to support G^{2}MPLS later on this section. Complete and low level description these extensions are presented in [G2MPLS-EXT] as G^{2}RSVP-TE.

The two fundamental RSVP message types are Path and Resv are described in [RFC2205]. The RSVP Path message originated from a source node and is transmitted to a destination node and is used to establish a connection. A Resv message in response to Path message is sent by the destination node back to the source node. A connection is established when the imitating node receives a Resv message for the connection. PathTear and ResvTear messages are use to explicitly remove the Path and Resv states. The PathTear is sent from the source to the destination and removes both the path and reservation state for the associated connection whereas the ResvTear message is sent in the other direction and only removes the associated reservation state. Here we use modified versions of RSVP messages to map already standardised abstract message (NS messages) plus the GNS abstract messages required to support G^{2}MPLS.

The modified versions of RSVP messages include two new objects - GNS_CALL_EXT and GNS_UNI. The GNS_CALL_EXT object appears in all RSVP messages in which a CALL_ID is present (Path, Resv, PathTear and PathEr). Its presence makes a standard call a G^{2}MPLS call and piggybacks the GNS transaction construct.

The GNS_UNI object appears in the RSVP Path message like the Generalized UNI. No specific error code is identified for this object. Detailed description of the modified RSVP messages is given in [G2MPLS-EXT].

Message No.	Abstract Message Name	RSVP Message
G.OUNI standardised OIF messages		
1.	NS Create Request	Path
2.	NS Create Response	Resv, PathErr
3.	NS Create Confirmation	ResvConf
4.	NS Delete Request	Path or Resv with ADMIN_STATUS “Deletion in Progress” bit
5.	NS Delete Response	PathErr with Path_State_Removed flag, PathTear
6.	NS Status Enquiry	Implicit
7.	NS Status Response	Implicit
8.	NS Notification	PathErr, ResvErr
Grid Network Service (GNS) abstract messages required to support G^{2}MPLS NCP		
9.	Grid resource allocation request	Path
10.	GNS Create Request	Path
11.	GNS Create Response	Resv, PathErr
12.	GNS Create Confirmation	ResvConf
13.	GNS Delete Request	Path or Resv with ADMIN_STATUS “Deletion in Progress” bit
14.	GNS Delete Response	PathErr with Path_State_Removed flag, PathTear
15.	GNS Status Enquiry	Implicit
16.	GNS Status Response	Implicit
17.	GNS Notification	PathErr, ResvErr

Table 4-9: Mapping between G.OUNI Abstract Messages and RSVP Messages

4.12.2 G.OUNI RSVP-TE signalling procedures

The RSVP-TE protocol definitions in this section apply only for G.OUNI signalling. The G.OUNI RSVP-TE messages contain values as if they are used to setup two separate Network Service (NS) or Grid Network Service (GNS) creations, one between the initiating G.OUNI-C and G.OUNI-N, and the other between the G.OUNI-N and the terminating G.OUNI-C. The network is assumed to provide coordination of signalling information between the initiating and the terminating side of the NS or GNS creations. The network is required to support the transport of the information from the Generalized UNI object from the source G.OUNI-N to the destination G.OUNI-N plus all sub-objects of the Generalized UNI object that are described further down and on [G2MPLS-EXT].

4.12.2.1 *RSVP-TE signalling procedures for Phosphorus Overlay model*

To create a Network Service (connection), a G.OUNI node sends a Path message to its adjacent G.OUNI-N node. After transportation of Path message over the network G.OUNI-N on the other end of the network sends the Path message to the destination G.OUNI-C. Then a Resv message is sent back from that destination G.OUNI-C to the attached G.OUNI-N and in the same way the Resv is sent to the G.OUNI-N attached to the initiated G.OUNI-C. When the Resv message Source G.OUNI-C receives the Resv message then the NS (connection) within the transport network can be assumed to be established. At that time the source client can start sending data whereas the destination client can only send data when it receives the ResvConf message initiated by the source G.OUNI-C. Figure 4-13 shows the timing diagram and message flow during successful NS creation.

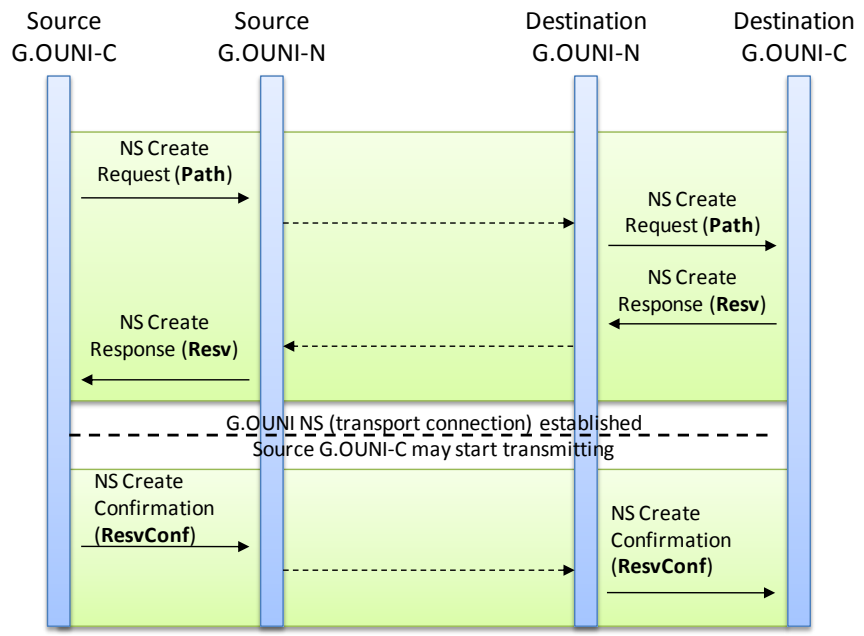

Figure 4-13: Successful Network Service Establishment

A NS may not be successfully created due to resource unavailability, policy or reachability constrains which is depicted in Figure 4-14. In this case the Path_State_Removed flag [OIF-UNI1.0R2-RSVP] is used.

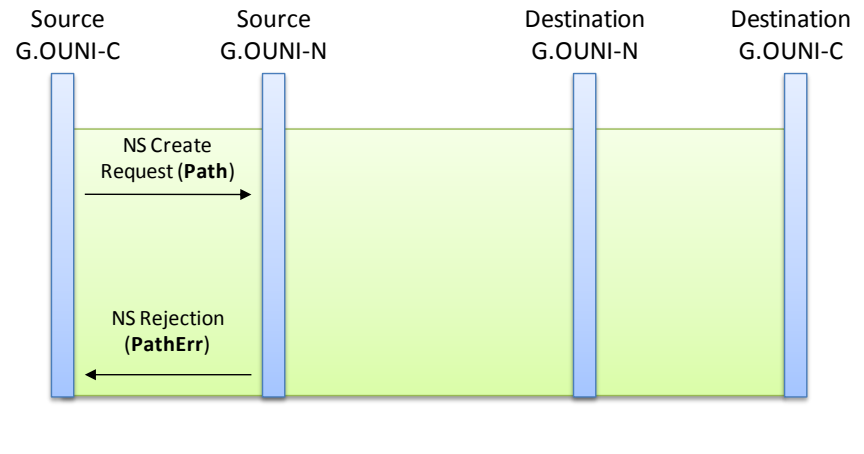

Figure 4-14: NS rejection by the Network using Path_State_Removed flag

If the Path_State_Removed flag is not set in the PathErr message then the following message flow that incorporates at PathTear should be supported.

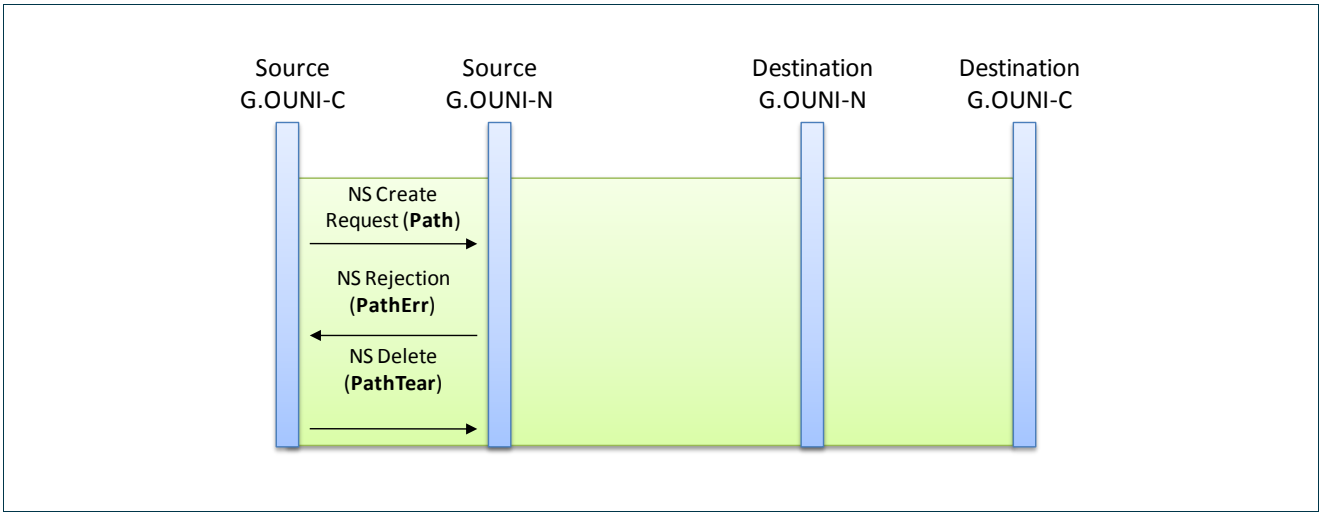

Figure 4-15: NS rejection by the Network without use of Path_State_Removed flag

In case of NS rejection from the destination G.OUNI-C the following message flow should be supported.

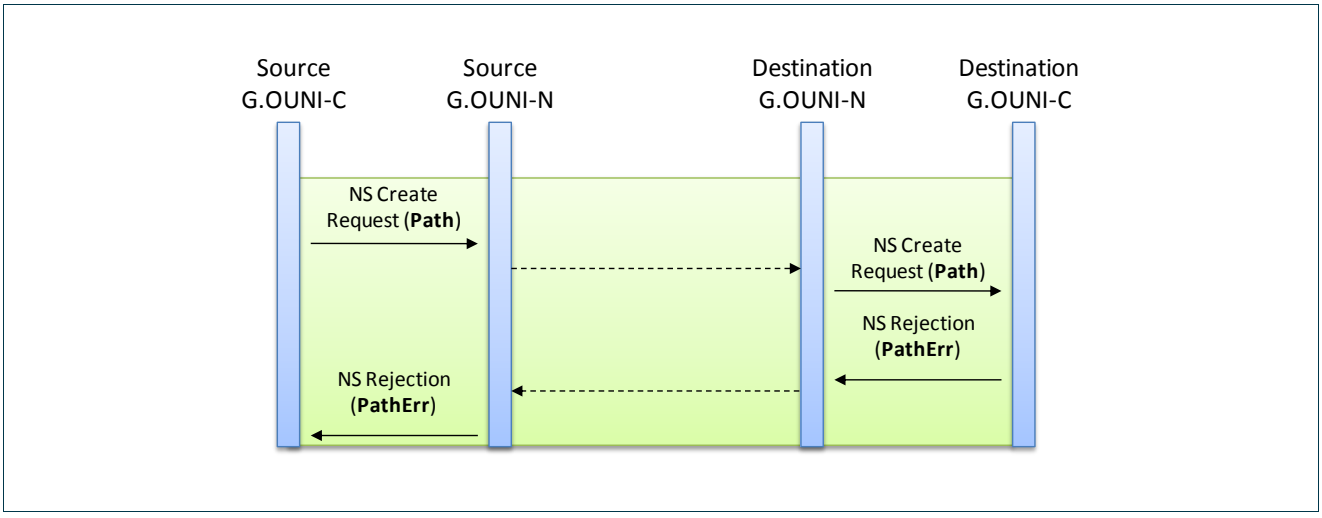

Figure 4-16: NS set-up rejection by the Destination G.OUNI-C

NS deletion procedure follows on the next figure which represents connection teardown initiated by the Source G.OUNI-C.

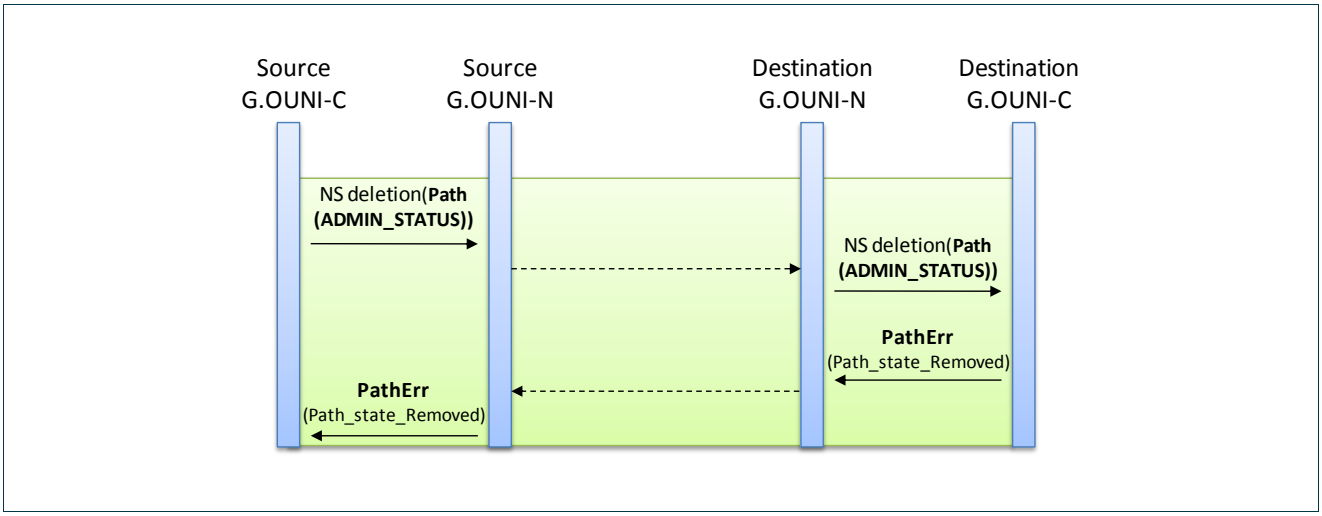

Figure 4-17: NS deletion initiated by the Source G.OUNI-C

A NS forced deletion process network error event is presented below.

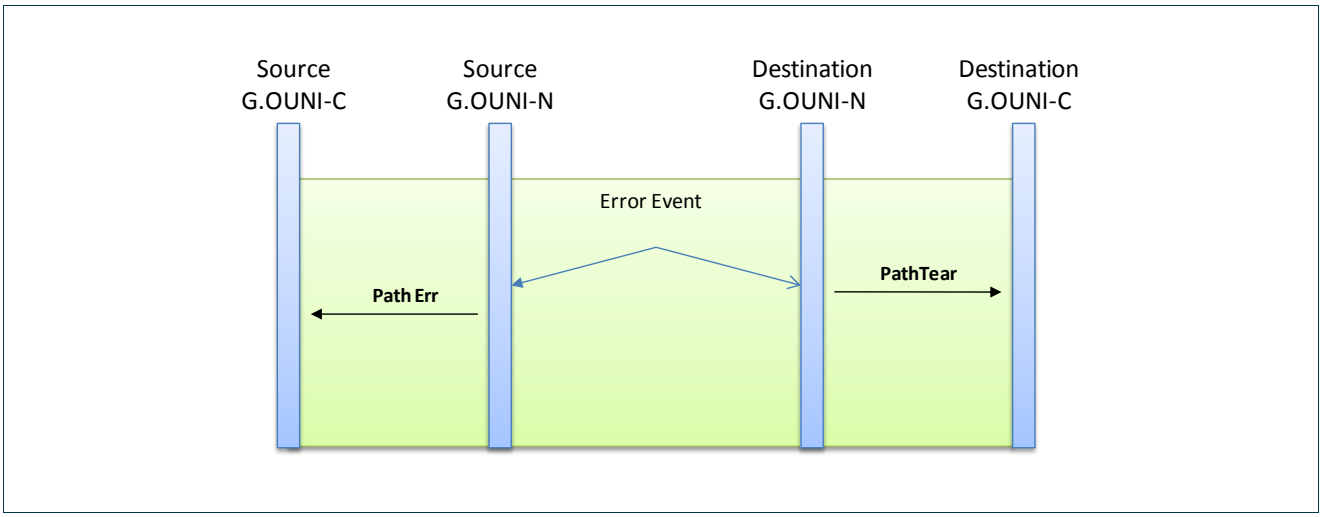

Figure 4-18: NS Forced deletion by the Network

4.12.2.2 *RSVP-TE signalling procedures for Phosphorus Integrated model*

The Grid Network Service (GNS) follows the same message flow format as the ones for the overlay model. The difference is that messages encapsulate GNS_CALL_EXT (on Path, Resv, PathTear, PathErr) and GNS_UNI

objects and its sub-objects listed on Table 4-8 (on Path) and extensively described in [G2MPLS-EXT] required to map JSDL information.

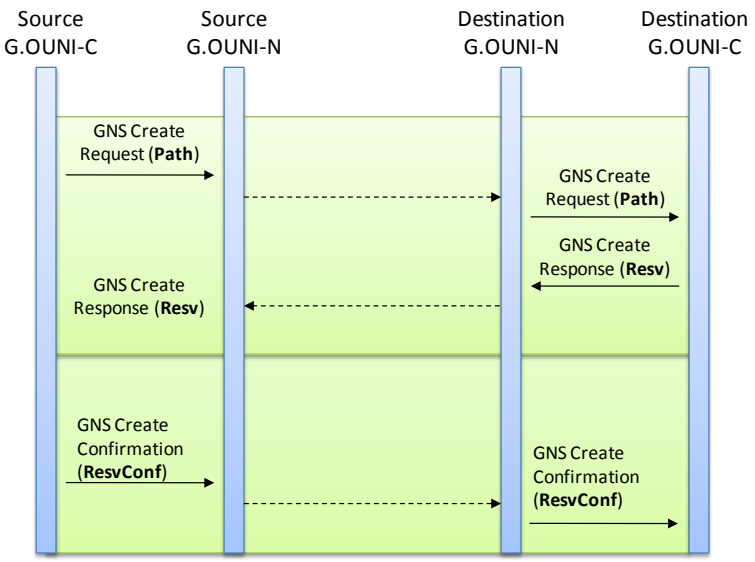

Figure 4-19: Successful Grid Network Service Establishment

The Grid resource allocation request is forwarded from G.OUNI-N to G.OUNI-C towards the remote Vsite in order to allocate Grid resources as an outcome of discovery and reservation of Grid and Network resources of G^{2}MPLS NCP under the integrated model.

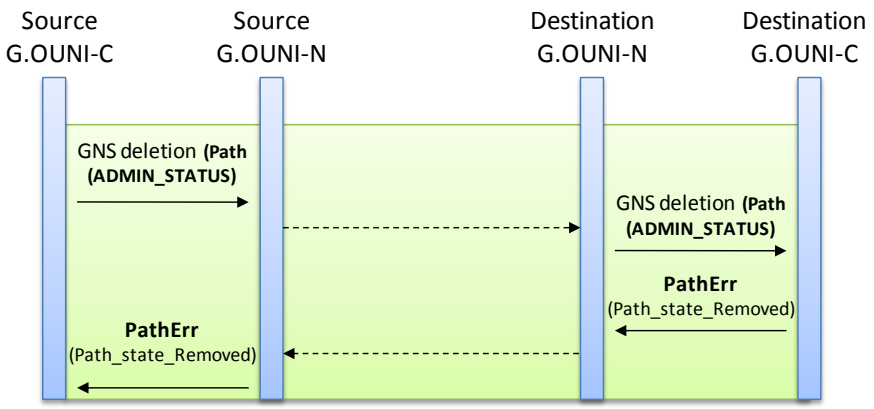

Figure 4-20: GNS deletion initiated by the Source G.OUNI-C

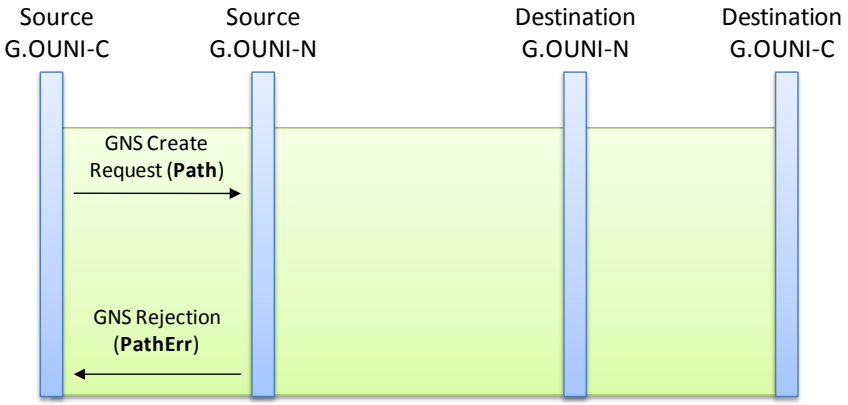

Figure 4-21: GNS rejection by the Network using Path_State_Removed flag

4.12.3 RSVP-TE message extensions

To extend the UNI RSVP-TE protocol with those Grid job specific parameters a new set of objects must be created and the TLV specification should be defined for each single parameter. All Grid and network objects used to define particular user requests are incorporated in the RSVP Messages. Specifically, two new RSVP

objects describing GNS attributes are added: GNS_CALL_EXT and GNS_UNI objects. Both objects follow the same standard RSVP structure defined in [IETF-RFC2205] for RSVP-TE objects

GNS_CALL_EXT is based on the standard CALL_ID object extending its attributes to specify job identification and time specification (advance reservation) for the job (Figure 4-22).

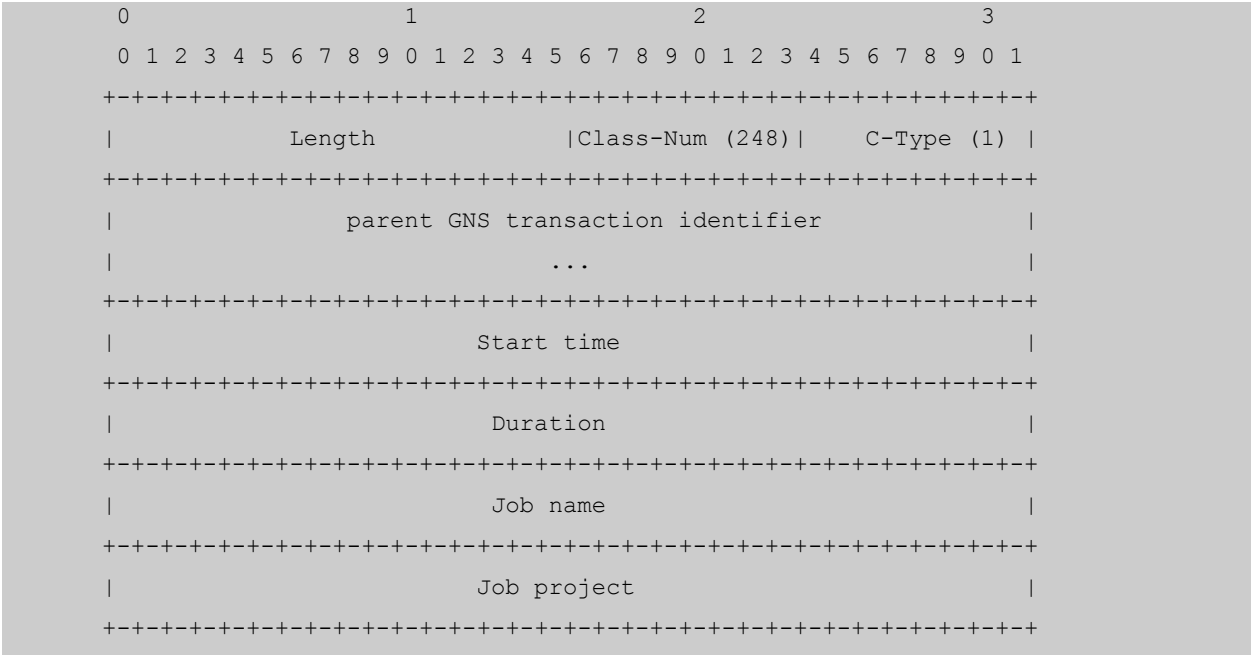

Figure 4-22: G.OUNI RSVP-TE GNS_CALL_EXT object format

GNS_UNI is based on the standard Generalized UNI object containing Grid specific parameters derived from the JSDL document and depicted in Table 4-8 (ref. Figure 4-23). This information is organized in GNS_UNI sub-objects (ref. Figure 4-23).

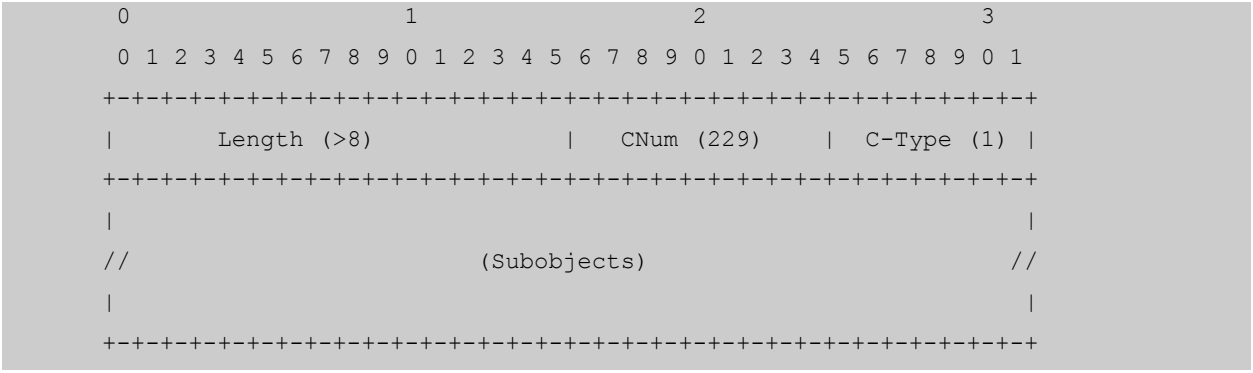

Figure 4-23: G.OUNI RSVP-TE GNS_UNI object format

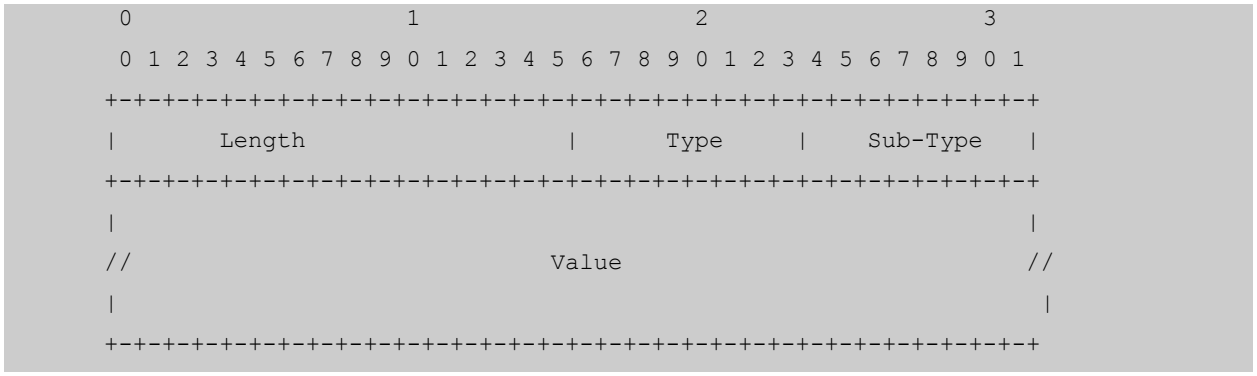

Figure 4-24: G.OUNI RSVP-TE GNS_UNI sub-object format

Description of GNS_CALL_EXT and GNS_UNI and its sub-objects can be found in [G2MPLS-EXT].

4.13 OSPF extensions

PHOSPHORUS has chosen the OSPF protocol for Grid site auto-discovery and grid resources capabilities advertising. It means that every G.OUNI-C must be able to exchange OSPF protocol messages. The advantage of the OSPF usage instead of the LMP is that, there is no need to translate all grid resource capabilities between OSPF protocol and LMP protocol. The OSPF protocol can work in the G.OUNI interface in nearly the same way as in G.ENNI interface. The G.OUNI-C is used by the grid middleware to exchange information with the G^{2}MPLS Control Plane.

OSPF routing extensions provided by PHOSPHORUS can be classified as:

- Grid resource description
- Grid resource availability calendar

Grid resource description characterizes Grid services and resources and Grid resource availability calendar provides scheduling information useful for advance reservations.

In GMPLS, each node gathers all network related information carried in OSPF TE-LSAs in the Traffic Engineering Database (TED) [IETF-RFC3630]. In G^{2}MPLS there is a need to add an additional database for the one of Grid extensions. This new database is Grid Resource Database (GRD). The rest of extensions don't need individual databases because this information will be gathered by the TED or GRD (Table 4-10).

G^{2}MPLS OSPF extension	Database

G²MPLS OSPF extension	Database
Grid Service Capability	**GRD**
Grid Resource Availability	**GRD**
Network Topology Information	**TED**
Optical Impairments	**TED**
Network End-point Assigned Addresses	**TED**
Network Resource Availability	**TED**

Table 4-10: OSPF extensions and OSPF databases relation

Each G²MPLS node possesses TED and GCD databases. The G.OUNI-N entity checks both databases and filters for any Grid middleware unnecessary information before sending them to the G.OUNI-C. The information received from the G.OUNI-C updates GCD and provokes refresh flooding in the domain. TED must not be updated because G.OUNI-C couldn't give any network information.

4.13.1 G.OUNI OSPF discovery procedures

The OSPF protocol in the G.OUNI interface can be used also for sharing knowledge on transport network resources like Transport Network Addresses (TNAs), network topology information and network TE-links availability schedules. However, it should be done only when it is necessary. G²MPLS Control Plane can limit the knowledge of the grid middleware on the network in case of the integrated deployment scenario. In the integrated model, if a grid middleware requests a job to be done without specifying resources to allocate, the grid middleware should not receive any network related information. If resources for jobs are always specified, the G.OUNI-N must send TNA addresses. In case of overlay model, a grid middleware must receive TNA addresses and, optionally, full or summarised topology information for ERO computation and network links availability schedules.

The message exchange depends on the architectural model. For the overlay model, service capability and availability information is advertised from one Grid site to others being piggybacked by the G²MPLS NCP (Figure 4-25). Thus, in this case, from the G.OUNI perspective, an OSPF update message containing the Grid Opaque LSAs may be initiated by G.OUNI-C or G.OUNI-N.

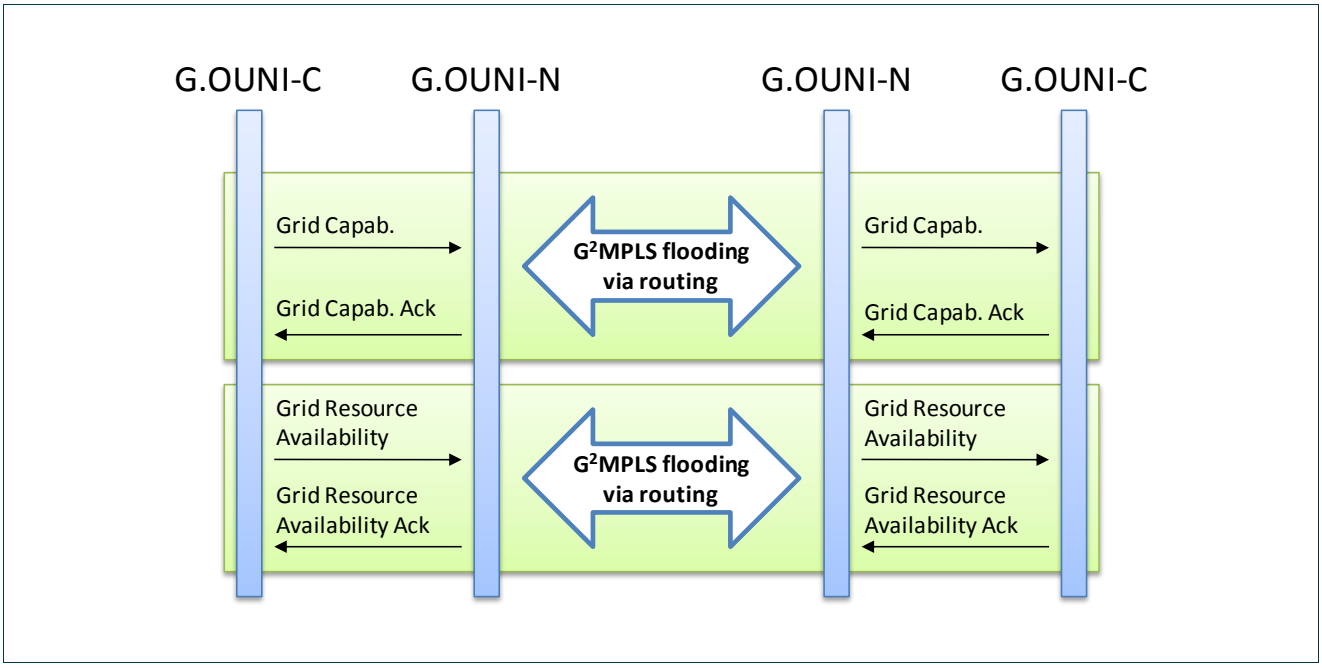

Figure 4-25: OSPF Grid Discovery message exchange in the overlay model

On the other hand, for the integrated model, service capability and availability information is just advertised from Grid sites to the G^{2}MPLS NCP (Figure 4-26). Thus, in this case, from the G.OUNI perspective, an OSPF update message containing the Grid Opaque LSAs can be initiated just by G.OUNI-C.

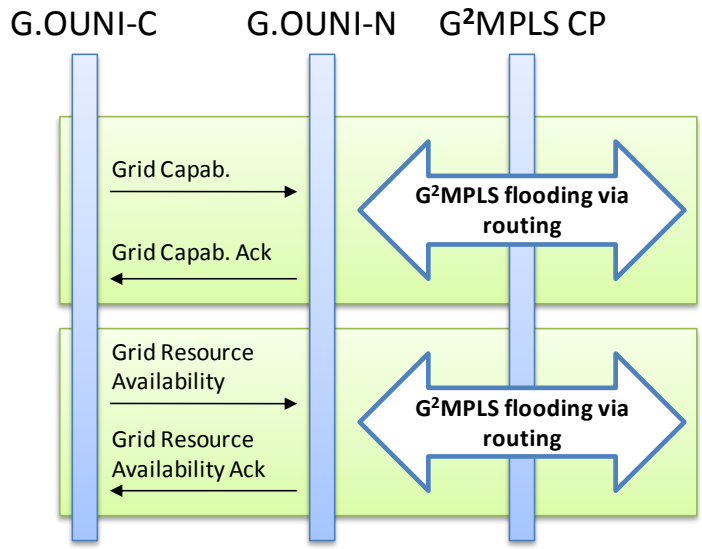

Figure 4-26: OSPF Grid Discovery message exchange in the integrated model

4.13.2 OSPF message extensions

Grid resource description and Grid resource availability extensions can be applied to G²MPLS routing protocol using new types of the OSPF Opaque LSA [IETF-RFC2370] presented on Figure 4-27. One OSPF message (LS Update) is able to carry one or more OSPF Opaque LSAs. If the OSPF message is long it is fragmented in IP layer.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            LS age            |     Options   |   9, 10 or 11 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Opaque Type  |               Opaque ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Advertising Router                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     LS Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         LS checksum           |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

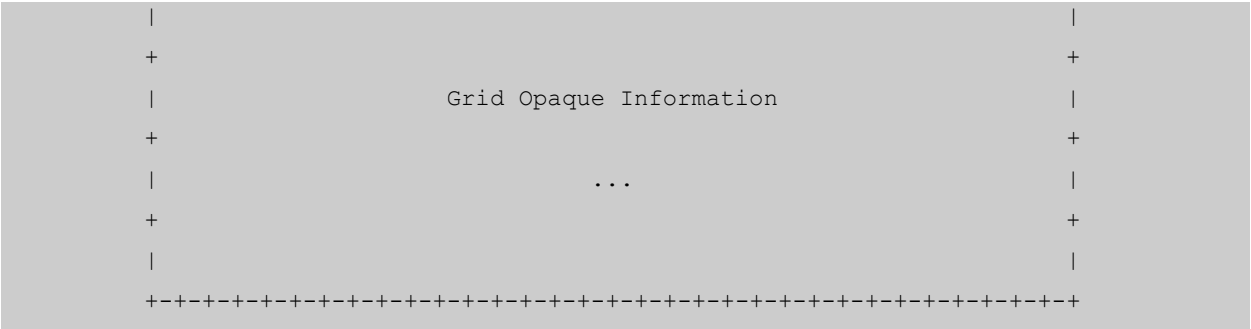

Figure 4-27: Grid OSPF Opaque LSA frame

The information carried by OSPF protocol is inserted in the Opaque Information field of Opaque LSA as a hierarchy of TLVs. TLV structure is presented in Figure 4-28.

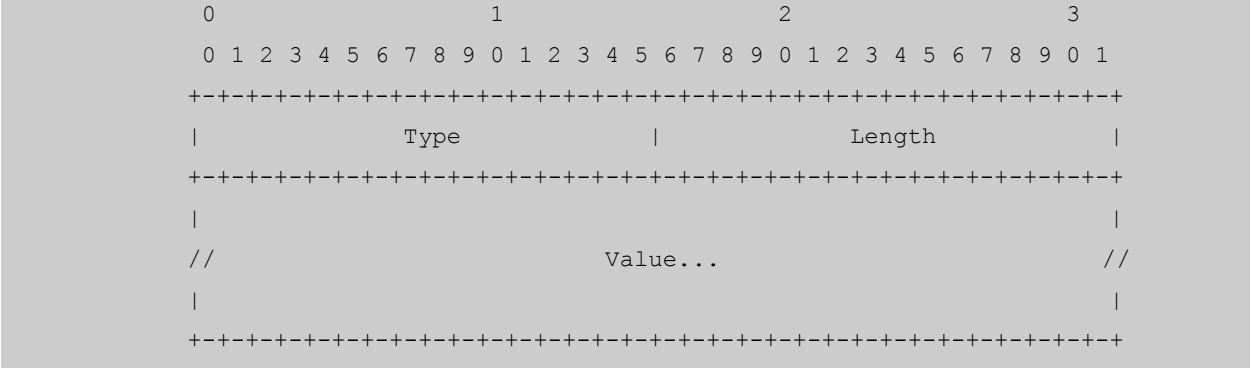

Figure 4-28: TLV frame for Grid extensions information (opaque type equal 248 or 249)

The detailed description of TLVs in Grid-LSA and Resource Calendar TLVs can be found in [G2MPLS-EXT].

4.14 LMP extensions

In the context of the PHOSPHORUS work the LMP messaging service discovery may be easily extended to support nodes and Grid site information exchange. The implementation of this feature will allow an integration of network and Grid resources and thus simplify the management of an infrastructure. Resource information messages will also significantly improve reservation and scheduling processes in a distributed computing environment. The choice of LMP to enable Grid Service/Resource capability and availability discovery instead of OSPF could be driven by the necessity of total compliance with OIF UNI in which service discovery is also based on LMP.

4.14.1 LMP discovery procedures

According to [OIF-UNI1.0R2-COMM] the service discovery procedure is performed always following the neighbour discovery procedure, so that the IPCC is/are already configured and ready to be used. The neighbour discovery is used by a client to determine the identities of the clients connected to the remote end of each data link and identify how these data links are interconnected. The discovery procedures are not mandatory; however, they must be replaced by manual procedures otherwise.

There are three types of messages used to perform service discovery with LMP which contain different information elements:

- *ServiceConfig*,
- *ServiceConfig Ack*,
- *ServiceConfig Nack*.

Service discovery messages are exchanged over an active IPCC based on LMP *Config* message exchange [IETF-RFC4204]. *ServiceConfig* should be always responded with *ServiceConfig Ack*/*Nack*. Both UNI-C and UNI-N can send *ServiceConfig* messages.

LMP messages are always prefixed with the common header (Figure 4-29):

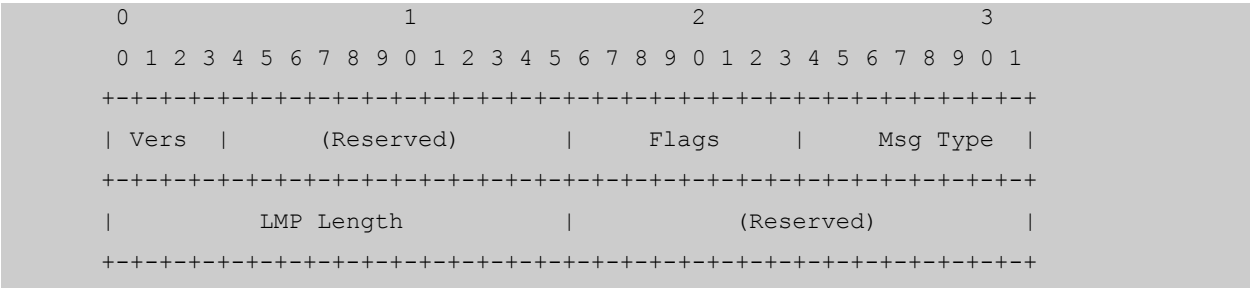

Figure 4-29: LMP common header

The extensions of LMP and service discovery for PHOSPHORUS project purposes are based on adding new types of objects that can be placed as *ServiceConfig* object content. The following table shows the *ServiceConfig* objects currently defined under UNI 1.0 (1-4) and the new objects proposed in PHOSPHORUS (5-6):

ServiceConfig C-Type	ServiceConfig Message Object	Message Direction
1	Signaling Protocol	G.OUNI-C → G.OUNI-N

2	Client Port-Level Service Attributes	G.OUNI-C → G.OUNI-N
3	Network Transparency & TCM Monitoring	G.OUNI-N → G.OUNI-C
4	Network Diversity	G.OUNI-N → G.OUNI-C
5	Grid Services Capability	G.OUNI-C → G.OUNI-N G.OUNI-N → G.OUNI-C
6	Grid Resources Availability	G.OUNI-C → G.OUNI-N G.OUNI-N → G.OUNI-C

Table 4-11: ServiceConfig Messages

The message exchange supposing an automatic service discovery depends on the architectural model. For the overlay model, service capability and availability information is advertised from one Grid site to others being piggybacked by the G^{2}MPLS NCP (Figure 4-30). Thus, in this case, from the G.OUNI perspective, a *ServiceConfig* message may be initiated by G.OUNI-C or G.OUNI-N.

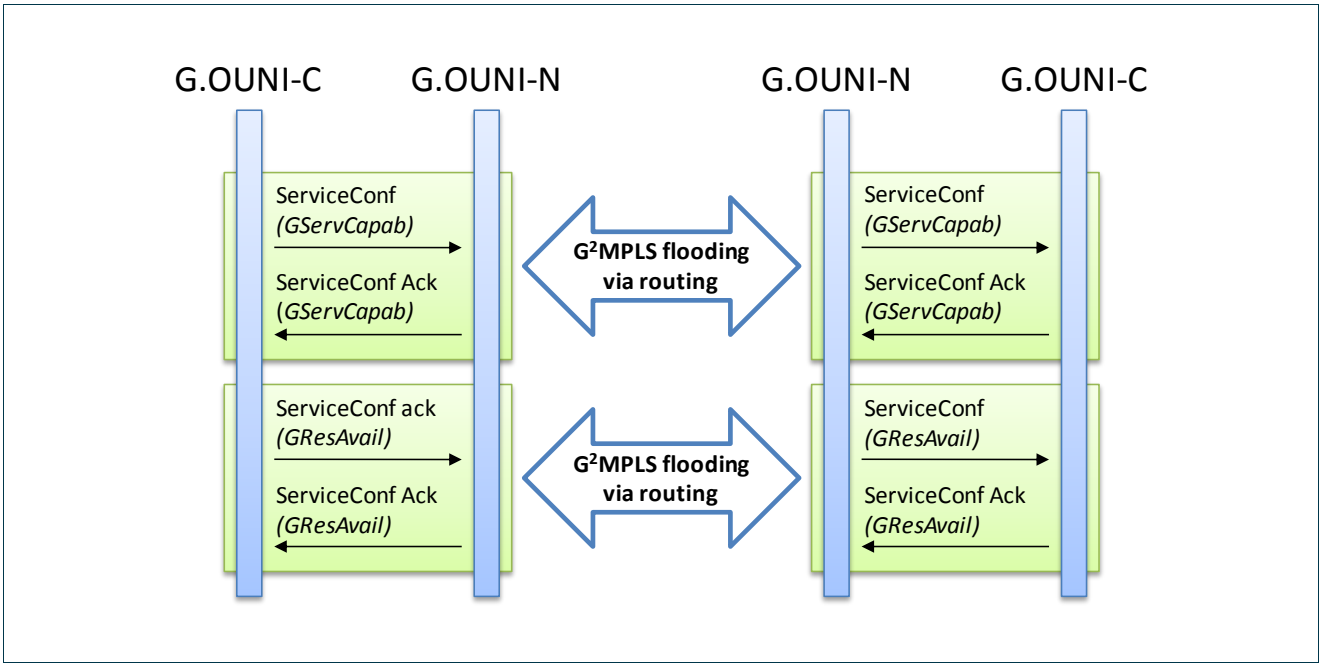

Figure 4-30: LMP Grid Discovery message exchange in the overlay model

On the other hand, for the integrated model, service capability and availability information is just advertised from Grid sites to the G^{2}MPLS NCP (Figure 4-31). Thus, in this case, from the G.OUNI perspective, a *ServiceConfig* message can be initiated just by G.OUNI-C.

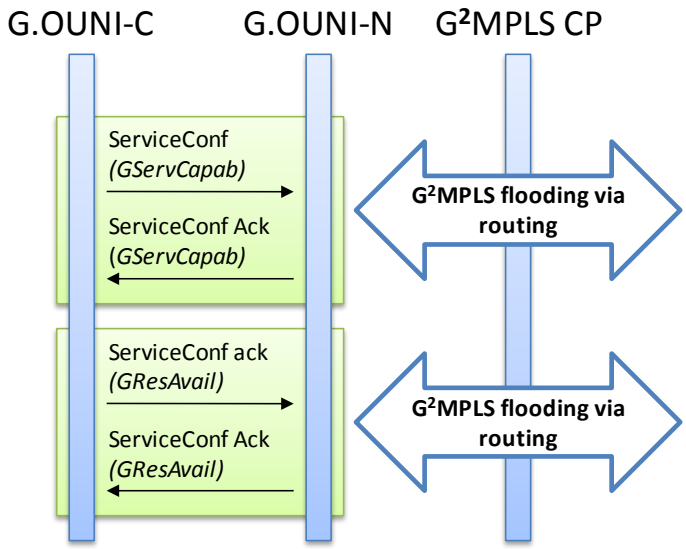

Figure 4-31: LMP Grid Discovery message exchange in the integrated model

4.14.2 LMP message extensions

LMP extensions for service discovery describe Grid service capabilities and Grid resource availabilities. The content of the new messages is enclosed in *ServiceConfig* objects, which are defined for all possible attributes. As seen in Table 4-11, there are four such objects defined in [OIF-UNI1.0R2-COMM], all associated with the network and the TNE. New identifiers (C-Type) have been assigned to the two new *ServiceConfig* messages avoiding interference with OIF standards and specifications.

The format of the *ServiceConfig* object is shown in (Figure 4-32):

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|    C-Type    |     Class     |        Length                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                (ServiceConfig Object Content)                //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4-32: ServiceConfig Object format

Where:

- N – defines whether the attribute is negotiable (N=1) or non-negotiable (N=0)
- C-Type – defines class type within the Object Class
- Class (8 bits) – defines the type of the Object – for ConfigService Object the value is 51
- Length – defines the length of the Object in bytes
- *ServiceConfig* Object Content – defines the supported services and attributes at the G.OUNI

4.14.2.1 *Grid Services Capability Object*

The *Grid Services Capability* object (Class = 51, C-Type = 5) indicates the capabilities of the Grid services such as types of services or applications (e.g. meta-scheduling, indexing, data mining, data storage, visualization, etc.) and the capabilities of the Grid resources (types of CPU, storage, OS, etc.) attached to the G.OUNI. The process of Grid services capability advertising is initialized by a client. Then, the object is sent from the local G.OUNI-C to the local G.OUNI-N (overlay/integrated) and from the remote G.OUNI-N to the remote G.OUNI-C (overlay).

The *Grid Services Capability* Object takes the following format:

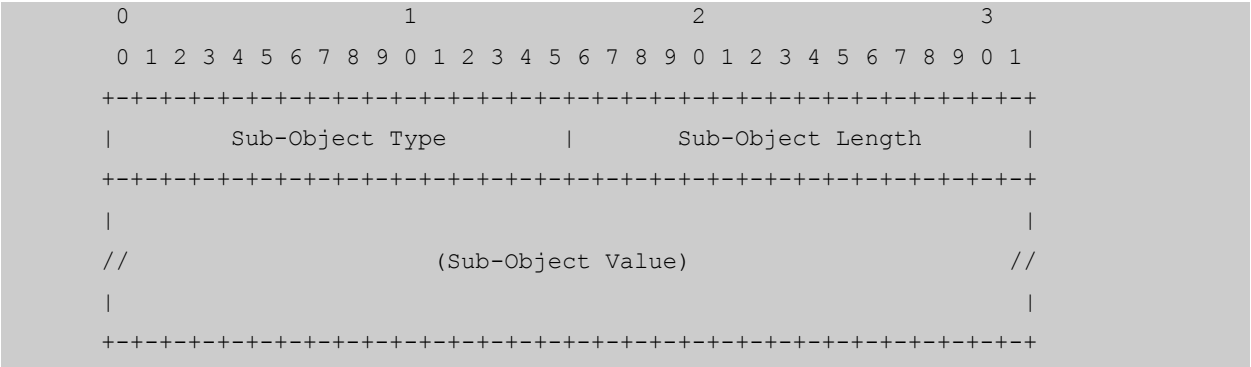

Figure 4-33: Grid Services Capability Object format

Where:

- Sub-Object Type – defines the type number of the sub-object
- Sub-Object Length – defines the length of the sub-object in bytes
- Sub-Object Value – defines the contents of the corresponding sub-object

Grid Services Capability sub-objects characterize the attributes and parameters identified in Table 4-7 and described in the GLUE schema [GLUE].

4.14.2.2 *Grid Resources Availability Object*

The *Grid Resources Availability* object (Class = 51, C-Type = 7) indicates the amount of Grid resources currently available. The process of Grid resources availability advertising is initialized by the client (i.e. G.OUNI-C to G.OUNI-N) and it takes the same format as *Grid Services Capability* Object.

Grid Resources Availability sub-objects characterize some of the attributes and parameters identified in Table 4-7, under *Computing Element*, *SubCluster*, *Software*, *Host*, *Storage Element* and *Storage Area Property* sub-tables, and described in the GLUE schema [GLUE].

5 G.E-NNI interface and functionalities

As Automatically Switched Optical Networks (ASONs) are deployed into new and existing networks, it cannot be assumed that such networks will be homogeneous (e.g. with respect to transport technologies, vendors, approach to control/management). In order to support deployment of a G^{2}MPLS control plane into heterogeneous Grid Network environments, it is essential to introduce and support the concept of control domains, and in particular, the specification of the signalling and routing information exchanged between such domains as defined in [OIF-E-NNI-Sig-1.0, OIF-E-NNI-Rtr-1.0]. However, extensions to standardised protocols and procedures are defined and described in order to support GNS services among different domains. The G.E-NNI reference point is defined to exist between G^{2}MPLS control domains. The nature of the information exchanged between control domains across the E-NNI reference point represents the common semantics of the information exchanged amongst it components, while allowing for different representation inside each domain. G.ENNI is instantiated by signalling and routing protocols and as such it becomes a G.E-NNI signal and routing interface.

5.1 Services supported over the G.E-NNI

Services offered over G.E-NNI are divided into two main categories: the network services and the Grid services. Network services are based on the ones already standardized in OIF E-NNI documents [OIF-E-NNI-Sig-1.0, OIF-E-NNI-Rtr-1.0] plus some extensions, which are required for PHOSPHORUS and are described below. Network services basically refer to on-demand and in-advance connections, given specific parameters such as bandwidth, ingress and egress access points. On the other hand, Grid Network Services include also Grid job service request, reservation and co-allocation as well as access to Grid resources such as CPU or storage. An overview of the Network services and Grid Network services supported are provided in sections 5.1.1 and 0 accordingly.

5.1.1 Network services offered over G.E-NNI

A control plane consists inherently of different functional entities, one of which is concerned with the Reference Point called the External Network-Network Interface (E-NNI). The E-NNI standards and definitions from OIF and ITU-T can be used as a basic platform for the G.E-NNI.

G.E-NNI interface specifications are driven by ITU standards [G.8080] and [G.807], which define three basic connection types according to the distribution of connection management functionality between control and management planes. These connections are the permanent connection (PC) provisioned by the management system, a switched connection (SC) provisioned by a signalling control plane after a user request, and finally a soft permanent connection (SPC) which represents an end-to-end connection from which the user-to-network portion is a PC established by the management system and the network portion is a SC using the control plane. E-NNI is only required for the inter-domain SCs and the SPCs. Both for SCs and SPCs the call parameters originating from the source user must be preserved and passed across the E-NNI(s). E-NNI must be able to transact explicit route information associated with the E-NNI gateways. Finally, it must support for call and connection separation as described in [G.8080]. The functions of the Network call controllers (NCCs) provided at gateways between control domains (i.e. separated by the E-NNI reference point) are defined according to policies associated with the interactions between control domains. In adition, signalling services that are going to be realised by the RSVP-TE, G.E-NNI scope are to utilize routing information exchange to support theASON routing architecture. The base protocol that will be used is the OSPF with extensions for Traffic Engineering [RFC3630] and GMPLS [RFC4203].

5.1.2 Grid services offered over G.E-NNI

In the same way as network services, Grid services are offered to clients over G.E-NNI. Grid services such as Grid job/service requests translated in GNS transactions between Grid service layer and G^{2}MPLS over G.OUNI have to be carried out over G.E-NNI in order to allow on-demand access to multi-domain Grid resources. This way G^{2}MPLS can perform GNS services such as reservation, co-allocation, actual use and release of Grid and network resources. Moreover, these procedures are also required in order to facilitate Grid resource management and negotiation between Grid client/application and G^{2}MPLS control plane.

Based on the G.OUNI services, G.ENNI should be able to propagate messages required to provide services such as:

1. Grid Service Discovery
2. Grid Resource Discovery

3. Grid Advance Reservation Request
4. Grid Advance Reservation Cancellation
5. Grid Resource co-allocation

5.2 G.E-NNI signalling

G^{2}MPLS E-NNI signalling is conceived to cope with:

- Grid job/service requests translated in setup of GNS transactions
- support of advance reservations
- support of implicit network destinations for Calls related to a Grid service (destination is an amount of CPU or storage wherever it is)

The G.E-NNI signalling interface exists between two signalling control domains and supports the procedures of [ASON-DCM] and [ASON-RSVP], extended with the GNS transaction related information specified in [GMPLS-EXT]. G.E-NNI signalling must be compatible with call/connection control originating from or destined to a G.OUNI complaint interface. Any unknown protocol objects shall be dealt with according to the methods of specific protocols.

5.2.1 G.E-NNI signalling Reference configurations

The G.E-NNI signalling reference configuration is illustrated in Figure 5-1 and is based on the E-NNI configuration derived in [OIF-E-NNI-Sig-1.0]. The G.E-NNI exists between two signalling control domains. The upstream protocol controller transmits a call request while the downstream protocol controller serves as receiver for the request. The upstream protocol control is referred to as G.ENNI-U; the downstream protocol control is referred to as G.ENNI-D. The protocol controller (either G.ENNI-U or G.ENNI-D) could include both network call controller (NCC) and connection controller (CC) functionality as defined in [G.8080].

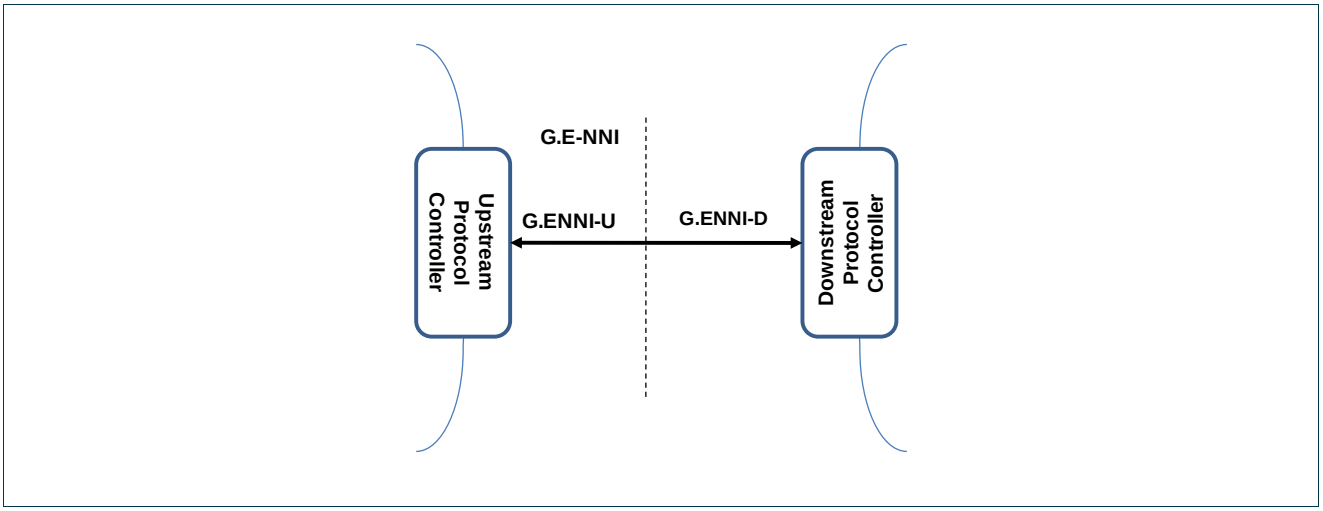

Figure 5-1: G.E-NNI reference configuration.

5.2.2 Main architectural entities

In full compliance with the ASON model, the upstream protocol controller transmits an NS or GNS request while the downstream protocol controller serves as receiver for the request. The upstream protocol controller is commonly referred to as G.ENNI-U; the downstream protocol controller is referred to as G.ENNI-D. A protocol controller may refer to either G.ENNI-U or G.ENNI-D.

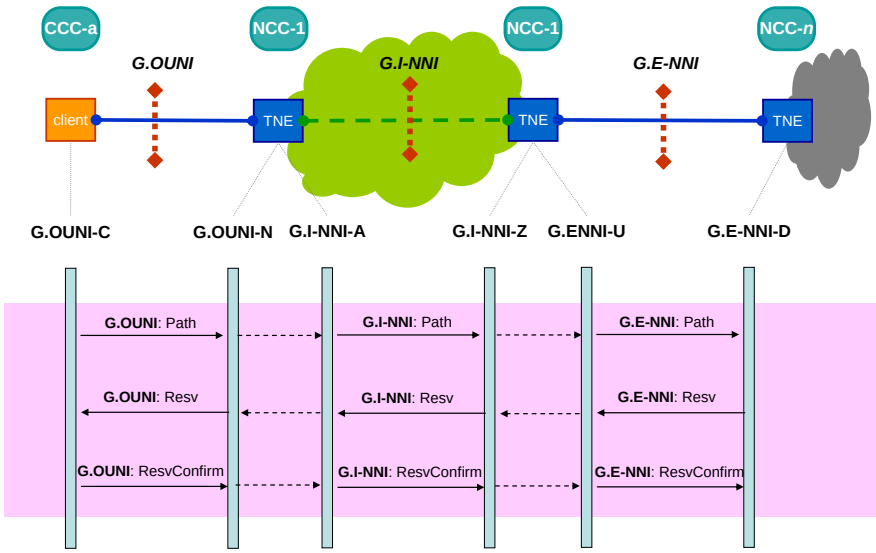

Figure 5-2: G^{2}RSVP-TE message mapping across the network reference points.

Identifiers for the Transport Resource Names, the Signalling Protocol Controller Identifiers and the Signalling Protocol Controller SCN Addresses are selected in the same addressing spaces specified in [OIF-E-NNI-Sig-01.0].

5.2.3 G.E-NNI signalling abstract messages

The G.E-NNI interface has knowledge of the GNS semantic and implements its functionalities through the following abstract messaging classes. These messages are called “abstract” because they do not represent any specific protocol messages and can be used as a semantic reference for specific implementations in signalling protocols (e.g. RSVP, CR-LDP, PNNI).

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated	Message Direction
G.E-NNI standardised OIF messages				
101.	NS Create Request	S	O/I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
102.	NS Create Response	S	O/I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated	Message Direction
103.	NS Create Confirmation	S	O/I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
104.	NS Delete Request	S	O/I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
105.	NS Delete Response	S	O/I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D
106.	NS Status Enquiry	S	O/I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
107.	NS Status Response	S	O/I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D
108.	NS Notification	S	O/I	G.E-NNI-D ➔ G.E-NNI-U
Grid Network Service (GNS) abstract messages required to support G^{2}MPLS NCP				
109.	GNS Create Request	S	I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
110.	GNS Create Response	S	I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D
111.	GNS Create Confirmation	S	I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
112.	GNS Delete Request	S	I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
113.	GNS Delete Response	S	I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D
114.	GNS Status Enquiry	S	I	G.E-NNI-U ➔ G.E-NNI-D G.E-NNI-D ➔ G.E-NNI-U
115.	GNS Status Response	S	I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D
116.	GNS Notification	S	I	G.E-NNI-D ➔ G.E-NNI-U G.E-NNI-U ➔ G.E-NNI-D

Table 5-1: G.E-NNI Messages

A list of G.E-NNI attributes is associated with each G.E-NNI message; some of them are mandatory for a given signalling message and some are optional.

5.2.4 RSVP-TE Extensions for G.E-NNI signaling

The selected signalling protocol for G^{2}MPLS operations across the G.E-NNI interface is RSVP, with its standard extensions contributed by IETF for the GMPLS part and by OIF for the E-NNI parts [OIF-E-NNI-Sig-01.0].

The signalling extensions and the message formats identified in [GMPLS-EXT] are also applicable to G.E-NNI signalling. Therefore, the base E-NNI RSVP protocol is extended with the GNS_CALL_EXT object (C-num = 248, C-Type = 1) and the GNS_UNI object (C-num = 249, C-Type = 1) as described in [GMPLS-EXT].

No specific additional extension is needed for the implementation and maintenance of the E-NNI call segments.

5.2.4.1 *Mapping of Abstract Messages to RSVP-TE Messages*

Table 5-2 provides a mapping of the abstract message to specific RSVP-TE messages used to support signaling across the E-NNI interface.

Message No.	Abstract Message	RSVP Message
101.	NS Create Request	Path
102.	NS Create Response	Resv, PathErr
103.	NS Create Confirmation	ResvConfirm
104.	NS Delete Request	Path or Resv with ADMIN_STATUS object and D&R bits set
105.	NS Delete Response	PathTear or PathErr with Path_State_Removed flag)
106.	NS Status Enquiry	Implicit
107.	NS Status Response	Implicit
108.	NS Notification	Notify or PathErr
109.	GNS Create Request	Path
110.	GNS Create Response	Resv, PathErr

Message No.	Abstract Message	RSVP Message
111.	GNS Create Confirmation	ResvConfirm
112.	GNS Delete Request	Path or Resv with ADMIN_STATUS object and D&R bits set
113.	GNS Delete Response	PathTear or PathErr with Path_State_Removed flag)
114.	GNS Status Enquiry	Implicit
115.	GNS Status Response	Implicit
116.	GNS Notification	Notify or PathErr

Table 5-2: G.E-NNI Abstract Messages mapped to RSVP-TE Messages.

5.2.5 G.E-NNI signalling flow for different scenarios

This section provides the signal flow of the G.E-NNI interface for normal setup release requests as well as exception and defect handling.

The deriving signalling flows for the different messaging classes are shown in Figure 5-3 and Figure 5-4, by skipping the ack-ing mechanism in NS and GNS requests for the sake of clarity. The network service (NS) setup request is initiated by the Grid middleware and through G.OUNI and G.I-NNI is propagated across the G.E-NNI signalling interface. This process continues until the request is received by the destination control domain. The request is completed after multiple message exchanges to setup the connection. The G.E-NNI signalling interface assumes that there will be three-message handshaking involved (with the third message being optional) in setting up the G.E-NNI portion or the network connection. Both Figure 5-3 and Figure 5-4 illustrate the signal flow across G.E-NNI portion of the NS and GNS requests by also showing the Grid MW, G.OUNI and G.I-NNI flows for the sake of completeness.

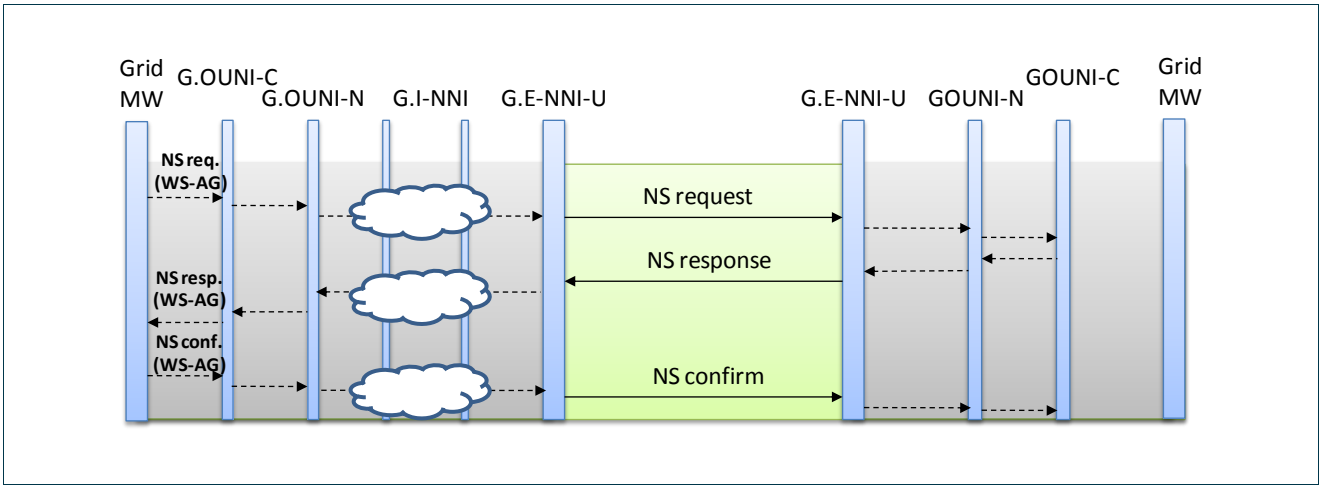

Figure 5-3: Message flow for NS request at G.E-NNI.

The same process is applied for the GNS request. In this case the request includes both network and grid resources.

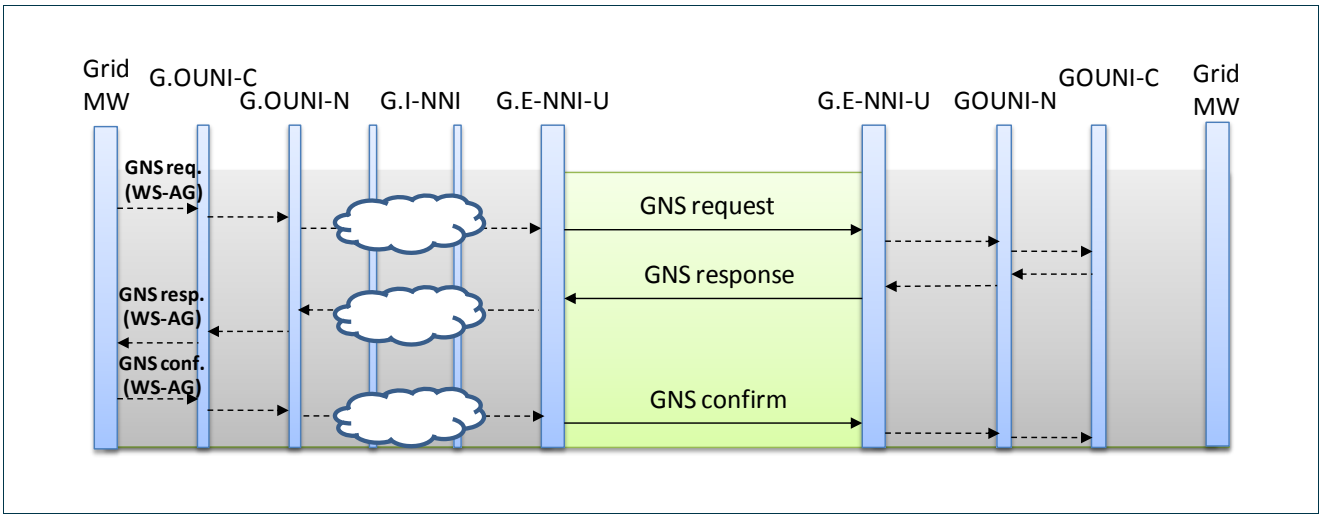

Figure 5-4: Message flow for GNS request at G.E-NNI.

In case of normal NS release request, several signal flows may be followed. This depends on who initiated the release request. Control plane operation is used in this case to support the release of the end-to-end connection. The G.E-NNI signalling interface assumes that there will be two-message handshaking involved in releasing the G.E-NNI portion of the NS connection as illustrated in Figure 5-5.

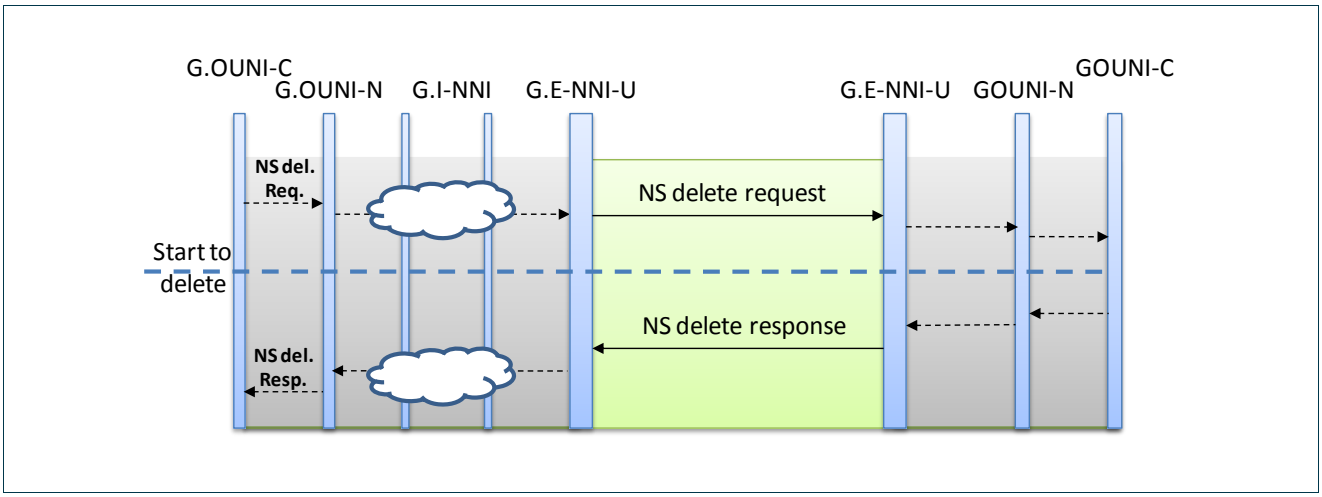

Figure 5-5 NS release initiated by source G.OUNI.

The same process applies for the GNS service release initiated by the source G.OUNI as shown in Figure 5-6.

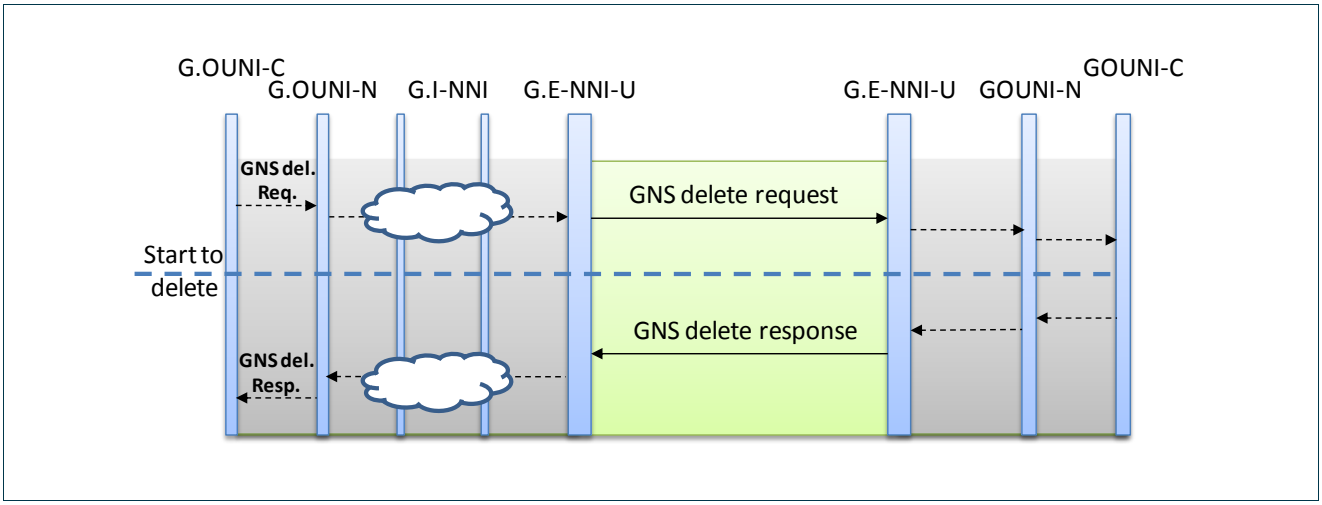

Figure 5-6 GNS release initiated by source G.OUNI.

In case of NS and GNS release initiation from the destination G.OUNI side then the signalling flows follows the process projected in Figure 5-7 and Figure 5-8 accordingly.

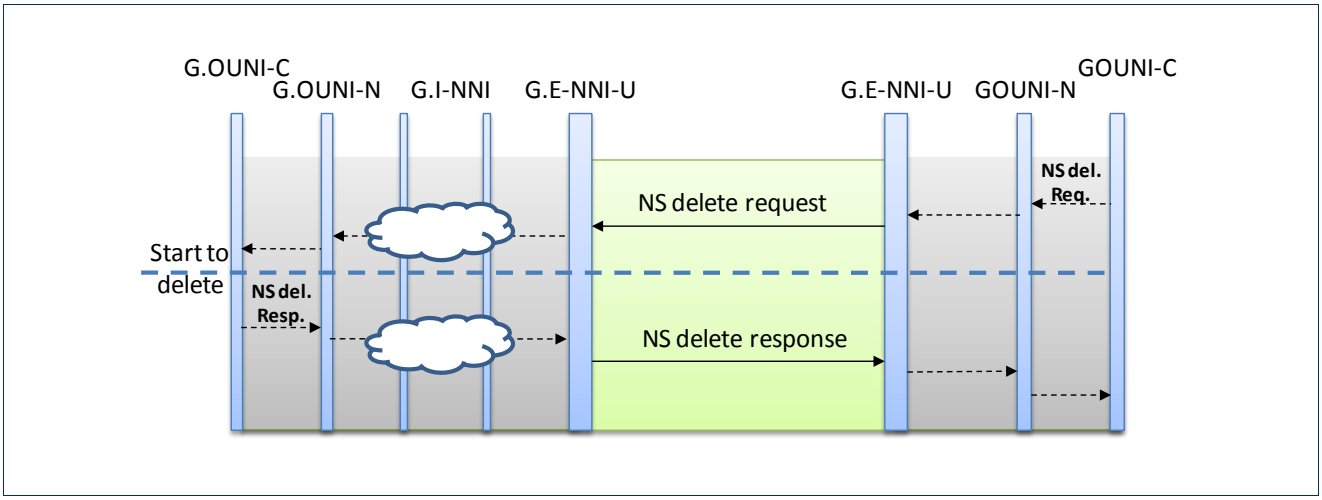

Figure 5-7 NS release initiated by destination G.OUNI.

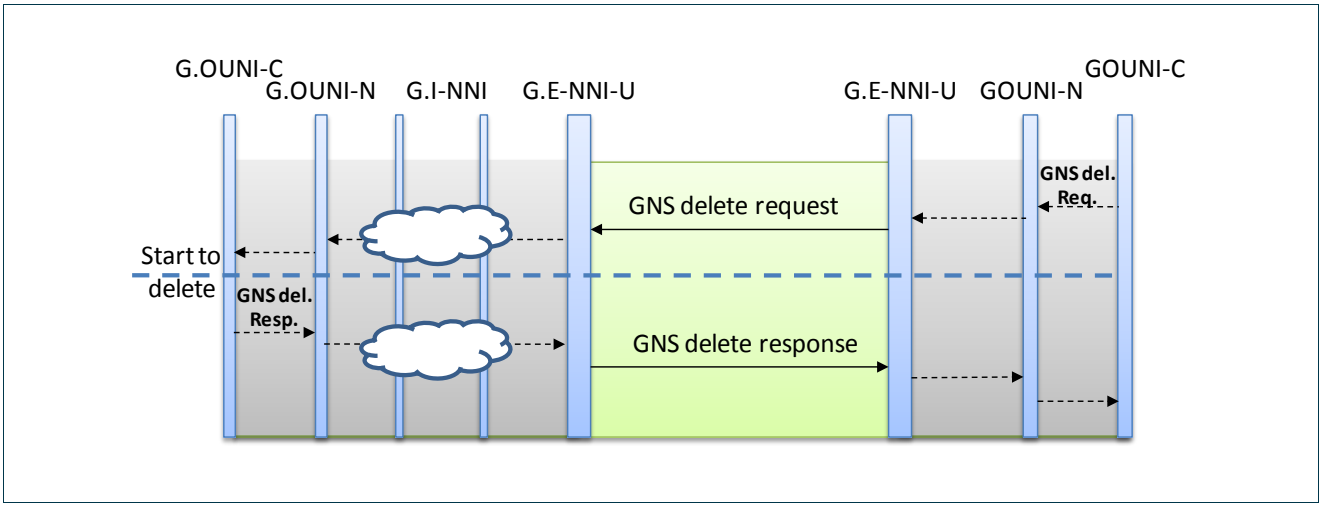

Figure 5-8 GNS release initiated by destination G.OUNI.

Another scenario of releasing NS and GNS exists where intermediate control plane nodes (connection controllers) may initiate a NS or GNS delete. This may occur due to failure on the network path in case of NS or due to failure of the Grid job process and/or network path for GNS case. The flows corresponding to these scenarios are shown below (Figure 5-9 and Figure 5-10).

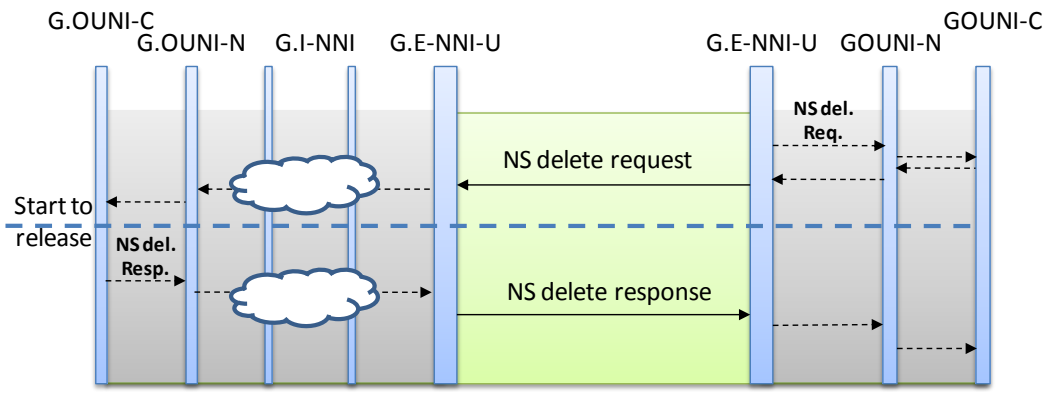

Figure 5-9 NS release initiated by intermediate node.

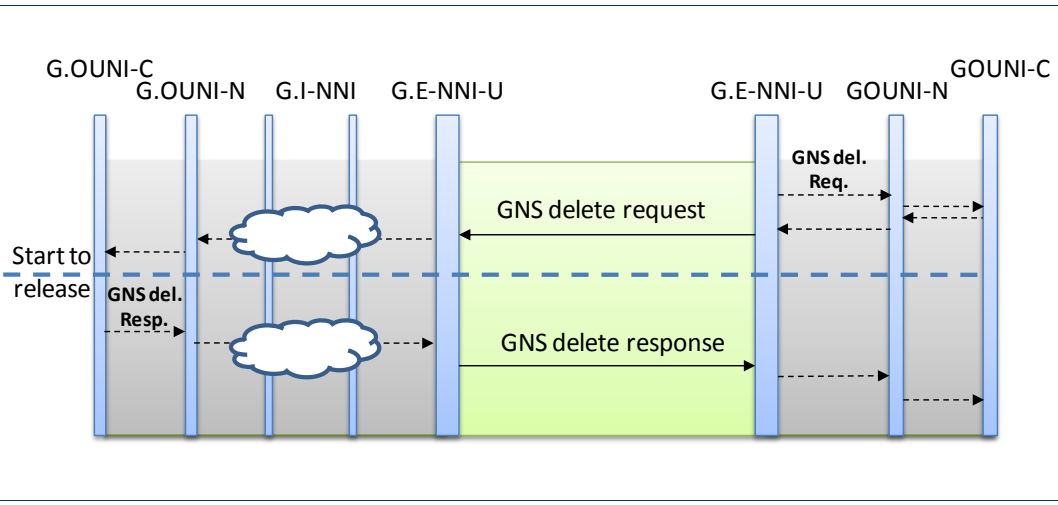

Figure 5-10 GNS release initiated by intermediate node.

Following the successful operations handled over G.E-NNI, operations on exception and defect handling can be reported. The causes of any type of rejection ranging from resource conflicts, to policies to time-outs are beyond the scope for this document and thus are not provided. The signal flows as a result of a rejection are described below. A rejected request can exist on different phase of the signal flow and as a result certain signals flows must be followed. The rejection can occur either during the first phase (NS or GNS request messages) or the second phase (response message). In the first case, any nodes along the request message path may cause rejection of the message. Thus the G^{2}MPLS can decide to release any pending resources by

sending a release message towards the source node (NCC) both for NS and GNS as shown in Figure 5-11 and Figure 5-12 respectively.

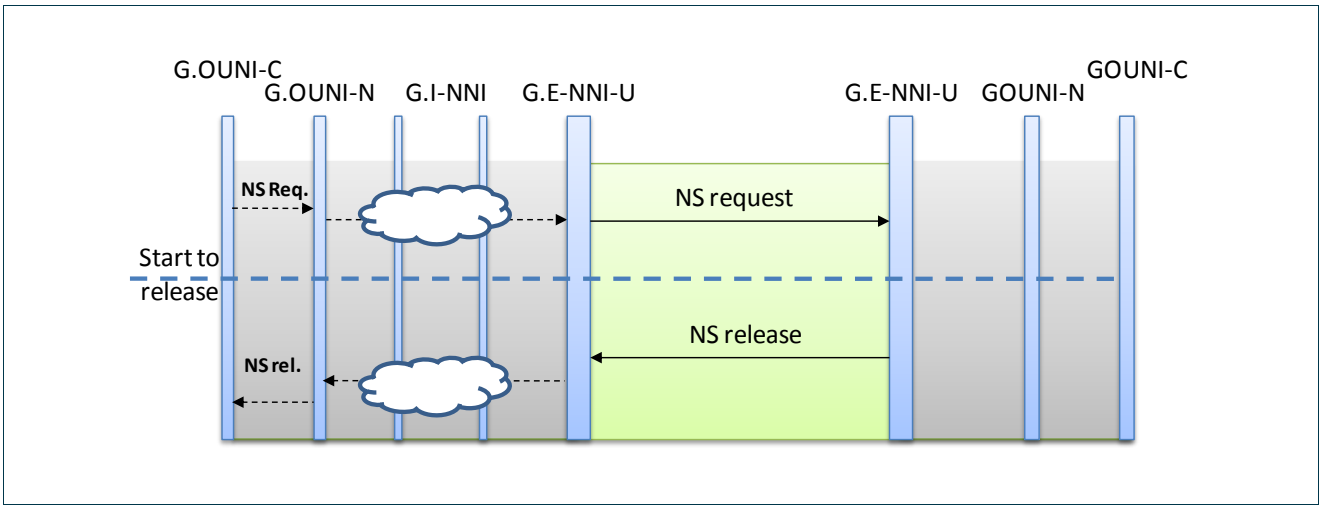

Figure 5-11 NS setup rejection during first phase (request).

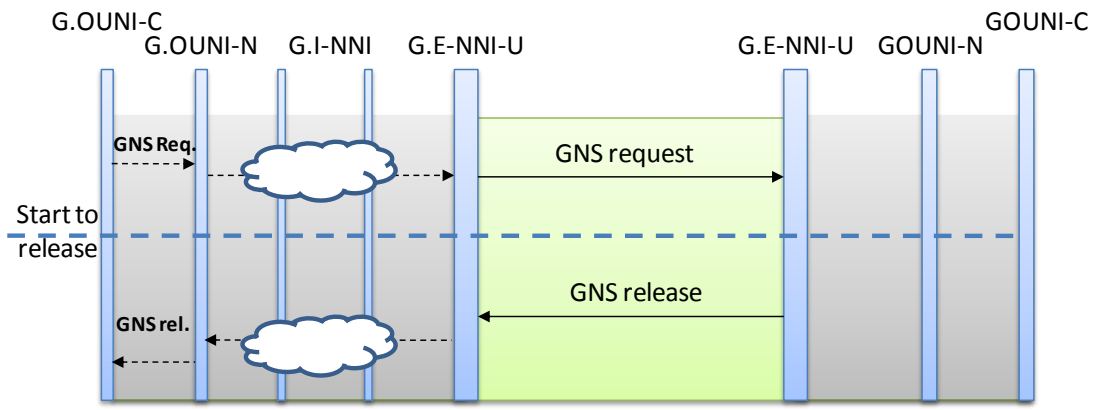

Figure 5-12 GNS setup rejection during first phase (request).

In case of rejection failure during response phase, then the following signal flows (Figure 5-13 and Figure 5-14) must be applied.

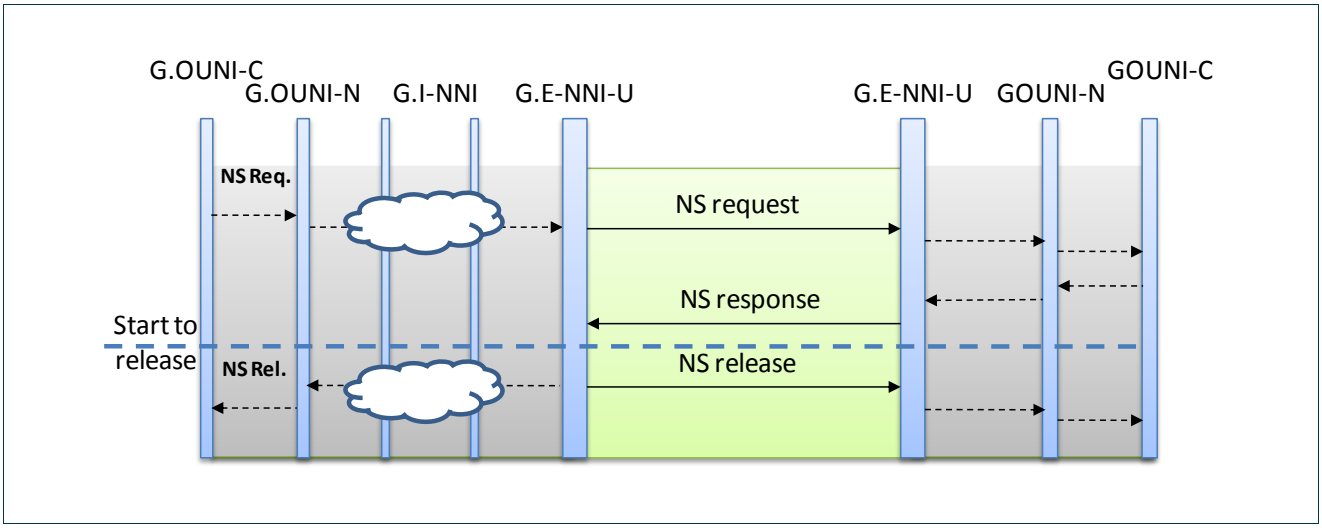

Figure 5-13 NS setup rejection during second phase (response).

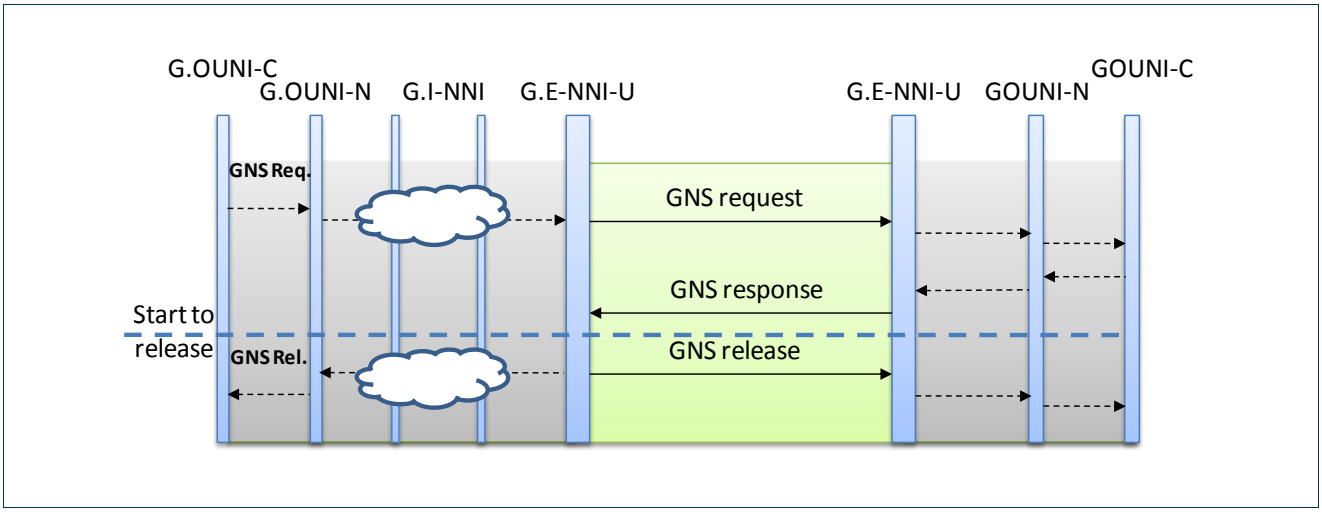

Figure 5-14 GNS setup rejection second phase (response).

Finally, another case is the rejection of a release request that may only happen when the request is not valid. For any other scenarios, the release request must be carried out. In case of unsuccessful release of resources, then error messages may be sent to the external controller (e.g. management system), however the response to release request should be successful for stopping the billing process and releasing the resources.

5.3 G.E-NNI routing

The G^{2}MPLS E-NNI routing disseminates information about control domains between Routing Control Domains (RCD) (Figure 5-15). It supports ASON routing architecture [G.7715], GMPLS routing [IETF-RFC: 3630, 3471, 4203, 4328], OIF routing [OIF-ENNI-OSPF] and Phosphorus G^{2}MPLS requirements [G2MPLS-ARCH].

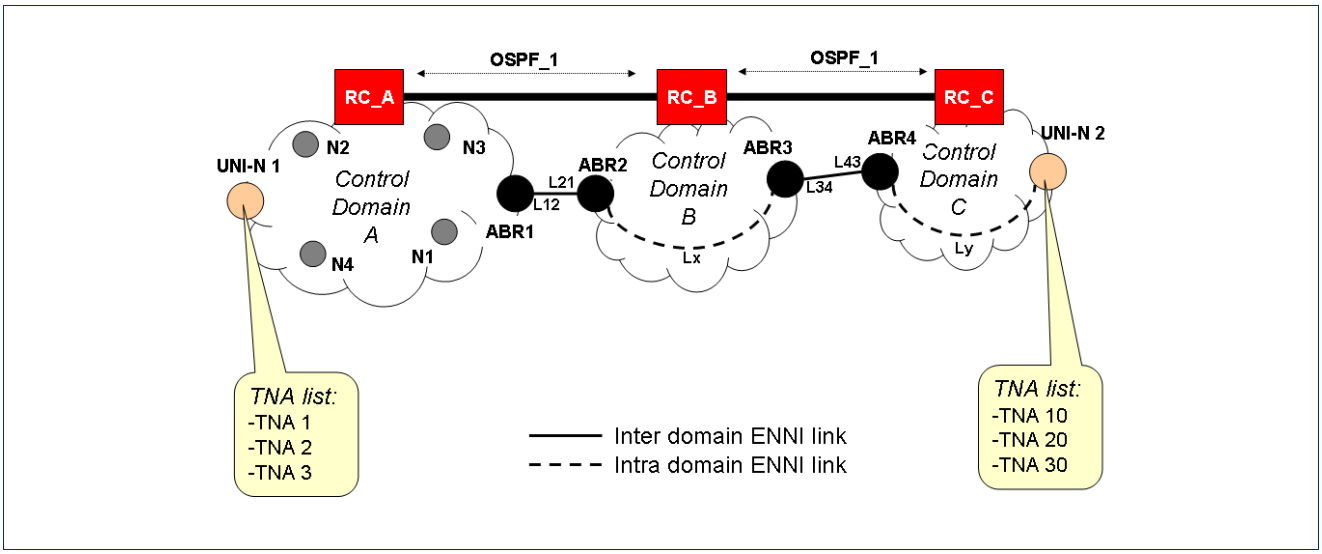

Figure 5-15: OIF E-NNI routing components.

The main purpose of the G^{2}MPLS ENNI routing is the inter-domain information flooding about local and learned Grid resources and network resources.

5.3.1 G.E-NNI basic components

The main entities taking part in the inter-domain routing exchange are inter-domain Routing Controllers (G.eNNI-RC) as shown in Figure 5-16. The G.eNNI-RC represents a Routing Control Domain (RCD). Every RCD has one and only one G.eNNI-RC responsible for external view of the domain. The G.eNNI-RC establishes routing adjacencies with every G.eNNI-RC placed in the neighbouring RCDs. Routing adjacencies are used to exchange G.E-NNI routing messages. The G.eNNI-RC gathers resources information from the local domain and floods the information to the adjacent G.eNNI-RCs. Additionally, the G.eNNI-RC is learning the information from the adjacent domain and it popularizes the information on the other adjacencies and also in the local domain.

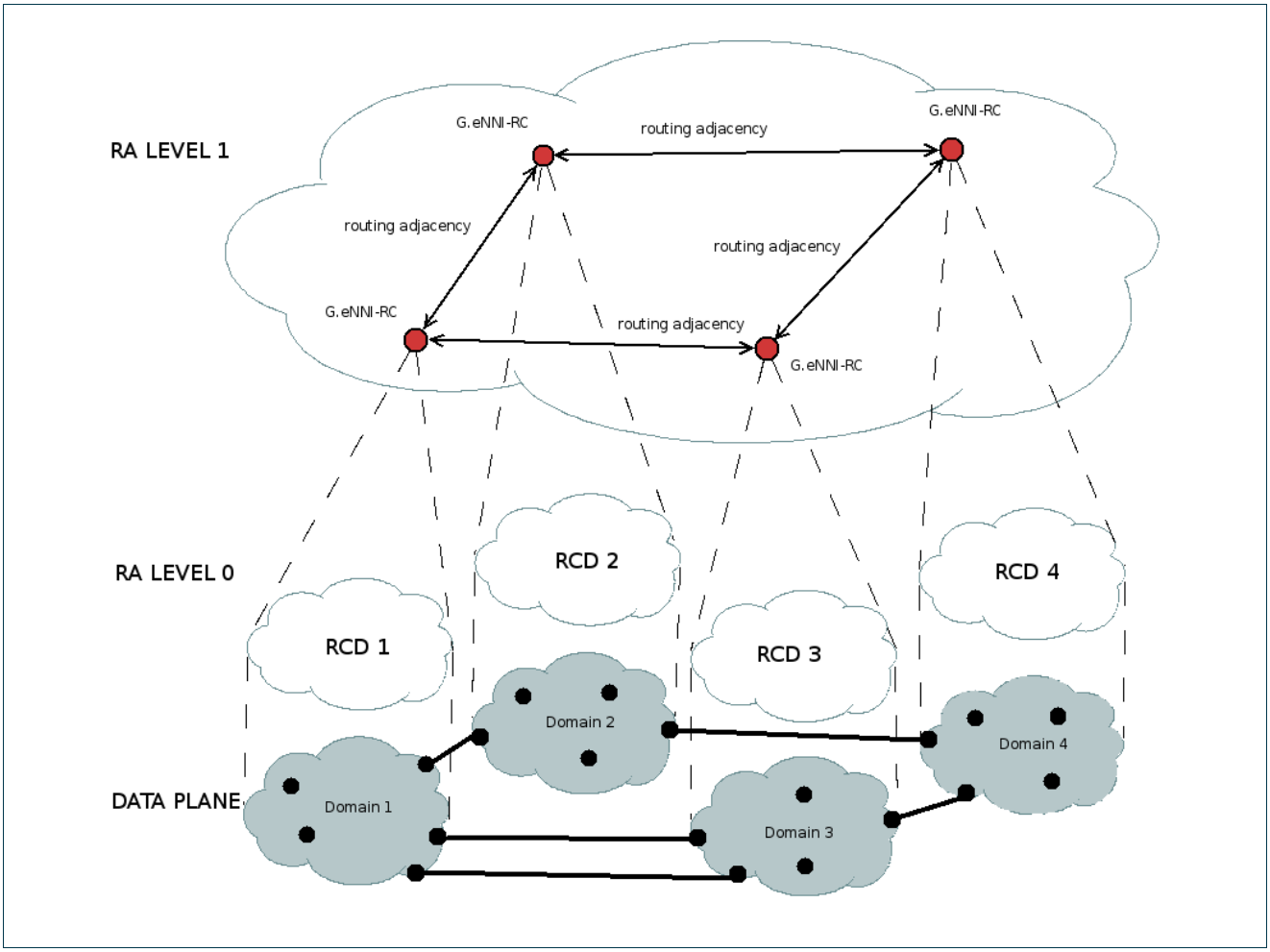

Figure 5-16: G.E-NNI routing.

To establish an inter-domain adjacency the G.eNNI-RC needs:

- neighbour G.eNNI-RC ID,
- SCN address of the neighbouring G.eNNI-RC.

5.3.2 Support of routing hierarchy

The G^{2}MPLS routing is based on double level hierarchy shown in Figure 5-17. The G.E-NNI routing is placed in RA level 1 of the hierarchy, whereas the G.I-NNI routing is placed in the RA level 0. The routing controller in the RA level 1 is called G.eNNI-RC. The routing controller in the RA level 0 is called G.iNNI-RC. The G.eNNI-RC

and exactly one elected G.iNNI-RC create a point of junction of these two RA levels, where information feed-down and feed-up occurs.

Information which should be feed down from RA level 1 to level 0 and flooded by intra-domain adjacencies:

a) grid resources of others domains,

b) network end-points (TNAs) of others domains,

c) network topology of others domains.

Information which should be feed up from RA level 0 to level 1 and flooded by inter-domain adjacencies:

a) grid resources of the local domain,

b) network end-points (TNAs) of the local domain,

c) [optionally] inter-domain TE-links connected to the domain border nodes,

d) [optionally] network topology abstraction of the local domain.

Grid information from the RCD should be summarized before flooding in RA level 1, which might result in scalability of the routing databases and traffic.

If the Area Border Routers (ABR) flood the G.E-NNI TE-links (inter-domain TE-links) in RA level 0, then the G.eNNI-RC can learn about G.E-NNI TE-links from RA level 0. G.E-NNI TE-links can be also statically configured in the G.eNNI-RC. In this case, there is no need to learn G.E-NNI TE-links from RA level 0, instead, the G.eNNI-RC must have configured, for every G.E-NNI TE-link, the following information:

- ABR Node ID of the remote end of G.E-NNI TE-link,
- G.eNNI-RC ID responsible for the neighbouring RCD of the remote end of G.E-NNI TE-link.

In default, any information about the local domain topology isn't flooded in the RA level 1 by the G.eNNI-RC. This behaviour can be changed and the G.eNNI-RC could generate the network topology abstraction of the local domain and flood summarized intra-domain TE-links, i. e. virtual connections abstractions between ABRs belong to the same base domain.

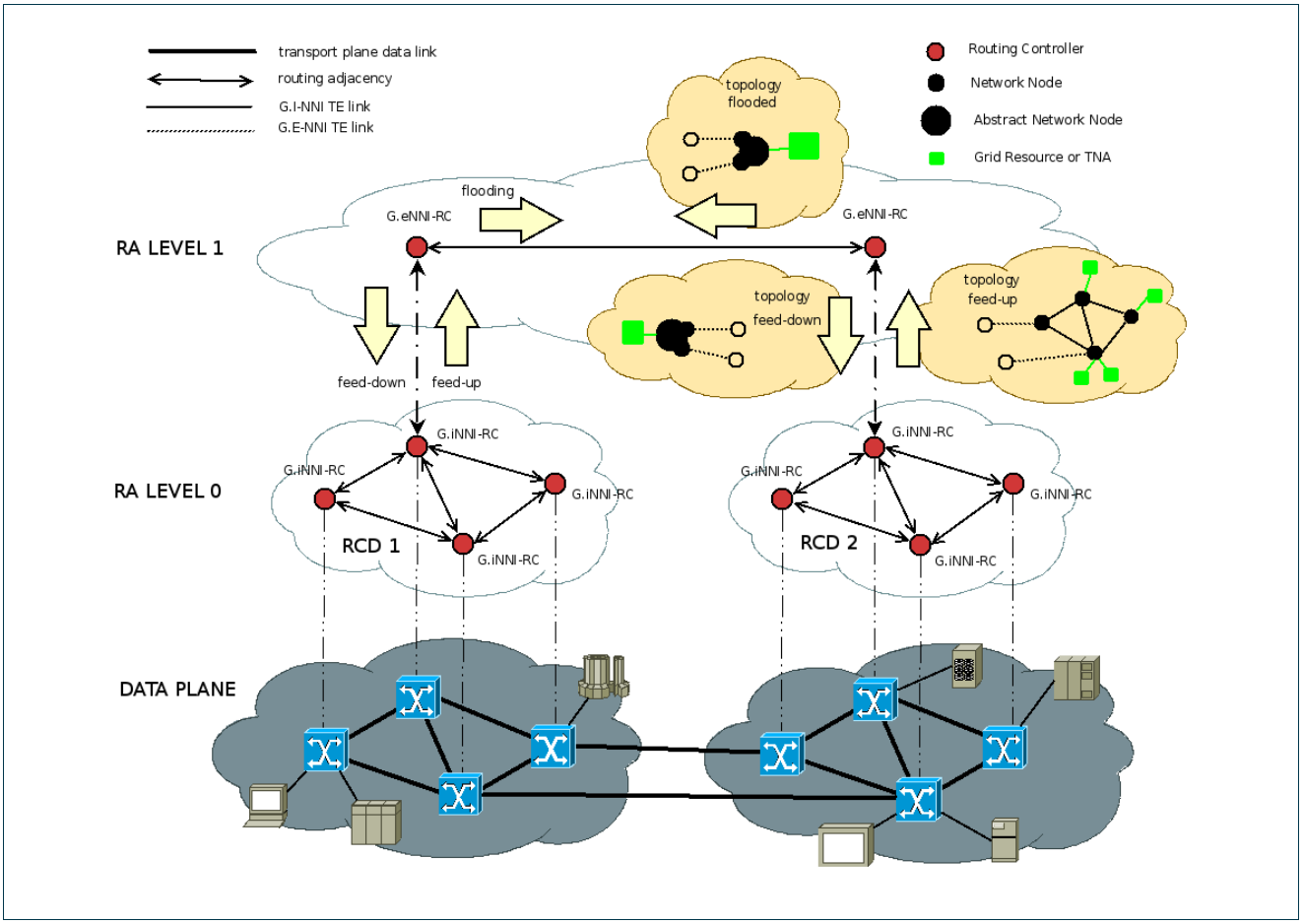

Figure 5-17: G^{2}MPLS routing hierarchy.

The G.eNNI-RC and the G.iNNI-RC are preventing from information closed-loop circulation by applying the rules:

- don’t feed-down incoming information with advertising router equal local RC ID,
- don’t feed-down information already has being locally feed-up,
- don’t feed-up incoming information with advertising router equal local RC ID,
- don’t feed-up information already has being locally feed-down.

5.3.3 Hierarchy and topology abstraction

Hierarchical routing can be used to enable the network to scale, and to provide isolation between different network domains. Topology abstraction can be used to reduce the amount of information carried by the inter-domain routing protocol. When hierarchy is created and topology abstraction is used, the externally advertised topology can be a transformed view of the actual internal topology of a contained Routing Area. This transformed view is intended specifically to provide information for computation of paths crossing the Routing Area, represented by advertisements of links and associated costs. This can impact the routing performance, depending on the conditions within the Routing Area and the use of tools that provide additional routing information, e.g., a Path Computation Element as discussed below. Advertisement of an abstracted topology of a multi-node domain MUST support a valid representation of connectivity within that domain in order to support correct path computation. This will avoid failure of path computation across the domain. In general, path computation should not have to infer from the control identifiers in use (such as the RC identifier) the data plane topology.

5.3.4 G.E-NNI routing abstract messages

In Table 5-3, there are listed information types exchanged by inter-domain routing adjacencies, which should be implemented by G.E-NNI routing protocol.

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated
24.	Grid service capability	R	O/I
25.	Grid resource availability	R	O/I
26.	Network resource availability	R	O/I
27.	Network topology information	R	O/I
28.	Network end-point assigned addresses (TNAs)	R	O/I

Table 5-3: G.E-NNI Routing Messages

5.3.5 OSPF-TE Extensions for G.E-NNI routing

The selected routing protocol for the G^{2}MPLS operations across the G.E-NNI interface is OSPF, with its standard extensions contributed by IETF for the GMPLS part and by OIF for the E-NNI parts [OIF-ENNI-OSPF].

The routing extensions and the message formats identified in [G2MPLS-EXT] are also applicable to G.E-NNI routing. Mapping of G.E-NNI routing messages into G.OSPF LSAs, TLVs and sub-TLVs are presented on Table 5-4.

Message No.	Abstract Message Name	G.OSPF LSA	G.OSPF TLVs	G.OSPF subTLVs
1.	Grid service capability	Grid LSA	Grid Site	many subTLVs
			Grid Service	many subTLVs
			Grid CE	many subTLVs
			Grid Sub-Cluster	many subTLVs
			Grid SE	many subTLVs
2.	Grid resource availability	Grid LSA	Grid Computing Element	CE calendar
			Grid Sub-Cluster	Sub-Cluster calendar
			Grid Storage Element	SE calendar
3.	Network resource availability	TE LSA	TE-link	TE-link calendar
4.	Network topology information	TE LSA	TE-link	TE-link calendar
5.	Network end-point assigned addresses (TNAs)	TE LSA	TNA address	Node ID TNA address

Table 5-4: G.E-NNI Abstract Messages mapped to OSPF advertisement parts.

Techniques of grid information summarization are described in [G2MPLS-EXT].

5.3.6 G.E-NNI routing flow

G^2.OSPF-TE protocol in G.E-NNI interface as described in [G2MPLS-EXT] can be used for sharing knowledge not only about transport network resources like Transport Network Addresses (TNAs), network topology information and network TE-links availability schedules but also Grid service and resource information. However, it should be done only when it is necessary. The message exchange follows the G2MPLS routing hierarchical model (section 5.3.2). For network information TE LSA is used to propagate network topology information, network TNAs, and network resource availability information as shown in Figure 5-18.

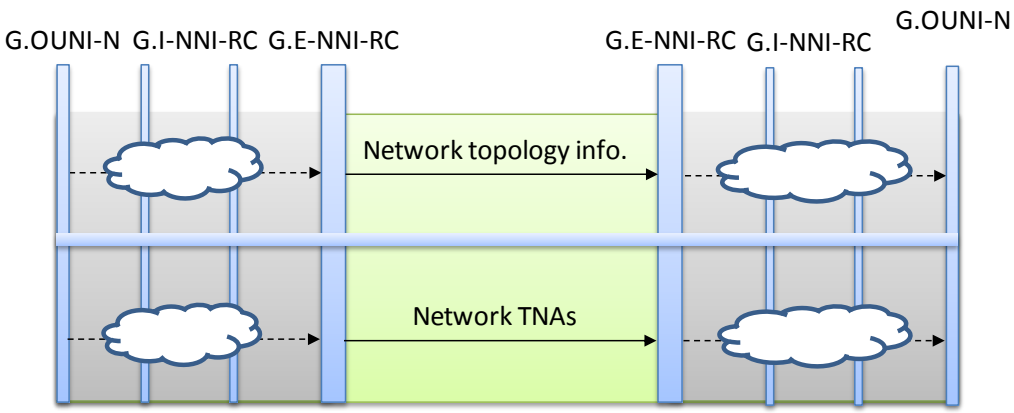

Figure 5-18: Inter-domain network information exchange.

Service capability and resource availability information is just advertised from Grid sites to the G^{2}MPLS NCP (Figure 4-26). Initiated by the G.OUNI, an OSPF update message containing the Grid Opaque LSAs can be flooded on the same domain over the G.I-NNI-RC and then over the G.E-NNI-RC can be propagated on different domains.

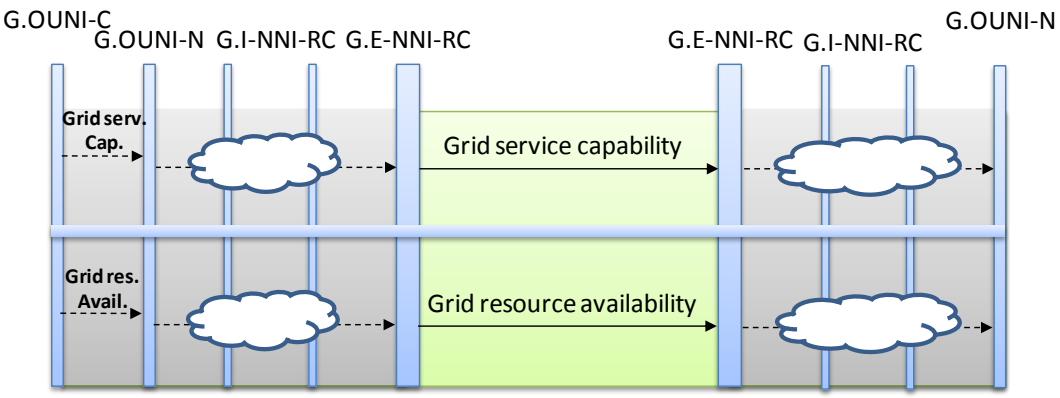

Figure 5-19 Inter-domain Grid information exchange.

6 G.I-NNI interface and functionalities

6.1 G.I-NNI signalling

The selected signalling protocol for G^{2}MPLS operations is G.RSVP-TE which applies to G.I-NNI, with its standard extensions contributed by IETF for the GMPLS part and by OIF for the UNI and E-NNI parts.

G^{2}MPLS extensions to RSVP-TE for the different G^{2}MPLS network reference points (G.OUNI, G.I-NNI, G.E-NNI) are conceived to cope with:

- Grid job/service requests translated in setup of *GNS transactions*
- support of *advance reservations*
- support of *implicit network destinations* for Calls related to a Grid service (destination is an amount of CPU or storage wherever it is)

GNS transaction is the logical container that provides a common root to different network call/connections traversing the same G^{2}MPLS User to Network reference point. It is related to the Job ID, which is a unique identifier for the job originated by the Grid middleware system (or by the user) that remains constant over the life of the job. GNS transaction is shareable in a distributed way in the G^{2}MPLS NCP, in order to enable different invocation models:

- *direct invocation model*, in which user is co-located at the same Grid site of the some Grid resources be involved in the job and issues a GNS transaction creation including also remote Grid sites;
- *indirect invocation model*, in which user is located remotely with respect to all the Grid sites to be involved in the job execution and issues a remote GNS transaction creation between them.

The submission of a job results in the setup of a GNS transaction, which in turn results in the signalling of a number of calls between the endpoints, being them explicitly selected like in standard Network Services or

implicitly inferred by the resource characterization (anycast request. Support of GNS transactions implies an extension of the CALL_ID object defined in standard ASON/GMPLS signalling.

Support of *advance reservations* in signalling may be split in two resource domains:

- head-end/far-end Grid resources
- network resources among them

Availability of resource calendars for both classes specified in [G2MPLS-EXT] can enable the enforcement of policies on advance reservation guarantee in signalling. Origin of the time schedule for the reservations is the user or the middleware, which usually specify the start time of a certain service. In case this time schedule is missing an immediate reservation is assumed. Duration of the service can be inferred by the time parameters of the job description. If not specified the resources are assumed to be reserved for long term call/connections until explicitly released. The trigger for release is always the middleware upon explicit command by the user or completion of a job in all its parts. Time guarantees are just part of SLAs specified in WebService Agreements and could also be declared as extensions to JSDL documents. Therefore, they can be mapped in specific protocol objects related to the time values of a call.

The *implicit declaration of the destination* of a job may be distinguished in two further sub-cases:

- participating Grid sites are specified, but the destination TNA is not declared. This implies that the G^{2}MPLS NCP will choose the best match in the set of the available network attachment points for the selected site. The implicit destination turns into an explicit destination in case the set contains just one TNA.
- some of the participating Grid site are implicitly declared in the GNS transaction by asking for a service they can provide (e.g. an amount of CPU or storage). This implies that the G^{2}MPLS NCP will pick at first the best site match in the set of the available Grid sites for that application and fulfilling the job requirements, and then the best match for the network attachment in the set of those available for the selected site.

The G^{2}MPLS signalling in case of implicit network destination implies that a new session must be set up from the ingress-node towards a new tunnel endpoint address when crankback and/or recovery occur in a domain. In Table 6-1, there are listed information types exchanged by intra-domain routing adjacencies, which should be implemented by G.I-NNI routing protocol. The list of abstract G.I-NNI messages is the same as for G.E-NNI interface. Similarly, the RSVP-TE protocol was used as the base protocol for introducing G.I-NNI extensions.

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated
1.	NS Create Request	S	O/I
2.	NS Create Response	S	O/I
3.	NS Create Confirmation	S	O/I
4.	NS Delete Request	S	O/I
5.	NS Delete Response	S	O/I
6.	NS Status Enquiry	S	O/I
7.	NS Status Response	S	O/I
8.	NS Notification	S	O/I
Grid Network Service (GNS) abstract messages required to support G^{2}MPLS NCP			
9.	Grid resource allocation request	S	I
10.	GNS Create Request	S	I
11.	GNS Create Response	S	I
12.	GNS Create Confirmation	S	I
13.	GNS Delete Request	S	I
14.	GNS Delete Response	S	I
15.	GNS Status Enquiry	S	I
16.	GNS Status Response	S	I
17.	GNS Notification	S	I

Table 6-1 G.I-NNI Signalling Messages.

6.2 G.I-NNI routing

The G^{2}MPLS I-NNI routing is disseminating information about resources inside Routing Control Domains (RCD) and supports GMPLS routing [IETF-RFC: 3630, 3471, 4203, 4328], and Phosphorus G^{2}MPLS requirements [G2MPLS-ARCH] (Figure 6-1).

The main entities taking part in the intra-domain routing exchange are the intra-domain Routing Controllers (G.iNNI-RC). The G.iNNI-RC residents in every Control Plane Node of the RCD, thus, there is one-to-one relation between the G.iNNI-RC and a transport plane network device. The G.iNNI-RC establishes routing adjacencies with every G.iNNI-RC placed in the RCD. Routing adjacencies are used to exchange G.I-NNI routing messages. The G.iNNI-RC gathers resources information from the local network device (network resources) and from every local G.OUNI-N module (TNAs and grid resources) and flood the information to the adjacent G.iNNI-RCs. Additionally, the G.iNNI-RC is learning the information from the adjacent G.iNNI-RC and it popularizes the information on other adjacencies. The G.iNNI-RC is placed in RA level 0 of the routing hierarchy.

To establish the intra-domain adjacency G.iNNI-RC needs:

- neighbour G.iNNI-RC ID,
- SCN address of the neighbouring G.iNNI-RC.

The G.iNNI-RC is flooding not only the local domain information but also the information from others RCDs received by the G.eNNI-RC feed-down operation. This information is used by PCE in the edge node, when calculating a path which is passing via any other domain, or in case, when a user is requesting list of globally available grid resources. The G.E-NNI messages are flooded on G.I-NNI interfaces in the same format as on G.E-NNI interfaces, thus many G.E-NNI specifications are applicable to the G.I-NNI interface too.

In order to configure any TE-link in the G.iNNI-RC, the G.iNNI-RC must have configured:

- remote G.iNNI-RC ID of the TE-link,
- TE-link local interface ID,
- TE-link remote interface ID.

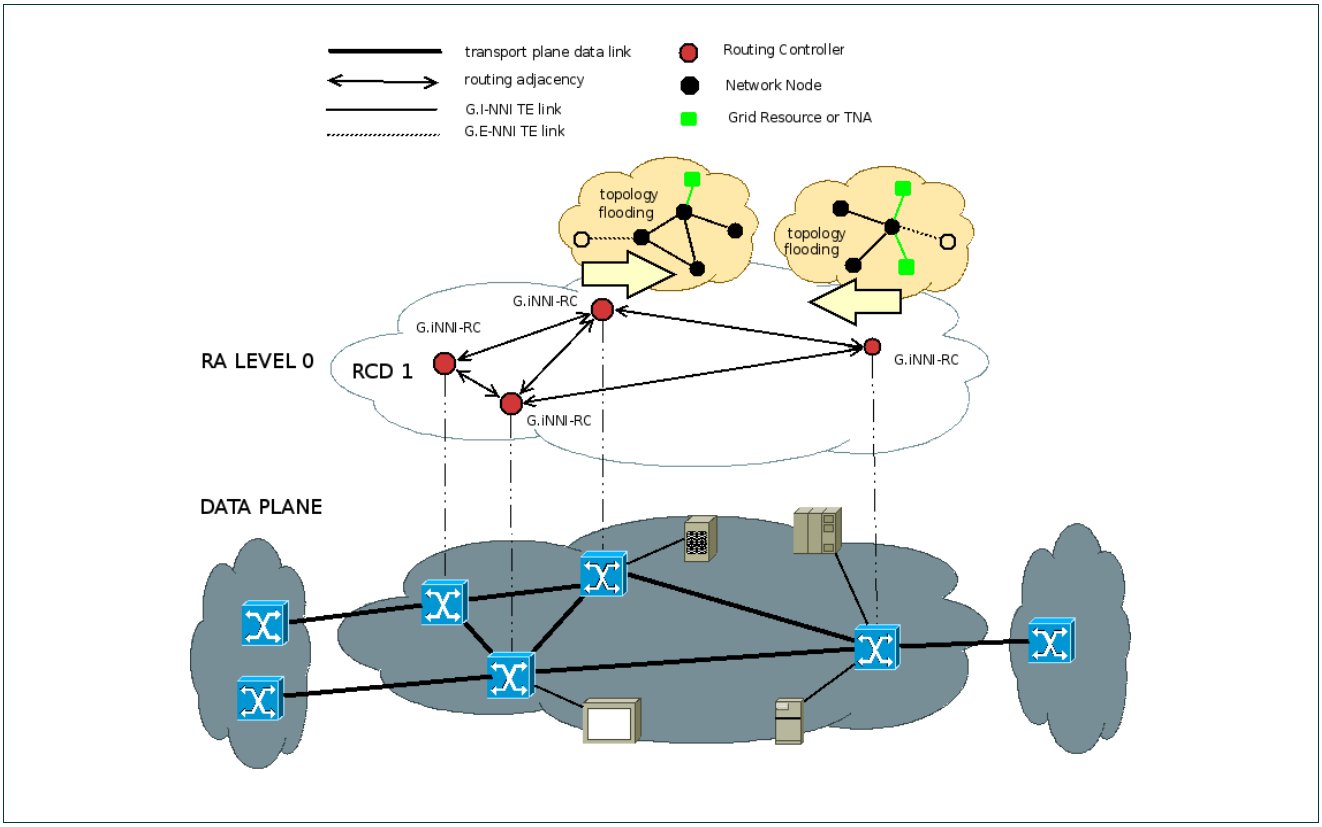

Figure 6-1: G.I-NNI routing (without showing of global network topology flooding).

The G.iNNI-RC of the Area Border Router (ABR) can flood the information about all connected G.E-NNI TE-links using G.I-NNI adjacencies and this information should be available in every G.iNNI-RC of the RCD. The G.eNNI-RC should be able to receive the G.E-NNI TE-links connected to the local RCD by feed-up from the RA level 0. If a border G.iNNI-RC must flood information about G.E-NNI TE-links, the some additional configuration should configured in the G.iNNI-RC:

- Node ID of remote ABR,
- G.eNNI-RC ID responsible for the neighbouring RCD.

In Table 6-2, there are listed information types exchanged by intra-domain routing adjacencies, which should be implemented by G.I-NNI routing protocol.

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated

Message No.	Abstract Message Name	Signalling/ Routing	Overlay/ Integrated
1.	Grid service capability	R	O/I
2.	Grid resource availability	R	O/I
3.	Network resource availability	R	O/I
4.	Network topology information	R	O/I
5.	Network end-point assigned addresses (TNAs)	R	O/I

Table 6-2: G.I-NNI Routing Messages

The list of abstract G.I-NNI messages is the same as for G.E-NNI interface. Similarly, the OSPF-TE protocol was used as the base protocol for introducing G.I-NNI extensions. The mapping of G.I-NNI routing messages into G.OSPF LSAs, TLVs and sub-TLVs are the same as for G.E-NNI, which is presented on Table 5-4.

7 Conclusions

This document is the final release of the Grid-GMPLS network interface specifications. The document provides an overview of the PHOSPHORUS control plane architectures (overlay and integrated) described in [G2MPLS-ARCH] and [G2MPLS-DEP] with regards to interfaces and an overview of the procedures and the services supported by each of them.

Then, G.OUNI, G.E-NNI and G.I-NNI are introduced by describing their role to the PHOSPHORUS G^{2}MPLS control plane. A description of Grid and network services that should be supported and the protocol extensions required to realise the G^{2}MPLS control plane are provided. A high level description of Grid and network services offered over all reference points are reported. Abstract messaging description for signalling, routing and discovery procedures is provided to map JSDL to RSVP and GLUE schema to OSPF protocols accordingly (and optionally LMP), which are then propagated over inter-domain G2MPLS control plane. This deliverable has driven the low-level design and development of the interfaces prototypes and also [G2MPLS-EXT] where detailed description of protocols extensions is provided.

8 References

8.1 Normative references

[GLUE] S. Andreozzi (Ed.), “GLUE Schema Specification – version 1.3”, Draft 3 – 16 Jan 2007, http://glueschema.forge.cnaf.infn.it/Spec/V13

[IETF-RFC2205] R. Braden (Ed.), “Resource Reservation Protocol (RSVP) – Version 1 Functional Specification”, IETF RFC 2205, September 1997, http://www.ietf.org/rfc/rfc2205.txt

[IETF-RFC2370] R. Coltun, “The OSPF Opaque LSA Option”, IETF RFC 2370, July 1998, http://www.ietf.org/rfc/rfc2370.txt

[IETF-RFC2748] D. Durham (Ed.), “The COPS (Common Open Policy Service) Protocol”, IETF RFC 2748, January 2000, http://www.ietf.org/rfc/rfc2748.txt

[IETF-RFC2903] Laat de, C., G. Gross, L. Gommans, J. Vollbrecht, D. Spence, "Generic AAA Architecture,” Experimental IETF RFC 2903, August 2000, http://www.ietf.org/rfc/rfc2903.txt

[IETF-RFC2904] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, "AAA Authorization Framework", IETF RFC 2904, August 2000, http://www.ietf.org/rfc/rfc2904.txt

[IETF-RFC3630] K. Katz (Ed.), “Traffic Engineering (TE) Extensions to OSPF Version 2”, IETF RFC 3630, September 2003, http://www.ietf.org/rfc/rfc3630.txt

[IETF-RFC4203] K. Kompella (Ed.), “OSPF Extensions in Support of Generalized Multi-Protocol Label Switching”, IETF RFC 4203, October 2005.

[IETF-RFC4204] J. Lang (Ed.), “Link Management Protocol (LMP)”, IETF RFC 4204, October 2005, http://www.ietf.org/rfc/rfc4204.txt

[IETF-RFC4940] K. Kompella, B. Fenner, “IANA Considerations for OSPF”, IETF RFC 4940, July 2007

[ITU-T-X.812] ITU-T Rec. X.812 (1995) | ISO/IEC 10181-2:1996, Information technology - Open systems interconnection - Security frameworks in open systems: Authentication framework. [Online], http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.811-199504-I!!PDF-E&type=items

[JSDL-SPEC] Job Submission Description Language specification, http://www.gridforum.org/documents/GFD.56.pdf

[OIF-UNI1.0] Optical Internetworking Forum, “User Network Interface (UNI) 1.0 Signalling Specification”, http://www.oiforum.com/public/documents/OIF-UNI-01.0.pdf

[OIF-UNI1.0R2-COMM] Optical Internetworking Forum, “UNI 1.0 Signaling Specification, Release 2: Common Part”, http://www.oiforum.com/public/documents/OIF-UNI-01.0-R2-Common.pdf

[OIF-UNI1.0R2-RSVP] Optical Internetworking Forum, “RSVP Extensions for User Network Interface (UNI) 1.0 Signaling, Release 2”, http://www.oiforum.com/public/documents/OIF-UNI-01.0-R2-RSVP.pdf

[OIF-UNI2.0-COMM] Optical Internetworking Forum, “UNI 2.0 Signaling Specification: Common Part”, draft OIF2005.204.02, work in progress

[OIF-UNI2.0-RSVP] Optical Internetworking Forum, “RSVP Extensions for User Network Interface (UNI) 2.0 Signaling”, draft OIF2005.205.01, work in progress

[OIF-E-NNI-Sig-1.0] Optical Interworking Forum, “Intra-Carrier E-NNI 1.0 Signaling Specification”, http://www.oiforum.com/public/documents/OIF-E-NNI-Sig-01.0-rev1.pdf.

[OIF-E-NNI-Rtr-1.0] Optical Interworking Forum, “External Network-Network Interface (E-NNI) OSPF-based Routing - 1.0”, http://www.oiforum.com/public/documents/OIF-ENNI-OSPF-01.0.pdf.

[WSA] Web Service Addressing, http://www.w3.org/2002/ws/addr

[WSAG-SPEC] Web Service Agreement specification, http://www.ogf.org/documents/GFD.107.pdf

[WSRF] Web Service Resource Framework, http://www.oasis-open.org/committees/wsrf

[WSRP-SPEC] Web Service Resource Properties specification, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf

8.2 Informational reference

[AAA-ARCH] PHOSPHORUS WP4, “AAA Architectures for multi-domain optical networking scenario's”, deliverable D4.1

[DEMCHENKO] Demchenko Y., L. Gommans, C. de Laat, “Using SAML and XACML for Complex Authorisation Scenarios in Dynamic Resource Provisioning”, in Proc. The Second International Conference on Availability, Reliability and Security (ARES 2007), Vienna, Austria, April 10-13, 2007. IEEE Computer Society, ISBN: 0-7695-2775-2, pp. 254-262

[G2MPLS-ARCH] PHOSPHORUS WP2, “The Grid-GMPLS Control Plane architecture”, deliverable D2.1

[G2MPLS-DEP] PHOSPHORUS WP2, “Deployment Models and Solutions of the Grid-GMPLS Control Plane”, deliverable D2.6

[G2MPLS-EXT] PHOSPHORUS WP2, “Grid-GMPLS network interfaces specification”, deliverable D2.2

[GOMMANS] "Token-based authorization of connection oriented network resources", by Leon Gommans, Franco Travostino, John Vollbrecht, Cees de Laat, and Robert Meijer, in Proceedings of GRIDNETS, San Jose, CA, USA, Oct 2004.

[OGF-GDF36] D. Simeonidou, R. Nejabati (Editors), “Optical Network Infrastructure for Grid”, GFD-I-036, August 2004, http://forge.gridforum.org/projects/ghpn-rg/

[OGF-G.OUNI] Grid User Network Interface (GUNI), Open Grid Forum (OGF), Grid High-Performance Networking (GHPN) Group, Informational track draft, May 2007 https://forge.gridforum.org/sf/docman/do/listDocuments/projects.ghpn-rg/docman.root.current_drafts

[ZHANG] Xinwen Zhang, Masayuki Nakae, Michael J. Covington, and Ravi Sandhu, A Usage-based Authorization Framework for Collaborative Computing Systems, in the proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT), 2006

9 Appendix

Advance Reservation Template

```
<?xml version="1.0"?>
<wsag:Template wsag:AgreementId=""
 wsag:TemplateId="PHOSPHORUS_ADVANCE_RESERVATION_TEMPLATE"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl">
 <wsag:Context>
  <wsag:ServiceProvider>AgreementInitiator</wsag:ServiceProvider>
<wsag:TemplateId>PHOSPHORUS_ADVANCE_RESERVATION_TEMPLATE</wsag:TemplateId>
 </wsag:Context>
 <wsag:Terms>
  <wsag:All>
   <wsag:ServiceDescriptionTerm wsag:Name="ATOMIC_SERVICE"
    wsag:ServiceName="COMPUTE">
    <jsdl:JobDefinition>
     <jsdl:JobDescription>
      <jsdl:Resources>
       <jsdl:TotalResourceCount>
        <jsdl:Exact>1.0</jsdl:Exact>
       </jsdl:TotalResourceCount>
      </jsdl:Resources>
     </jsdl:JobDescription>
    </jsdl:JobDefinition>
```

```
    </wsag:ServiceDescriptionTerm>
    <wsag:GuaranteeTerm wsag:Name="ADVANCE_RESERVATION_GUARANTEE">
      <wsag:ServiceScope wsag:ServiceName="COMPUTE_SERVICE" />
      <wsag:ServiceLevelObjective>
        <wsag:KPITarget>
          <wsag:KPIName>ADVANCE_RESERVATION</wsag:KPIName>
          <wsag:CustomServiceLevel>
            <ext:ReservationTime>
              2007-03-15T11:53:41.921+01:00
            </ext:ReservationTime>
          </wsag:CustomServiceLevel>
        </wsag:KPITarget>
      </wsag:ServiceLevelObjective>
      <wsag:BusinessValueList />
    </wsag:GuaranteeTerm>
  </wsag:All>
</wsag:Terms>
  <wsag:CreationConstraints>
  <wsag:Item>
    <wsag:Location>//wsag:GuaranteeTerm/wsag:ServiceLevelObjective
                    /wsag:KPITarget/wsag:CustomServiceLevel
                    /ext:ReservationTime
    </wsag:Location>
    <wsag:ItemConstraint>
      <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:union>
          <xs:simpleType>
            <xs:restriction base="xs:dateTime">
              <xs:minInclusive value="2007-08-10T17:00:00+02:00" />
              <xs:maxInclusive value="2007-08-10T123:00:00+02:00" />
            </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:minInclusive value="2007-08-10T23:00:00+02:00" />
              <xs:maxInclusive value="2007-12-24T18:00:00+02:00" />
```

```
            </xs:restriction>
          </xs:simpleType>
        </xs:union>
      </xs:simpleType>
    </wsag:ItemConstraint>
  </wsag:Item>
  <wsag:Item>
    <wsag:Location>//jsdl:JobDefinition/jsdl:JobDescription
                              /jsdl:Resources/jsdl:TotalCPUCount/jsdl:Exact
    </wsag:Location>
    <wsag:ItemConstraint>
      <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:restriction base="xs:double">
          <xs:minInclusive value="1" />
          <xs:maxInclusive value="128" />
        </xs:restriction>
      </xs:simpleType>
    </wsag:ItemConstraint>
  </wsag:Item>
 </wsag:CreationConstraints>
</wsag:Template>
```

<END-OF-DOCUMENT>