034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D.2.9

Design of Grid-GMPLS interworking with NRPS

Due date of deliverable: 2008-09-30
Actual submission date: 2008-09-30
Document code: Phosphorus-WP2-D2.9

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	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Design of Grid-GMPLS interworking with NRPS

Abstract

This deliverable, named “Design of Grid-GMPLS interworking with NRPS”, reports on the interworking architectures, interoperability scenarios between G^{2}MPLS and Harmony system. It also defines the signalling and routing specifications of the HG2 GW (Harmony G^{2}MPLS gatetway) and methods that have been implemented to provide this gateway. The work shown here is the outcome of the WP2-WP1 collaboration.

List of Contributors

Eduard Escalona	UEssex
Georgios Zervas	UEssex
Reza Nejabati	UEssex
Dimitra Simeonidou	UEssex
Alexander Willner	UoB
Christian de Waal	UoB
Jordi Ferrer	I2cat
Nicola Ciulli	NXW
Gino Carrozzo	NXW
Giacomo Bernini	NXW
Damian Parniewicz	PSNC

Reviewed and commented on behalf of WP5 by: Chris Develder IBBT

Table of Contents

Table of Figures

List of Tables

0 Executive Summary

This deliverable, named “Design of Grid-GMPLS interworking with NRPS”, reports on the interworking architectures and interoperability scenarios between G^{2}MPLS and Harmony system. It also defines the signalling and routing specifications of the HG2 GW (Harmony-G^{2}MPLS gateway) and methods that have been implemented to provide this gateway.

G²MPLS and Harmony share the final goal of the automatic setup of inter-domain connections, when G²MPLS is run in the Phosphorus Overlay mode, i.e. when G²MPLS sets up just Network Services instead of Grid Network Services. In this context, the two Control Plane approaches have many commonalities and their interworking is the topic of this deliverable. It deals with architectural considerations derived from two identified interoperability scenarios a) Northbound interaction, which is an overlay-style control of GMPLS/G²MPLS by Harmony/NRPS and b) East-West interaction, which implies a peering between G²MPLS and Harmony.

Signalling and routing specifications for both G²MPLS and Harmony system have been defined including internal entities and messages and compatibility issues. Detailed description of methods/system blocks used, list of messages and attributes exchanged for implementation purposes on Harmony system and G²MPLS are reported. These consider all functions required for the signalling and routing interactions from Harmony to G²MPLS and vice versa.

The remained of this deliverable is organised as follows. First, section 1 reports on G^{2}MPLS control plane in the PHOSPHORUS framework that also includes interoperability considerations with HARMONY system and discuss about the architecture. Then, an overview of the Harmony system is provided in section 1.2 providing information about the architecture and some initial information about the gateway towards interoperability with other systems. Section 1.3 provides detailed information regarding the interworking architecture between G^{2}MPLS control plane and Harmony system providing two different interoperability scenarios. The second scenarios include three different cases out of which the third one is selected from implementation. Then, detailed information about the signalling and routing specifications are provided - to support the third case of the second scenario - including the HG^2 GW (Harmony G^{2}MPLS gateway) message flows. Furthermore, in section 3 low level details that are used for implementation purposes and have to do with HG^2 GW Web Service, HG^2 GW Corba Client/Servant and the HG^2 GW translator. Finally section 4 summarizes our conclusions.

1 Interworking Architecture

1.1 G²MPLS Control Plane

In the PHOSPHORUS framework, different layers are identified as shown in Figure 1-1:

- Grid layer,
- G²MPLS Network Control Plane, Harmony System
- Transport Plane.

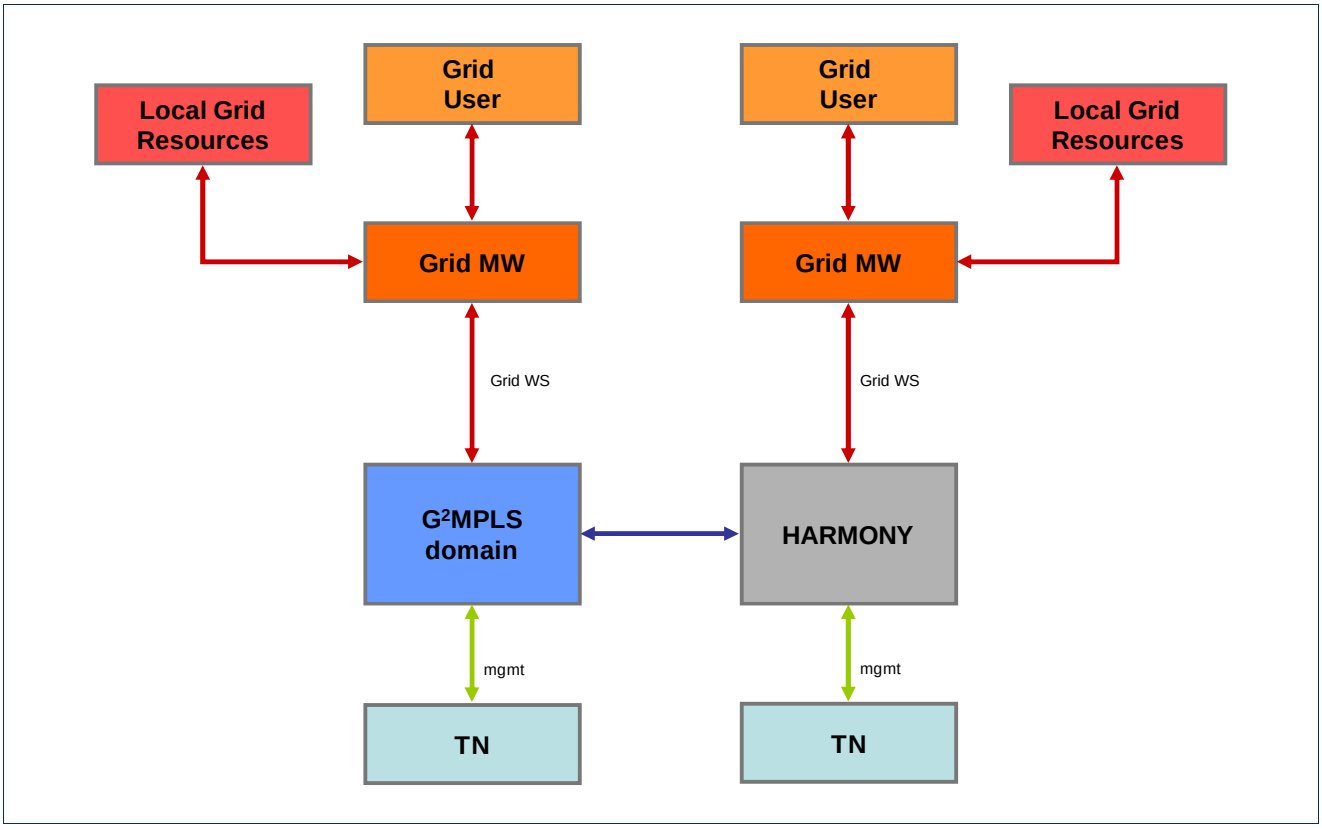

Figure 1-1: G²MPLS positioning in the PHOSPHORUS framework including HARMONY system.

The Transport Plane is the basic layer comprising all the data bearing equipments and their configuration interfaces.

G^{2}MPLS Network Control Plane is aimed to provide:

- *Discovery and advertisement* of Grid capabilities and resources of the participating Grid sites (*Vsites*);
- *Grid and Network Service setup* including:
  - *Coordination* with the Grid local job scheduler in the middleware responsible for the local configuration and management of the Grid job;
  - *Configuration* of the network connections among the Vsites participating to the Grid job;
  - *Management of resiliency* for the installed network services and possible escalation to the Grid middleware components that could be responsible for check-pointing and recovering the whole job;
  - *Advanced reservation*s of Grid and network resources;
- *Service monitoring* both for the Grid job and the related network connections.

The Grid layer is intended to comprise both Grid application and Grid middleware. Discussion on the architecture of the Grid layer is out of the scope in this document. The relevant aspect of this layer in a G^{2}MPLS perspective is the functionalities exported to/by the underlying network Control and Management Planes. G^{2}MPLS is primarily intended to interconnect remote instances of the Grid middleware, which are responsible for managing Grid resources localized in different sites (ref. Figure 1-1). In a more visionary scenario and in support of possible future applications, G^{2}MPLS could also interconnect directly Grid users/applications and Grid resources. Moreover, the southbound interface of the Grid layer represents a technological boundary, not necessarily an administrative boundary. This reference point cannot be addressed with standard IETF peer/integrated models, because any Grid component in the application or even in the middleware cannot peer with any Control Plane instance running on a network node.

Nevertheless, G^{2}MPLS is required to be deployed in structured, heterogeneous and multi-domain networks, possibly using other non-GMPLS technologies. The inter-domain natural bent of G^{2}MPLS raises the issue of the interaction and cooperation with this technology.

Discussions on the G^{2}MPLS interworking with the Harmony system are provided in section 1.3 onwards.

1.2 Harmony

The Harmony system is a multi-domain, multi-vendor network resource brokering system. Harmony, developed under the European Union co-funded project PHOSPHORUS, defines an architecture for a service layer between the Grid middleware and applications and the Network Resource Provisioning Systems (NRPS).

Harmony was born with two main assumptions: the system had to be multi-domain and had to be capable of creating end-to-end optical and layer 2 paths in a seamless environment for the scientific personnel at the end points. These conditions were set because PHOSPHORUS aims to make existing provisioning systems

(NRPS) interoperable and fill the gap between these hardware-coupled software pieces and the Grid application middleware. The NRPSs compatible with Harmony at the current implementation stage are:

- ARGIA/UCLP: stands for the User Controlled Light-Paths initiative, developed by CRC, Inocybe Technologies (both from Canada) and i2CAT Foundation (Catalonia, Spain). It provides a network virtualization framework upon which communities of users can build their own services or applications. Articulated Private Network (APNs) are presented as the first services. APNs can be considered as a next generation Virtual Private Network where a user can create a complex, multi-domain network topology by binding together network resources, time slices, switching nodes and virtual or real routing services.

- ARGON: stands for the Allocation and Reservation in Grid-enabled Optic Networks. It was developed to manage resources of advanced network equipment as it is present in the German VIOLA test-bed. The advance reservation service of ARGON is able to operate on the GMPLS as well as on the MPLS level. It guarantees the requested level of QoS for applications for the requested time interval. This feature enables a Meta-Scheduling Service to seamlessly integrate the network resources into a Grid environment..

- DRAC: stands for the Dynamic Resource Allocation Controller initiative. It is the world's first commercial-grade network abstraction and mediation middleware platform, acting as an agent for network clients to negotiate and reserve appropriate network resources on their behalf. DRAC uses client's QoS requirements and pre-defined policies to negotiate end-to-end connectivity across heterogeneous domains in support of just-in-time or scheduled computing workflows.

Moreover, Harmony has developed a *Thin NRPS* entity which is able to communicate with the Optical User to Network Interface (OUNI) of a Generalized-MPLS Control Plane (GMPLS-CP) –although it is not considered an NRPS itself– and perform signaling operations and path management

The integration between application middleware and optical transport networks pretended by Harmony has been successfully achieved based on an architecture with three planes: the Network Service Plane, the Network Resource Provisioning Plane and the Control Plane. This architecture is built over a SOA model and its compliant with the Web Service Resource Framework version 1.2. The Figure 1-2 depicts the architecture of the system and shows the above mentioned layers and their components.

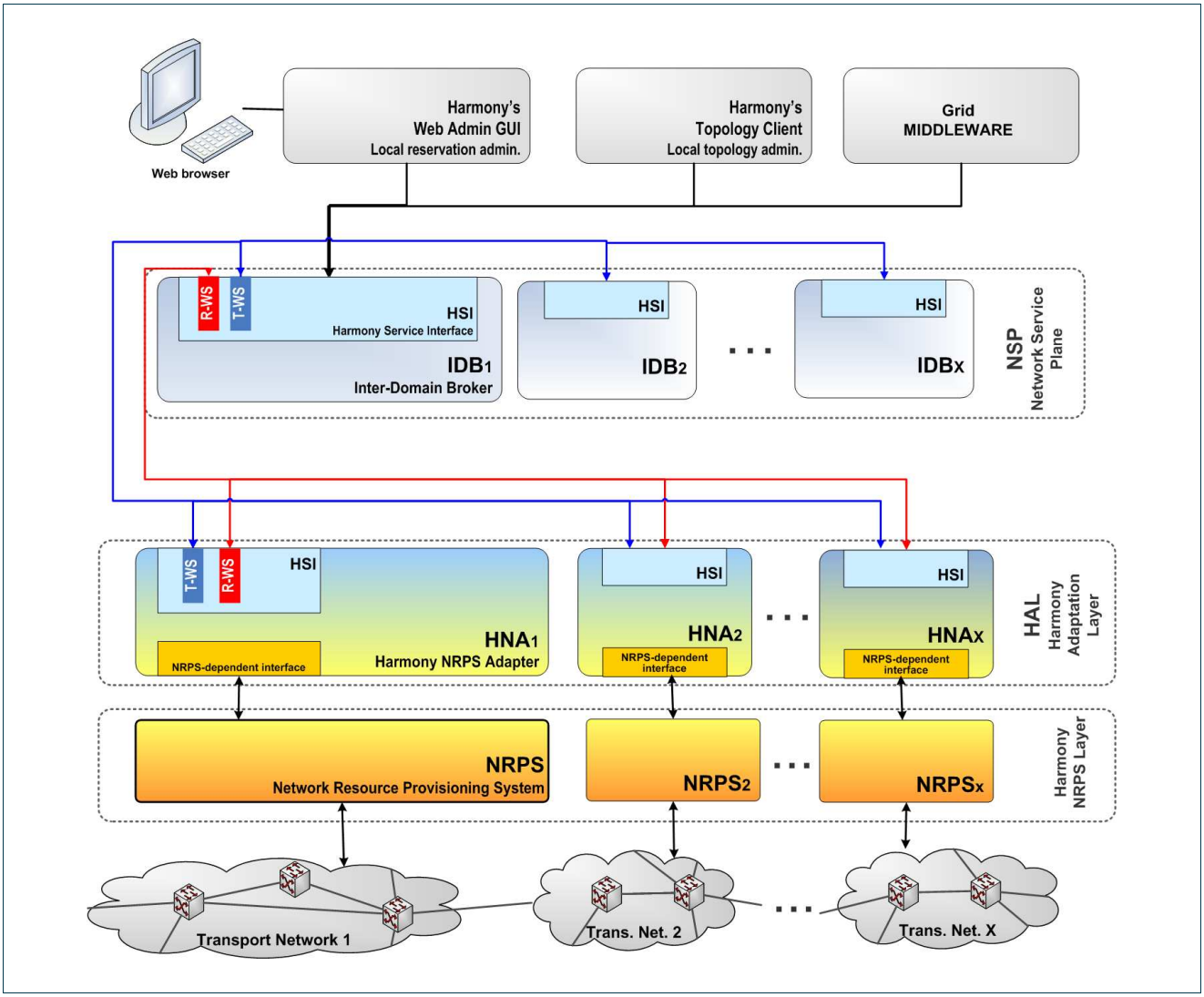

Figure 1-2: Harmony architecture

Harmony service interface (HSI) is the component that enables the communication between the distinct layers which compose the global architecture of the system. This service interface is common both for the network service layer and the adaptation layer. The HS contains three web services: Reservation-WS, Topology-WS and Notification-WS. Moreover, there is a common part of the three Web Services inside the whole interface. This common part defines the data type common for the three components.

Nowadays, there are several challenges beyond the end-to-end, multi-domain path provisioning. Thus, there are several research projects towards this topic. For any integration between Harmony and any other system is necessary to build a gateway, with the Harmony service interface on the one hand and the interface of the other system on the other hand. The principle behind the integration is a translation principle; this means that the gateway maps the requests in one system-language to the other system-language, making possible the communication between the two different systems

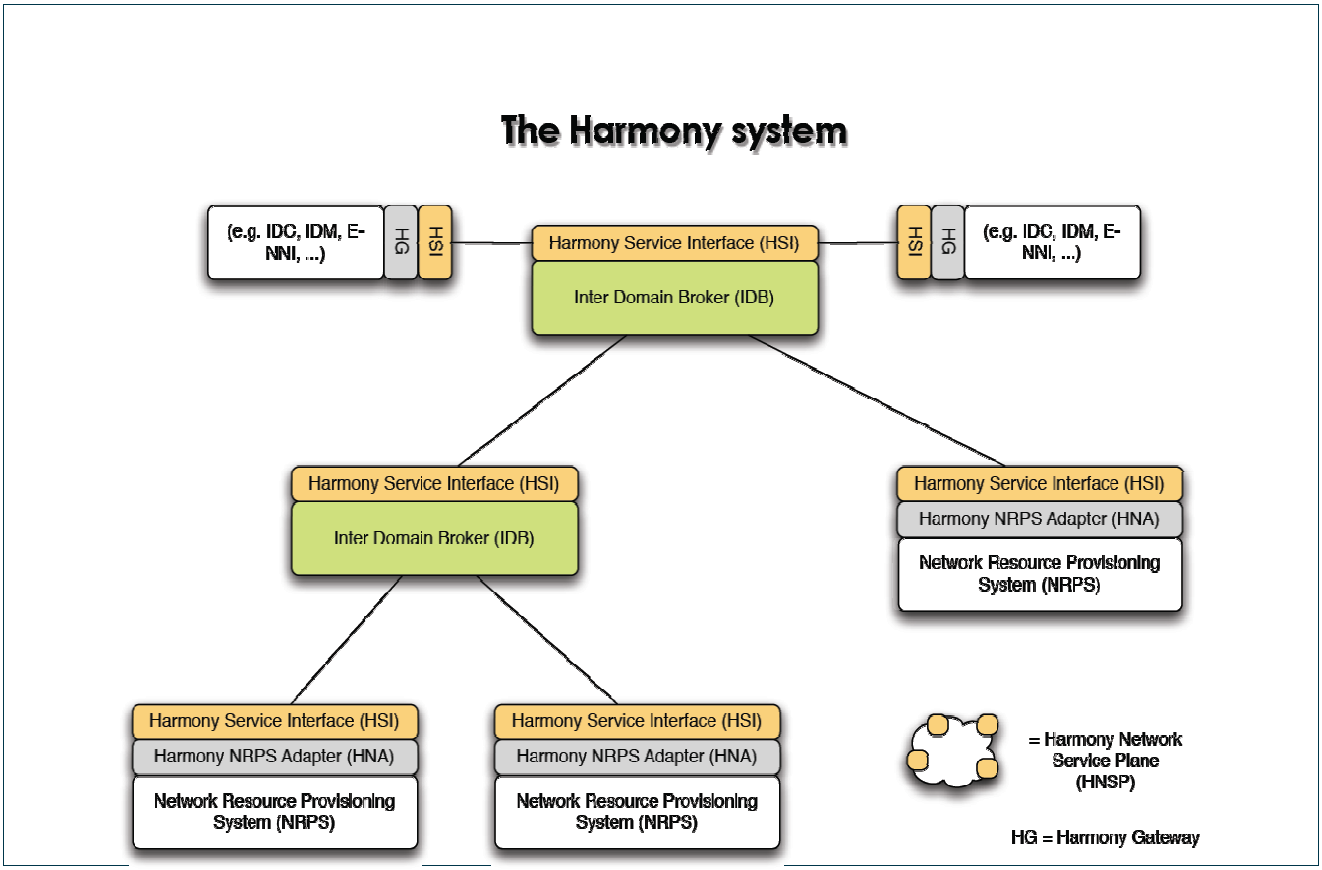

Figure 1-3: Harmony gateway towards interoperability with other systems

1.3 G²MPLS - Harmony interworking

The Phosphorus project collects among its technologies two different architectural approaches for the Control Plane: the Grid-enabled GMPLS (G²MPLS) deliveredF by WP2, and the Harmony Network Service Plane delivered by WP1 briefly described in sections 1 and 2. These approaches originate from different research objectives. On the one hand, G²MPLS implements the seamless integration of Grid service setup dynamics with the automatic procedures for the call/connection setup and recovery; G²MPLS capitalizes on the standardized protocols of the ASON/GMPLS Control Plane and brings a complete Control Plane architecture widely deployed in commercial networks into the NRENs optical infrastructures. On the other hand, Harmony implements an orchestration layer among Network Resource Provisioning Systems (UCLP, Argon, D-RAC) in order to coordinate the setup of inter-domain connections between different control technologies; NRPS-es are centralized domain-wide control solutions, which mainly provide reservations between two endpoints by directly configuring the equipments. NRPS-es are traditionally conceived by/for NRENs, which are dense of switching capabilities in small sized topologies and, thus, have been more interested in the automatic setup of connections (in advance, if necessary) than in their survivability and crankback.

Despite of the different frameworks, G²MPLS and Harmony share the final goal of the automatic setup of inter-domain connections, above all when G²MPLS is run in the Phosphorus Overlay mode [PH-WP2-D2.1], i.e. when G²MPLS sets up just Network Services instead of Grid Network Services[1]. In this context, the two Control Plane approaches have many commonalities and their interworking is a challenging though reasonable objective, above all for NRENs environments.

This chapter deals with architectural issues that have been solved for the interworking between G²MPLS and Harmony. These issues derive from two identified interoperability scenarios, as discussed further in Section 1.4:

- Northbound interaction, which is an overlay-style control of GMPLS/G²MPLS by Harmony/NRPS
- East-West interaction, which implies a peering between G²MPLS and Harmony,

In the first scenario, i.e. Northbound, G²MPLS runs as standard GMPLS with the scope limited to a single domain. In this case, the architectural issues are limited to the signalling part, since just the setup of reservations, possibly in advance, is requested by Harmony/NRPS. On the contrary, the interoperability is particularly challenging in the latter scenario, i.e. East-West, because the Control Plane peering implies shareable descriptions of topology elements and the stitching of the two connection reservation processes.

As a result of the joint analysis of the two architectural models, a basic compatibility has been identified for the reservation setup signalling models (both based on request/response phases) and routing models (both hierarchical and with Traffic Engineering). The substantial differences occur in the addressing spaces and in the invocation methods (both parameter contents and number of tiers), which imply the use of an adaptation function.

Moreover, the two Control Planes adopt different technologies (IP protocols for G²MPLS and Web-Services for Harmony) and, thus, a mediation function is needed to keep the two reservation sessions synchronized and alive, and eventually allow the communication between the two decision entities.

For all these purposes, a specific gateway module (Harmony-G²MPLS gateway, HG²-GW) has been identified and designed, aimed to provide the needed translations and mediations between the routing and signalling entities of G²MPLS and Harmony Control Planes. This module exposes a Web-Service interface towards Harmony and a CORBA interface towards G²MPLS, in a way similar to the approach used for interfacing the Grid Middleware to the G²MPLS through the G.OUNI Gateway (ref. [PH-WP2-D2.3]).

[1] As defined in [PH-WP2-D2.1], GNS is a service that allows the provisioning of network and Grid resources in a single-step, through a set of seamlessly integrated procedures. G²MPLS indirect calls and anycasting calls are also GNS-es, since they derive from specific requirements of the Grid applications/middleware.

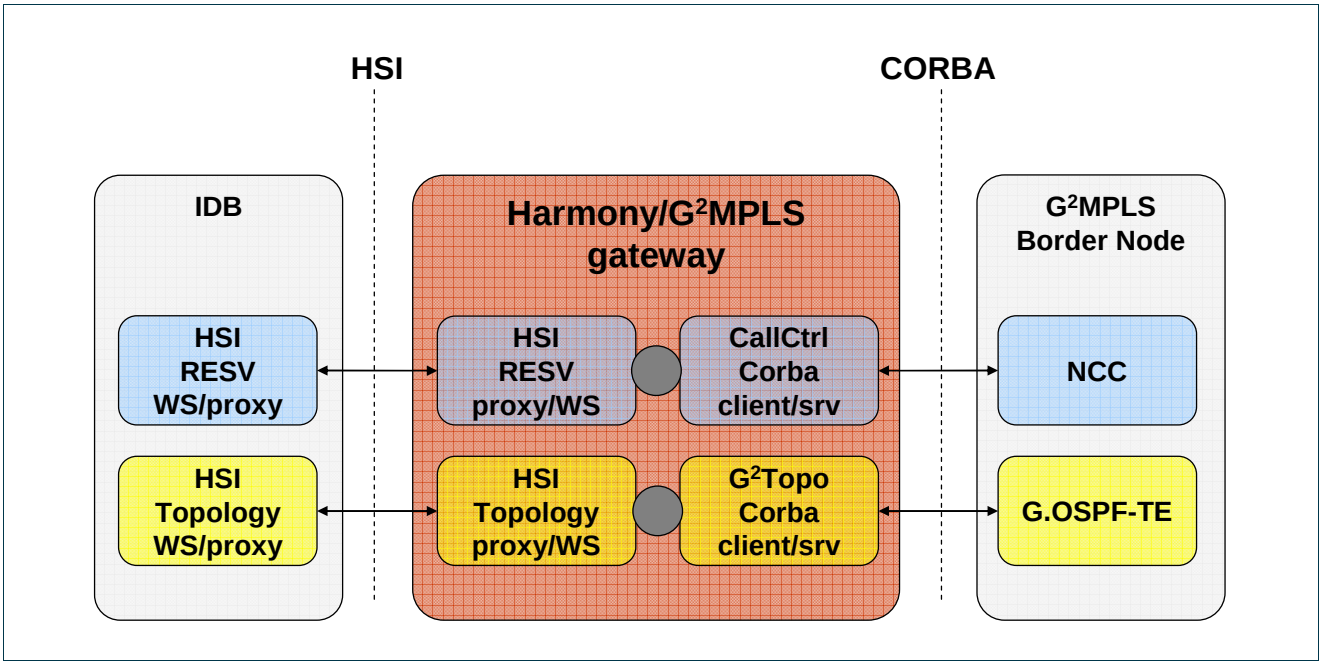

Figure 1-4: Harmony G²MPLS gateway functional decomposition and external interfaces.

The reminder of this chapter provides details about the identified interoperability scenarios between G²MPLS and Harmony, their respective signalling and routing specifications by focusing on the compatibilities, the specification of the message flow in the HG²-GW for the reservation process and the topology exchange.

1.4 Interoperability scenarios

Different interoperability scenarios have been identified and discussed for the communication between G²MPLS and Harmony/NRPS. Two main cases show the scenarios with regards to the roles assigned to each of the actors, Northbound interfacing and East-West interfacing:

- *Northbound interfacing*
  - *Case A*: standard GMPLS as the only controller of a domain interfaced to the Harmony/NSP through a gateway module,
  - *Case B:* standard GMPLS as “slave” controller of a domain mastered by an NRPS;
- *East-West interfacing*
  - *Case C: full-fledged G²MPLS as the only controller of its Grid & network resources and peering with a neighbouring Harmony NSP*

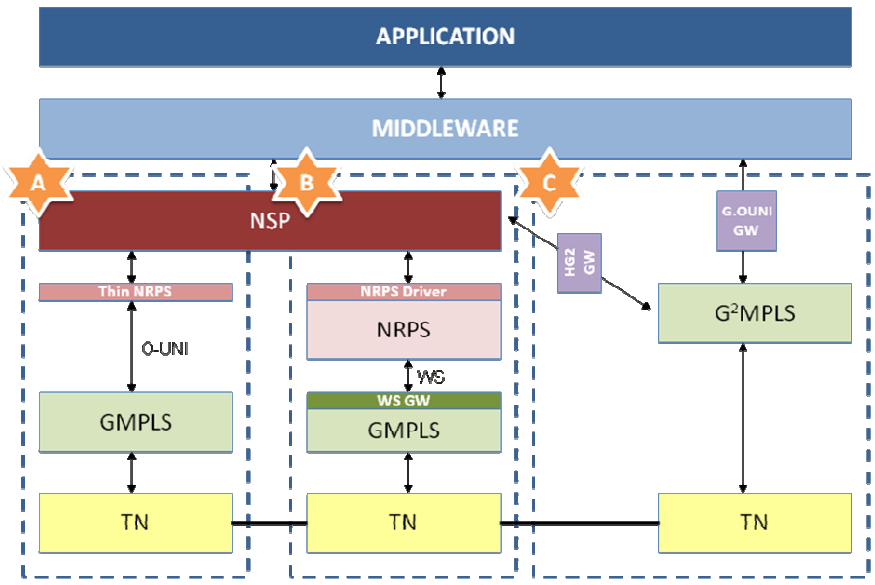

Figure 1-5: GMPLS/G²MPLS interworking with Harmony

1.4.1 Interworking through northbound interface

Harmony northbound services establish and release network circuits and retrieve topology and connections status information. Therefore, the external interface with G^2MPLS is limited to operate just with network resources as a standard GMPLS Control Plane. [PH-WP2-D2.1] shows the possible options with use cases.

In the first of the identified cases (A), an NRPS driver acting as a “thin NRPS” translates incoming requests from the Network Service Plane (NSP) into standard O-UNI RSVP messages, establishing Soft Permanent Connections (SPC) between defined end points which are represented as TNAs. In this case, advance reservations are just managed at the NSP layer. Here, GMPLS performs the path computation to find the best route so the NSP assumes that there is always an available path between the end points to enable advance reservations; then, the thin NRPS issues instant connection requests to the underlying GMPLS just before the start time of each advance reservation. This approach has been implemented and described in [PH-WP1-D1.3].

The second case (B) supposes that an NRPS is the only entity managing the GMPLS domain. Here, path computation is done at the NRPS layer, and the explicit route is passed to GMPLS, therefore, NRPS must know the full topology information of the network. Since O-UNI does not support topology information retrieval, a proposed solution is to implement the interface using Web Services based on a simple request/response model. Advance reservations can be provided at the NRPS layer since it is the only manager of the GMPLS controlled network and no misalignments in the calendars may occur. Deliverable [PH-W/P2-D2.1] illustrates two different approaches for this case, a centralized one, in which the WS Server is placed inside the NRPS building a GMPLS instance and using I-NNI to communicate with the rest of GMPLS nodes, and a distributed

one, in which the WS servant is co-located with each edge GMPLS network element and accesses connection and topology services via an API.

1.4.2 Interworking through east/west interfaces

The interworking between G²MPLS and Harmony systems can happen through the G²MPLS east/westbound interfaces when the G²MPLS domain and the NSP are considered to be at the same level. Nevertheless, transactions can just involve network services since even if G²MPLS controls network and Grid resources, the scope of Harmony is limited to support just network services. The completion of the Grid transactions is up to the Grid Middleware layer through dedicated communications.

Three different solutions have been identified to be adopted in this scenario. The main issue of this approach is that on one side, the Harmony system uses Web Services as the external interface while on the other side G²MPLS uses E-NNI protocols. G²MPLS can also expose a Corba interface to directly access the border node processes.

The first option considers an external language to use by both sides (Harmony and G²MPLS) (Figure 1-6). Here, two gateways are needed to implement the chosen interface in Harmony and in the G²MPLS border controller. The main drawback of this solution is that two adapters have to be implemented to translate calls and parameters into the new interface, which has to be carefully selected to support the services required.

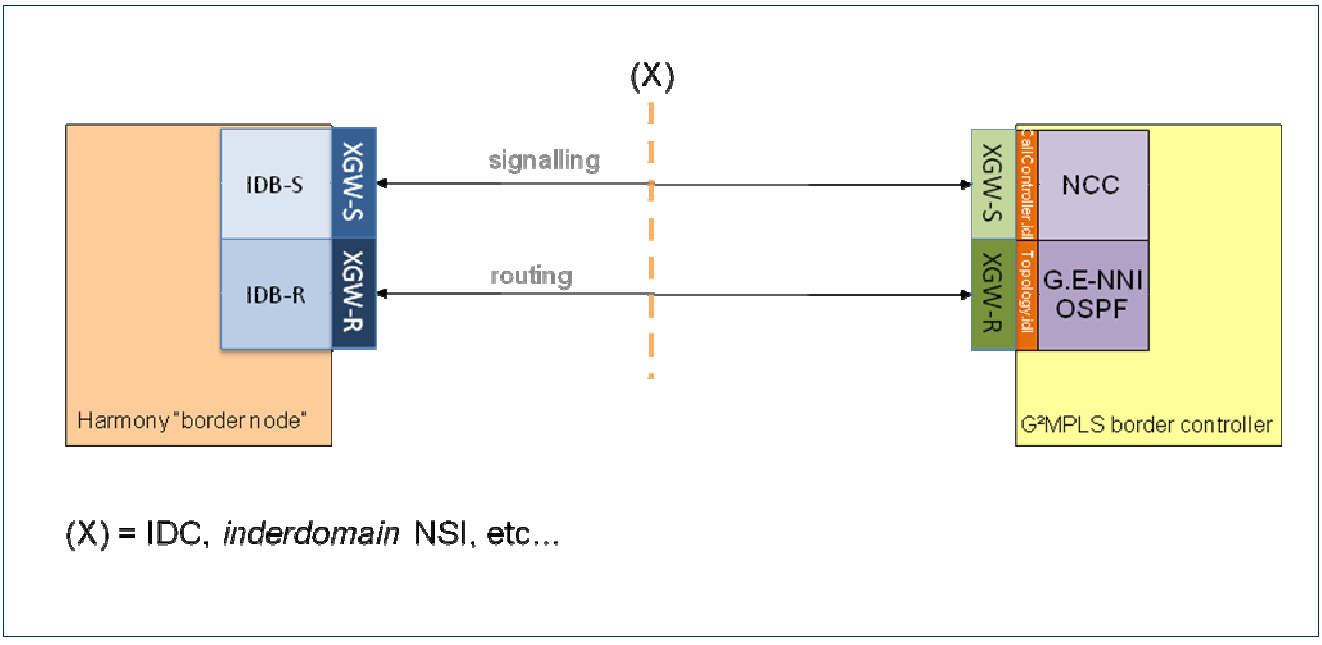

Figure 1-6: GMPLS/G²MPLS interworking with Harmony (east-west case 2)

The second solution implies that Harmony should "learn" E-NNI language (Figure 1-7). To achieve this, a "light" G^{2}MPLS controller is installed within the Harmony IDB using a Corba API that communicates with the G^{2}MPLS controllers. Calls are mapped into the appropriate Corba methods and the corresponding process will generate the E-NNI signalling or routing messages and viceversa.

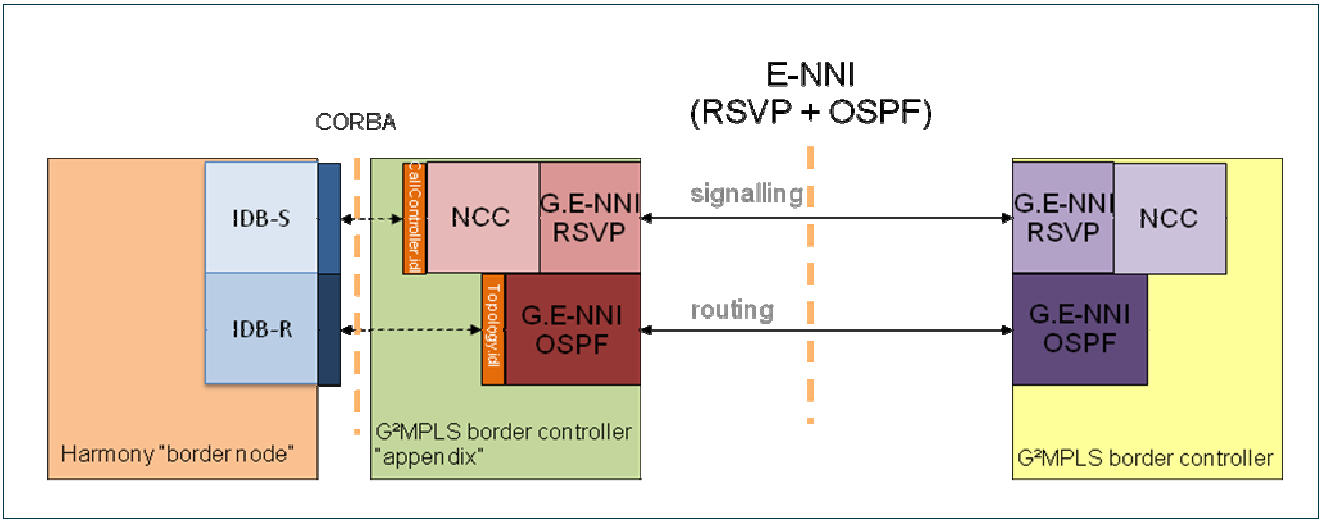

Figure 1-7: GMPLS/G²MPLS interworking with Harmony (east-west case 1)

Finally, the third solution incorporates a gateway that implements a web client/server able to parse and generate Harmony Web Services requests and responses and translate the calls into the corresponding Corba methods that trigger the G^{2}MPLS services (Figure 1-8).

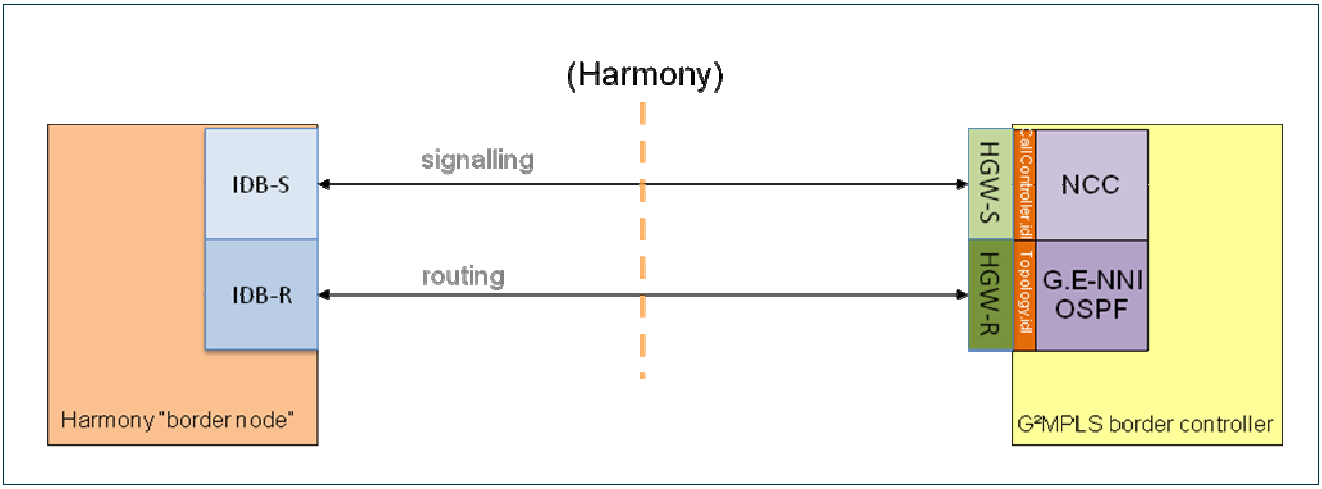

Figure 1-8: GMPLS/G²MPLS interworking with Harmony (east-west case 3)

The implementation of this solution is the one described in the next sections of this deliverable.

2 Signalling and routing specifications

2.1 Signalling specifications

2.1.1 G²MPLS

G²MPLS signalling is based on the Distributed Call and Connection Management (DCM) defined by ASON/GMPLS and extended on purpose with Grid resource specifications. Foundations for this model are:

- the RSVP-TE protocol used at the different network interfaces (G.UNI, G-I.NNI, G.E-NNI);
- the Call and Recovery bundle Controllers, operating on each G²MPLS edge or border node.

Details on the G²MPLS signalling have been widely discussed in [PH-WP2-D2.7], [PH-WP2-D2.2] and [PH-WP2-D2.3], both in terms of message flows and message contents. In this context, the main features are summarized in the perspective of the G²MPLS/Harmony interoperation (ref. Figure 2-1).

Main initiators of the call and connection signalling are either the CCC in case of Switched Connections or the NCC in case of Soft Permanent Connections. NCC can also progress the signalling of an inter-domain call.

Each G²MPLS Call is setup with 3-tier message flows among the involved entities:

- at G.OUNI, the CallSetupRequest / CallSetupIndication / CallSetupConfirm is mapped into G.UNI Path / Resv / ResvConf
- at G.E-NNI Path / Resv / ResvConf
- between Network Call Controllers, the ISI CallSetupRequest / CallSetupIndication / CallSetupConfirm

G²MPLS signalling does not use RFC2205 RSVP soft state in order to limit the Signalling Communication Network (SCN) load. G²MPLS signalling uses Explicit Routing, also in the inter-domain call/connection establishment. The trigger for G²MPLS signalling is always generated inside a xCC module, which can be stimulated through a Corba interface deeply described in [PH-WP2-D2.3]. Therefore, the interoperation between G²MPLS and any other technologies for reservation of Network Services does not happen at the RSVP protocol level, but occurs at the higher Corba level, by stimulating directly the NCC. Relevant methods that need to be used for the interoperation with Harmony are discussed in Section 3.1.1.

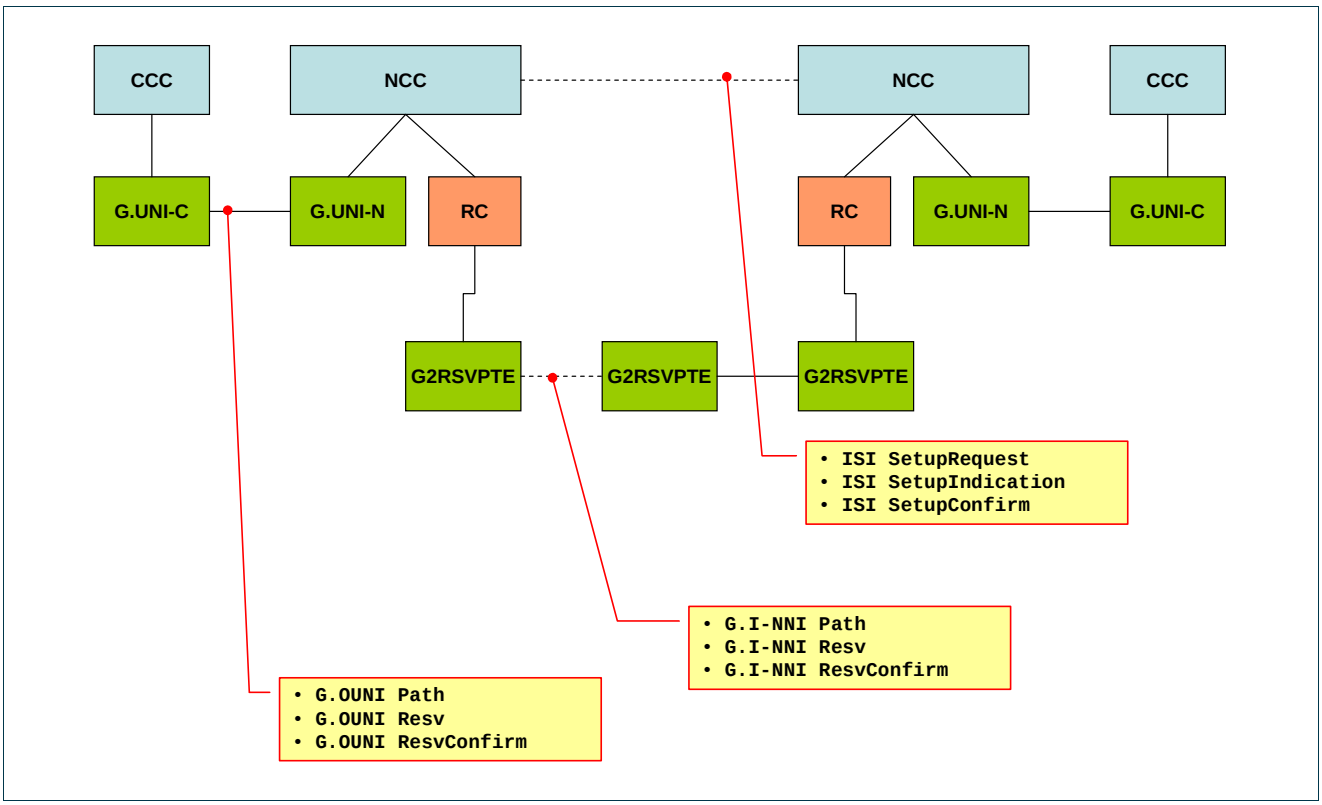

Figure 2-1: G²MPLS signalling entities and messages.

2.1.2 Harmony

The Reservation interface is the component of the Harmony Service Interface which enables the signalling within the Harmony system. Reservation interface is web service based and allows the user or the middleware to create, cancel and query advanced resource reservations and check for the availability of these network resources.

This reservation interface has been widely described in [PH-WP1-D1.4]. Reservation-WS has two main components:

- Reservation wsdl (Web Service Description Language). This component describes all the methods that can be invoked over the interface. The main requests are:
  - Reservation request: reserves network resources for one or more connections.
  - Availability request: checks the availability of the resources required for connections with the endpoints in the same domain or in different domains.
  - Reservation status: queries one ore more reservations previously made.
  - Cancel reservation: deletes a reservation releasing all the resources and tagging them as available.
- Reservation types (XML Schema Definition). This component describes all the specific data types used in the signalling part of Harmony.

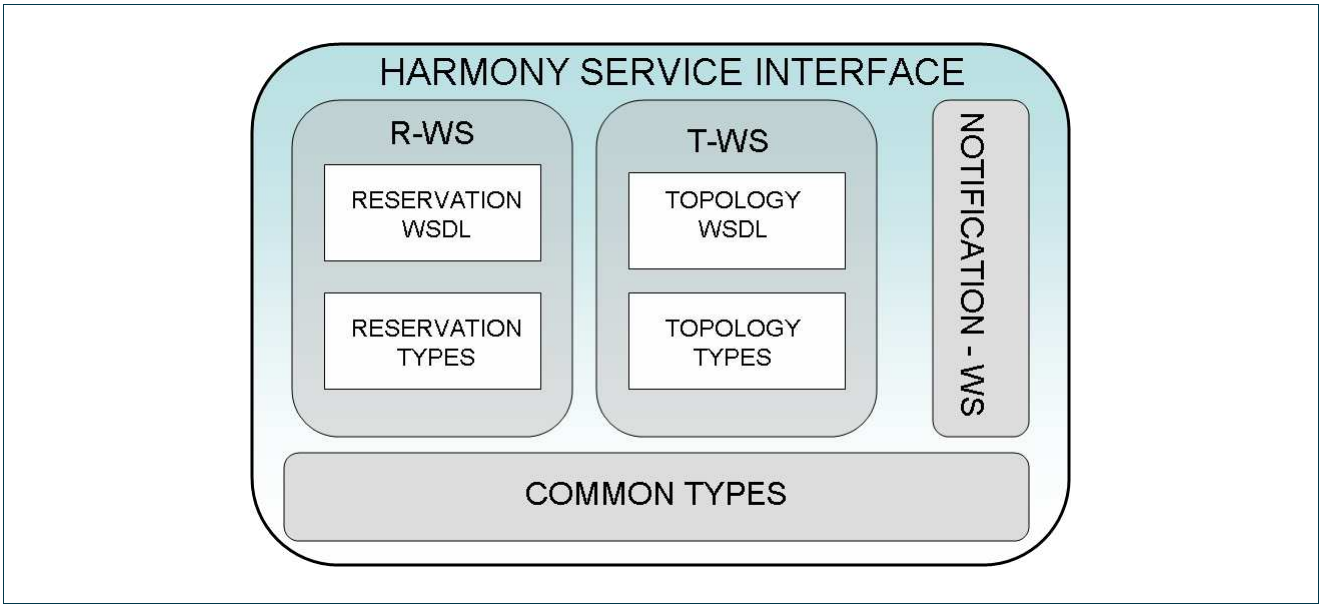

Figure 2-2: Harmony Service Interface structure

Moreover, it is useful to remark that this WS uses the common data types defined in the Common types file.

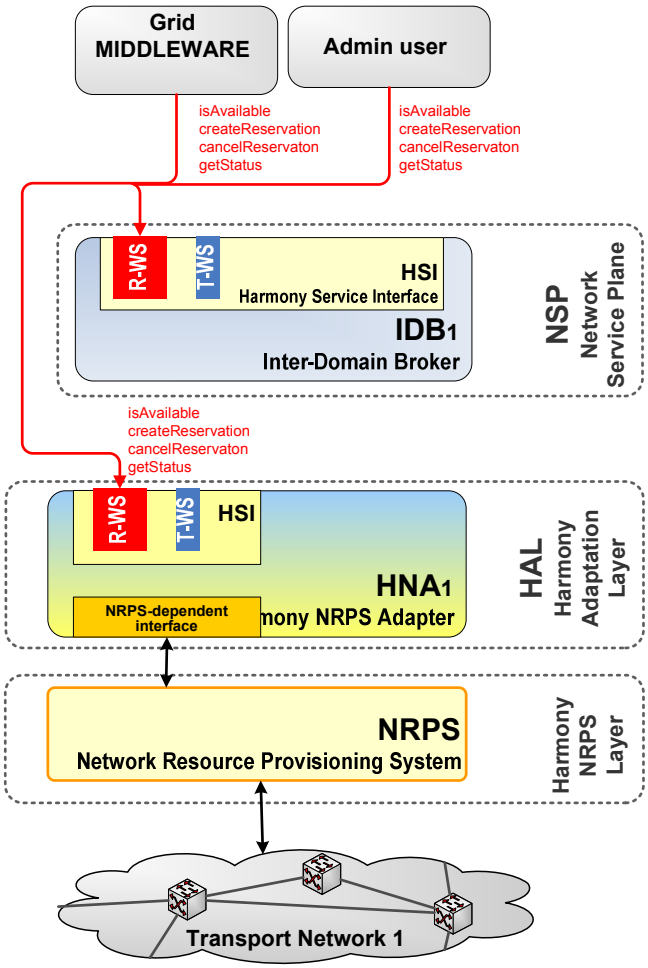

Figure 2-3: Signalling inside Harmony

2.1.3 Compatibility

The main signalling compatibility problem lies on mapping blocking method calls from Harmony to non-blocking methods from G^{2}MPLS. In the Harmony side, when the Inter-domain broker receives a create reservation request, the module in charge of forwarding the requests to the NRPS waits until the response is received. On the other hand, G^{2}MPLS uses non-blocking methods for creating a network resource reservation by acknowledging each request. This problem is solved in the gateway by implementing a blocking call that waits for an east-west message event coming from G^{2}MPLS indicating success or failure in the establishment of the connection. If this event is not received, a timeout also indicates failure in the procedure.

2.2 Routing specifications

2.2.1 G²MPLS

G²MPLS routing E-NNI interface is used to send information between peering domains about Routing Controllers (which represents a single domain), inter-domain TE-links and TNAs. These three kinds of information are flooded between peering Routing Controllers. Routing Controller is receiving via G²MPLS routing E-NNI interface not only information about peer G²MPLS domain but also information related to other G²MPLS domains achievable only via this peer domain. Thus, every Routing Controller contains a database with overall multi-domain G²MPLS topology. G²MPLS Routing Controller is responsible for the topology summarization/abstraction process. However, regarding network information, only TNAs assigned to domain are summarized and originated via G.E-NNI interface as single TNA with prefix length. G²MPLS Routing is based on the OSPF-TE protocol. In this protocol all topology information is divided into standalone pieces. Any piece of information is flooded using an OSPF opaque LSA. Any change of information triggers an OSPF opaque LSA update. There are two main time-constrains in each Routing Controller:

- every information, if no change happens, is re-originated every half an hour,
- when no update is received in period of one hour the information is deleted.

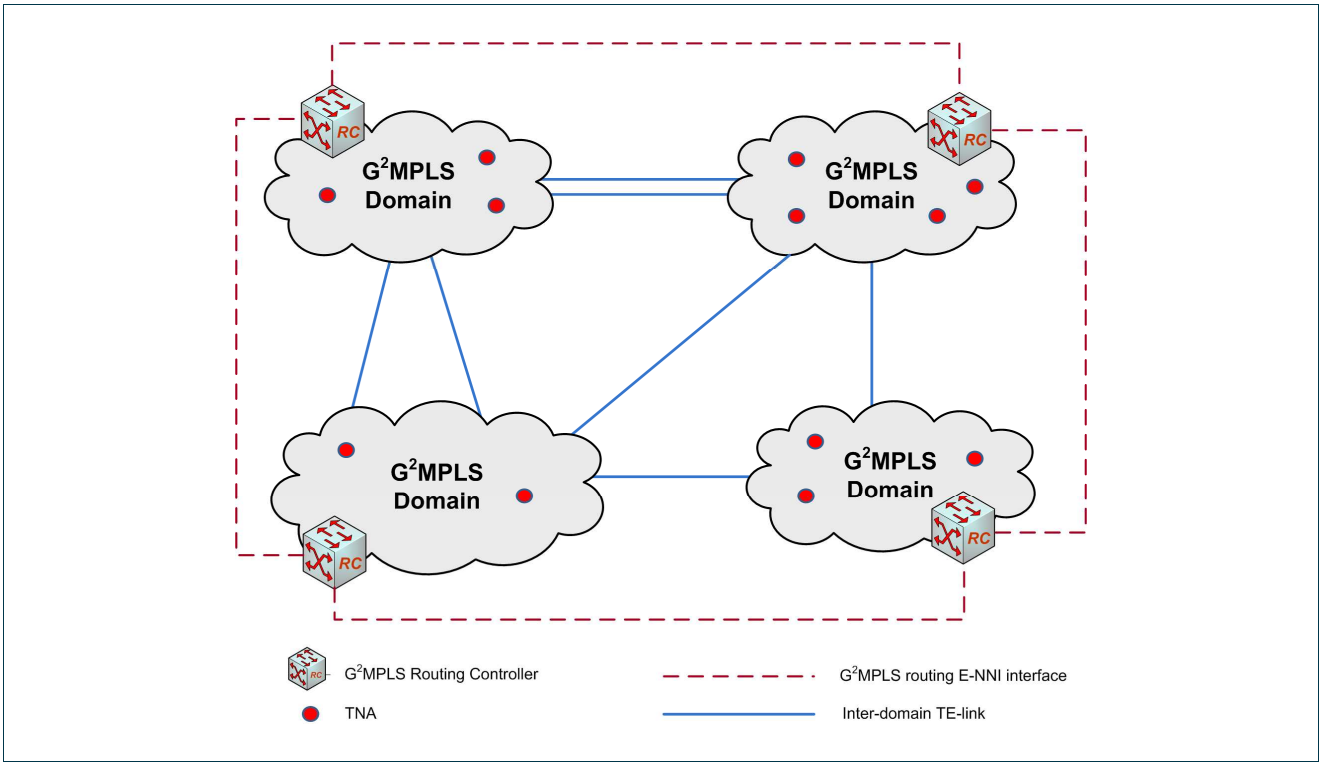

Figure 2-4: G²MPLS inter-domain routing

G^{2}MPLS Routing Controller functionality is implemented in G^{2}MPLS Controller as G.OSPF-ENNI module. This module contains a Corba client and servant allowing external access to all inter-domain routing information gathered by G^{2}MPLS Routing Controller. Using this Corba client/servant it is possible to trigger similar actions which happen on G^{2}MPLS E-NNI interface. Table 2.1 presents available Corba methods for each kind of structure. This table presents that G^{2}MPLS Routing Controller offers elementary actions; however, only part of them would probably be useful in Harmony and G^{2}MPLS interworking.

Information about	Corba function	Related structures	Description
Node	nodeAdd	nodeIdent	Add a node to database
	nodeDel	nodeIdent	Delete a node from database
	nodeGetAll	nodeIdentSeq	Get all nodes from database
	netNodeUpdate	nodeIdent, netNodeParams	Update more details in already present node in database
	netNodeGet	nodeIdent, netNodeParams	Get more details about a node from database
TNA	tnaIdAdd	tnaIdent	Add a TNA to database
	tnaIdDel	tnaIdent	Delete a TNA from database
	tnaIdsGetAllFromNode	nodeIdent, tnaIdentSeq	Get all TNAs associated to Routing Controller (TNA is present in the domain controlled by Routing Controller) from database
Inter-domain TE-link	linkAdd	teLinkIdent	Add a TE-link to database
	linkDel	teLinkIdent	Delete a TE-link from database
	teLinkGetAllFromNode	teLinkIdent, teLinkIdentSeq	Get all TE-links associated to Routing Controller (one end of TE-link is present in the domain controlled by Routing Controller) from database
	teLinkUpdateCom	teLinkIdent, teLinkComParams	Update more details in already present TE-link in database
	teLinkGetCom	teLinkIdent, teLinkComParams	Get more details about a TE-link from database
	teLinkUpdateTdm	*teLinkIdent, teLinkTdmParams*	Update more details in already present TDM TE-link in database
	teLinkGetTdm	*teLinkIdent, teLinkTdmParams*	Get more details about a TDM TE-link from database
	teLinkUpdateLscG709	*teLinkIdent, teLinkLscG709Params*	Update more details in already present G.709 TE-link in database
	teLinkGetLscG709	*teLinkIdent, teLinkLscG709Params*	Get more details about a G.709 TE-link from database
	teLinkUpdateLscWdm	*teLinkIdent, teLinkLscWdmParams*	Update more details in already present WDM TE-link in database

	teLinkGetLscWdm	*teLinkIdent, teLinkLscWdmParams*	Get more details about a WDM TE-link from database
	teLinkUpdateStates	teLinkIdent, statesBundle	Update TE-link administrative and operation states in database
	teLinkGetStates	teLinkIdent, statesBundle	Get TE-link administrative and operation states from database
	teLinkUpdateGenBw	*teLinkIdent, availBwPerPrio*	Update available TE-link bandwidth per each priority in database
	teLinkGetGenBw	*teLinkIdent, availBwPerPrio*	Get available TE-link bandwidth per each priority from database
	teLinkUpdateTdmBw	*teLinkIdent, freeCTPSeq*	Update TDM TE-link free connection termination points in database
	teLinkGetTdmBw	*teLinkIdent, freeCTPSeq*	Get TDM TE-link free connection termination points from database
	teLinkUpdateLscG709Bw	*teLinkIdent, freeCTPSeq*	Update G.709 TE-link free connection termination points in database
	teLinkGetLscG709Bw	*teLinkIdent, freeCTPSeq*	Get G.709 TE-link free connection termination points from database
	teLinkUpdateLscWdmBw	*teLinkIdent, teLinkWdmLambdasBitmap*	Update available TE-link lambdas bitmap in database
	teLinkGetLscWdmBw	*teLinkIdent, teLinkWdmLambdasBitmap*	Get available TE-link lambdas bitmap from database
	teLinkAppendSrlgs	*teLinkIdent, srlgSeq*	Update TE-link shared risk link group membership in database
	teLinkGetSrlgs	*teLinkIdent, srlgSeq*	Get TE-link shared risk link group membership from database
	teLinkAppendCalendar	teLinkIdent, teLinkCalendarSeq	Append TE-link availability calendar in database
	teLinkGetCalendar	teLinkIdent, teLinkCalendarSeq	Get TE-link availability calendar from database
	teLinkAppendIsc	*teLinkIdent, iscSeq*	Get TE-link interface switching capability descriptor from database
	teLinkGetIsc	*teLinkIdent, iscSeq*	Update TE-link interface switching capability descriptor in database

Table 2.1: Available methods for G^{2}MPLS E-NNI topology and more detailed TE information exchange.

Most important structures which are used by G^{2}MPLS E-NNI Corba methods are presented in Table 2.2. Main structures (Figure 2-5) for building network topology are:

- nodeIdent – in case of G.E-NNI it is Routing Controller ID which represent a domain,
- tnaIdent – composed of three identifiers: TNA ID with address prefix length, Network node ID where TNA is connected and Routing Controller ID which is responsible for that domain,

- teLinkIdent – composed of six identifiers: local and remote TE-link endpoints IDs, local and remote controllers IDs where TE-links is installed, local and remote Routing Controller IDs which are responsible for local and remote domain.

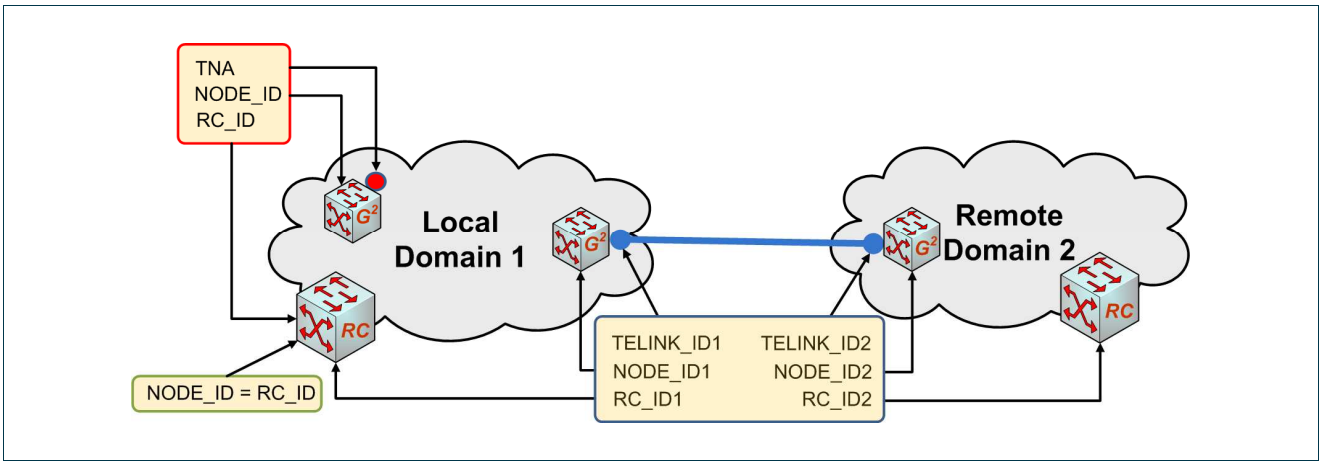

Figure 2-5: Identifiers of G²MPLS inter-domain routing objects

The other structures contain additional and more detailed information about main topology structures.

Structure Name	Attributes	Attribute type	Description
nodeIdent	nodeId	uint32	Identifier of the node which in case of G.E-NNI is Routing Controller (represent a domain)
netNodeParams	isDomain	boolean	True in case of Routing Controller
	state	statesBundle	Operational and Administrative node status
	colors	uint32	All Te-link colours associated to node
	areas	areaSeq	Routing Areas membership
tnaIdent	rc	uint32	Routing Controller associated with the TNA
	node	uint32	Network node associated with the TNA
	tna	ipv4 or ipv6 or NSAP	TNA identifier
	prefix	uint8	TNA address prefix

			(number of bits)
teLinkIdent	localNodeId	uint32	Local network node identifier
	localId	ipv4 or ipv6 or uint32	Local TE-link endpoint identifier
	remoteNodeId	uint32	Remote network node identifier
	remoteId	ipv4 or ipv6 or uint32	Remote TE-link endpoint identifier
	localRcId	uint32	Local Router Controller identifier
	remoteRcId	uint32	Remote Router Controller identifier
teLinkComParams	adminMetric	uint32	Administrative TE-link metric
	teMetric	uint32	TE-link metric
	teColorMask	uint32	administrative group membership for this TE-link
	teProtectionTypeMask	uint8	TE-link protection capabilities
	teMaxBw	uint32	TE-link maximum bandwidth that can be used on this TE-link
	teMaxResvBw	uint32	TE-link maximum bandwidth that may be reserved on this TE-link
statesBundle	operState	OPERSTATE_UP OPERSTATE_DOWN	Operational status
	adminState	ADMINSTATE_DISABLED ADMINSTATE_ENABLED	Administrative status

Table 2.2: G^{2}MPLS E-NNI topology database structures

2.2.2 Harmony

Table 2-3 shows the DomainInformationType used to advertise domain information, and Table 2-4 shows the InterdomainlinkType that is used to add information about a domain's interdomain links in a domain. In the distributed mode of operation, topology information is disseminated solely by advertising such DomainInformationType in regular intervals by calling the addOrEditDomain operation the destination domain's Topology-WS. In the hierarchical mode, the advertisements are sent to the respective parent domains; in the peer-to-peer mode, they are flooded to all peer domains.

Inside the DomainInformationType, only the DomainId and ReservationEPR fields are mandatory for several reasons. However, in the distributed mode of operation, where only the addOrEditDomain operation is used to advertise domain information stored in a DomainInformationType, the following fields should be set additionally:

- The *Relationship* field (which defaults to "subdomain" if it is not set for backward compatibility reasons).
- The *SequenceNumber* field is important in peer-to-peer scenarios to allow old information to be discarded.
- The *InterdomainLink* field contains all information necessary for receivers to lookup the required topology information if the advertisements of both domains connected by an interdomain link are known. Note that it is not necessary for a domain to know the TNA associated with an interdomain link within its peer domain.

Name	Type	optional / unbounded	description
DomainId	string	N/N	Unique identifier for the domain
Relationship	string	Y/N	Enumeration. Can be one of the self-explanatory values "subdomain" or "peer".
SequenceNumber	int	Y/N	Increased by the origin domain, allows others to check if information has changed and is more current than previously stored information.
Description	string	Y/N	Short description of the domain.
ReservationEPR	anyURI	N/N	Endpoint reference of the domain's Reservation-WS.
TopologyEPR	anyURI	Y/N	Endpoint reference of the domain's Topology-WS.
NotificationEPR	anyURI	Y/N	Endpoint reference of the domain's Notification-WS.
TNAPrefix	string	Y/Y	List of TNA prefixes the domain is responsible for.
InterdomainLink	Interdomain-LinkType	Y/Y	List of interdomain links originating from the domain.
avgDelay	int	Y/N	Average delay for paths in this domain; path computer optimization.
maxBW	int	Y/N	Maximum bandwidth for paths in this domain; path computer optimization.

Feature	string	Y/Y	List of features supported by this domain. Currently unused.

Table 2-3: Fields in the DomainInformationType.

Name	Type	optional / unbounded	description
LinkId	string	N/N	ID of the link, must be unique only within same pair of domains.
SourceEndpoint	string	N/N	TNA of the source endpoint in the domain advertising this interdomain link.
DestinationDomain	string	N/N	Identifier of the destination domain this link connects to.

Table 2-4: Fields in the InterdomainLinkType.

2.2.3 Compatibility

In a hierarchical model HG^2 GW topology service advertises only topology associated with the local G^{2}MPLS domain. Harmony path computation behaves in a similar way as G^{2}MPLS PCE. Topology information is not pushed from Harmony to G^{2}MPLS since G^{2}MPLS PCE only calculates paths inside the G^{2}MPLS domain.

When a peer model is used, some problems in topology information exchanging have to be addressed. Every G^{2}MPLS domain advertises the overall inter-domain topology known. Thus, if Harmony domain is connected to two or more G^{2}MPLS domains it will receive more copies of the inter-domain topology of G^{2}MPLS domains. The second problem is related to Harmony topology information exchange in peer model where each Harmony domain sends its own topology among them. When more than one G^{2}MPLS domain will receive the same topology of the same Harmony domain, then both domains will flood this topology information to other G^{2}MPLS domains.

To address the first problem, some solutions have been identified:

- Each G^{2}MPLS domain runs its own HG2 topology gateway which connects to each IDL-TOPOLOGY-WS which should advertise only topology associated with local G^{2}MPLS domain. This solution (Figure 2-6) gives a better flexibility in connecting Harmony and G^{2}MPLS domains; if any G^{2}MPLS domain does not run HG^2 GW then it can't be seen by the Harmony domains.

- HG^2 GW is present only in G^{2}MPLS domains peering to Harmony domains (Figure 2-7). HG^2 GW advertises only inter-domain links between Harmony and G^{2}MPLS domains and all TNAs that G^{2}MPLS domain knows (as its own TNAs). HG2 topology gateway simplifies all G^{2}MPLS domains to one G^{2}MPLS domain and each HG2 topology gateway exchanges topology with every Harmony gateway. If there are more HG^2 GW then the same or similar topology view can exist in different Harmony

domains. A drawback is that Harmony Path computation may not choose the best G^{2}MPLS border node because of the lack of distance information regarding the destination TNA. In this case, PCE computes the path to destination TNA via intermediate G^{2}MPLS domains.

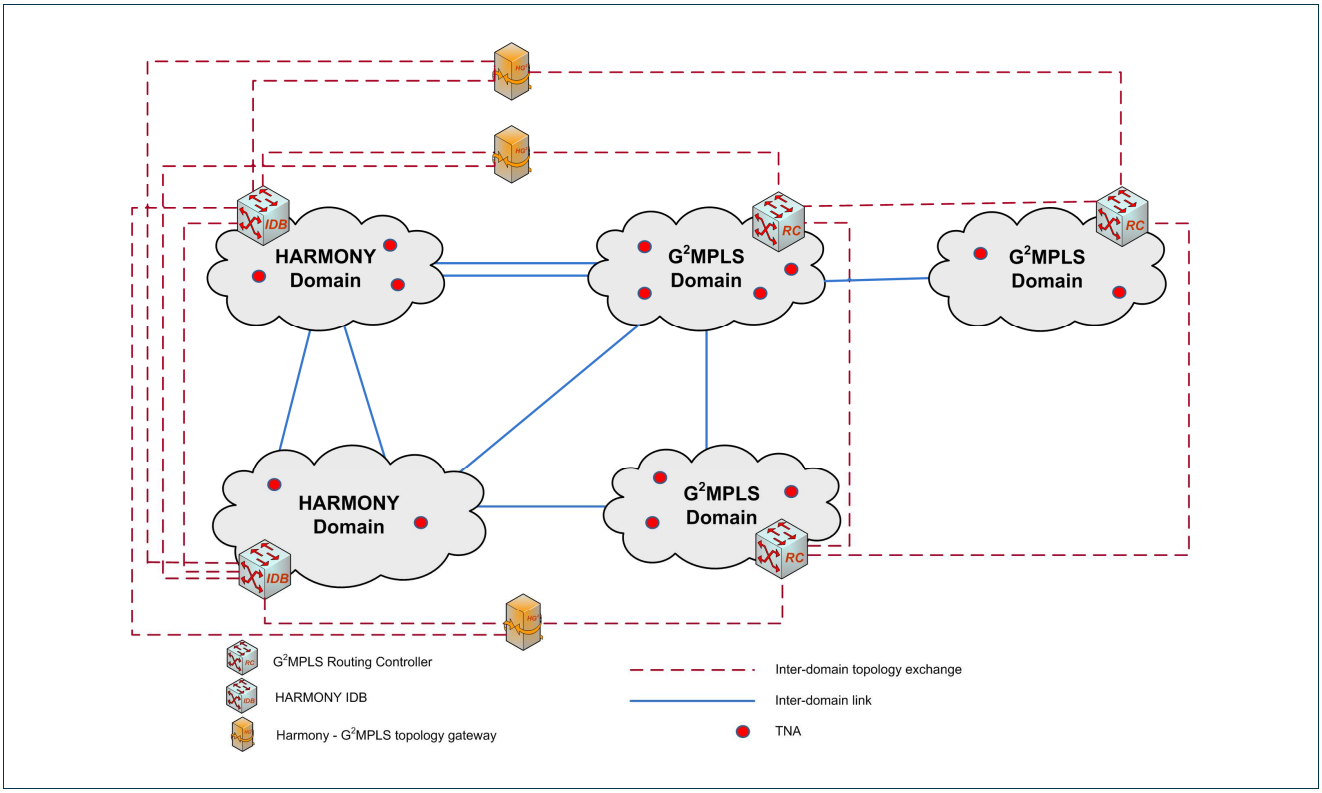

Figure 2-6: Routing interoperabilty - solution one - eachl domain peers only its own topology with each other

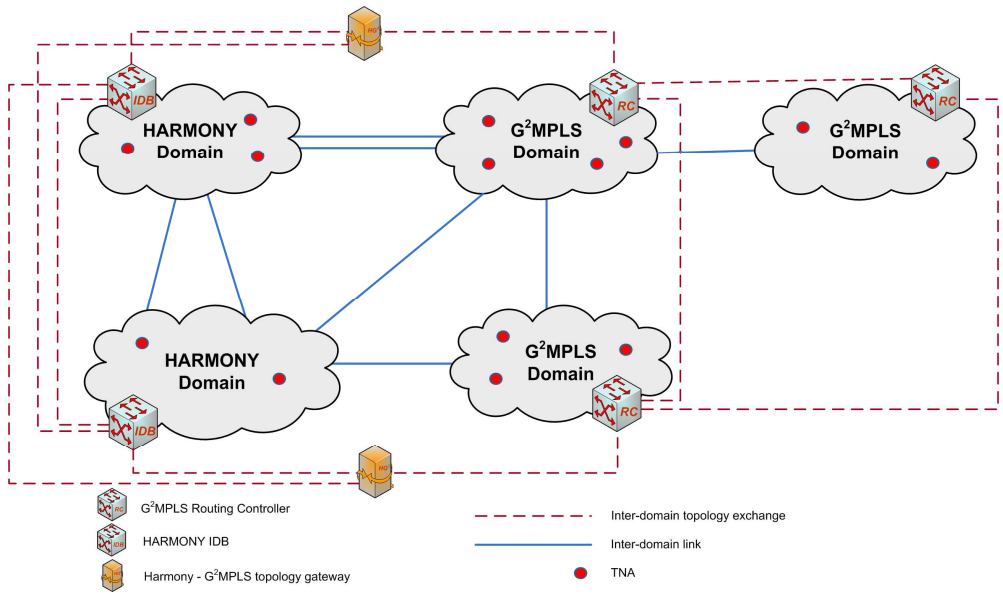

Figure 2-7: Routing interoperabilty - solution two - only border G2MPLS domains peer topology with each HARMONY domain.

- The third solution (Figure 2-8) is the simplest one since just one HG^2 GW is needed. The gateway can retrieve full G^{2}MPLS inter-domain topology from any G^{2}MPLS Routing Controller. It also pushes Harmony domains topology to one G^{2}MPLS Routing Controller. On the Harmony side HG^2 GW connect to each Harmony domain and push topology related to each G^{2}MPLS domain.

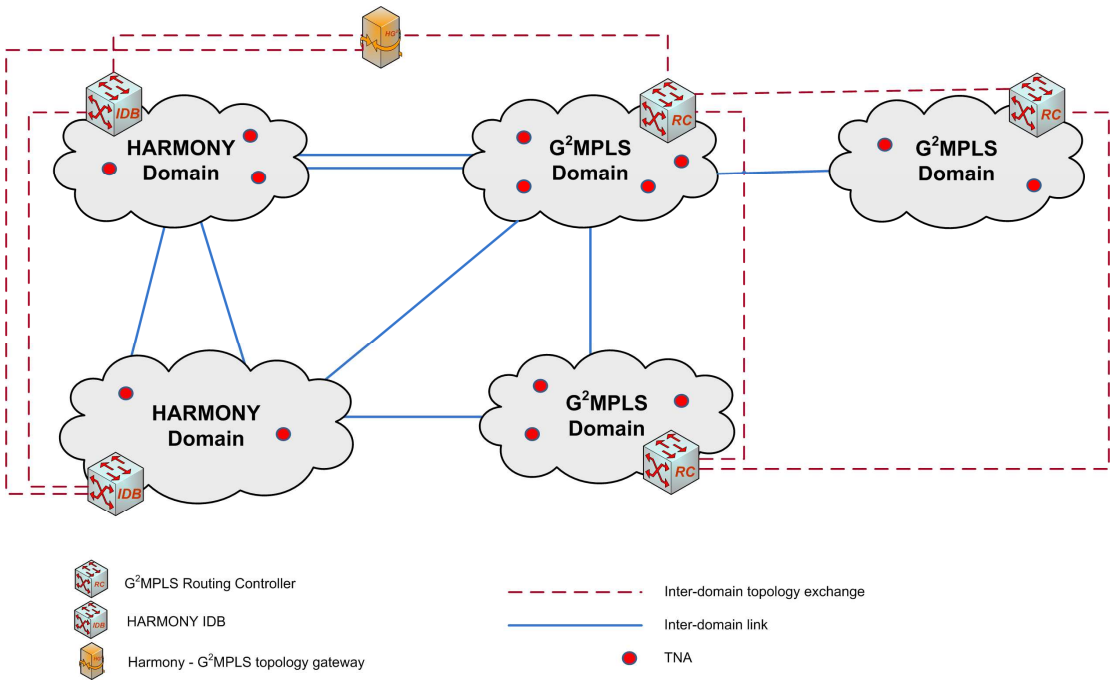

Figure 2-8: Routing interoperabilty - solution third - only one topology exchange gateway between HARMONY domains and G2MPLS domains

There are also few possibilities for solving of the second problem with multiple Harmony domains topology flooding between G^{2}MPLS domains:

- Statically choosing one designed HG^2 GW which will push topology information about all Harmony domains to one G^{2}MPLS domain. G^{2}MPLS domain will flood these topologies to all connected G^{2}MPLS domains. There rest of HG^2 GW will only transfer topology information on one direction: from G^{2}MPLS domain to Harmony domain.

- The second solution is introducing a special flag in G^{2}MPLS OSPF opaque LSAs which will cause that received Harmony topology will not be flooded to other G^{2}MPLS domains. In that case every G^{2}MPLS domain can import any Harmony topology without consequences to other G^{2}MPLS domains.

2.3 HG² GW Message flow charts

2.3.1 Signalling

2.3.1.1 *Overview of the Message Workflow*

In order to provision and control a path in both systems, mainly two aspects have to be considered:

1. Mapping of request and response data structures (cf. Section 1.4.1).
2. Mapping of synchronous and asynchronous method calls.

In a standard workflow is shown that is needed to administrate a reservation from the Harmony point of view. Starting point is a client library that is called by a user or another system.

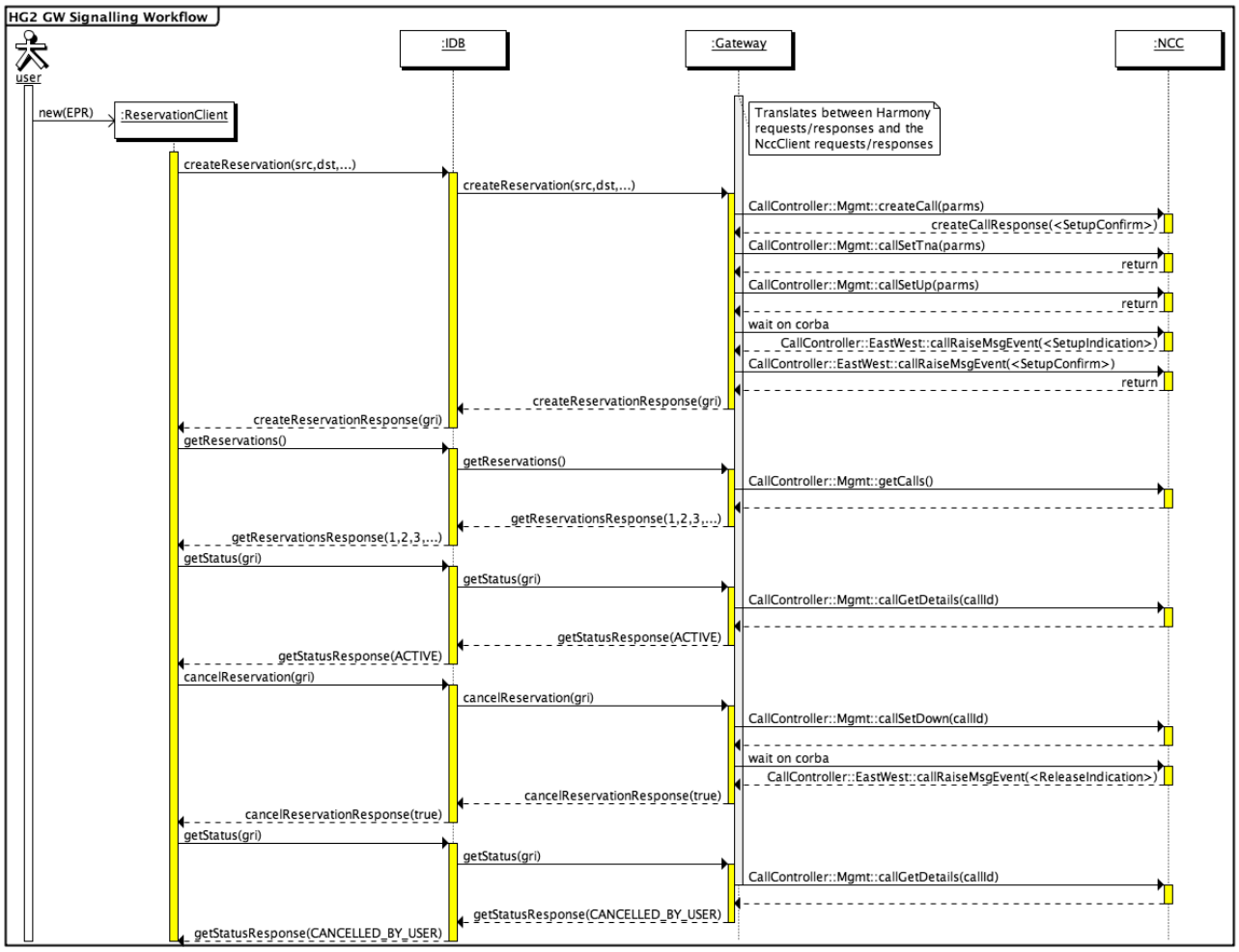

Figure 2-9: HG² GW Signalling Workflow – from Harmony to G²MPLS.

The next five synchronous method calls going from the Harmony system to a gateway can be described as follows:

1. createReservation: After this reservation request a path should be provisioned within the G²MPLS domain and a reservation identifier is expected to be returned. In order to setup a path in G²MPLS five asynchronous messages are needed to be exchanged with the Network Call Controller (NCC).

2. getReservations: Often, a GUI is controlling reservations on behalf of a user. In order to show existing reservations, they needed to be queried first. This request can be mapped to a single NCC request and returns a list of available reservations made in the G²MPLS domain.

3. getStatus: After a reservation was made, the current status of the provisioned path could be queried to ensure that it is in an active (immediate reservation) or pending state (advance reservation). Again this request can be mapped to a single NCC request.

4. cancelReservation: In case that a user wants to cancel a requested path earlier than its finish time, this request must be sent. Two asynchronous method calls are used to communicate with the NCC and in general, a success message will be returned.

5. getStatus: Finally, in order to verify whether the path was tore down, the getStatus request is sent again. It should return a cancelled_by_user or tear_down_in_progress status at this point of time.

2.3.2 Routing

In case of getting topology information from HG^2 GW topology gateway some of G.OSPF-ENNI Corba methods have to be called. Using Routing Controller nodeIdent and extracting the HG2 topology gateway config file, it is possible to retrieve Routing Controller information, all TNA addresses present in G^{2}MPLS domain and all local inter-domain TE-links identifiers.

Using inter-domain TE-links identifiers, it is then possible to retrieve all inter-domain TE-links parameters.

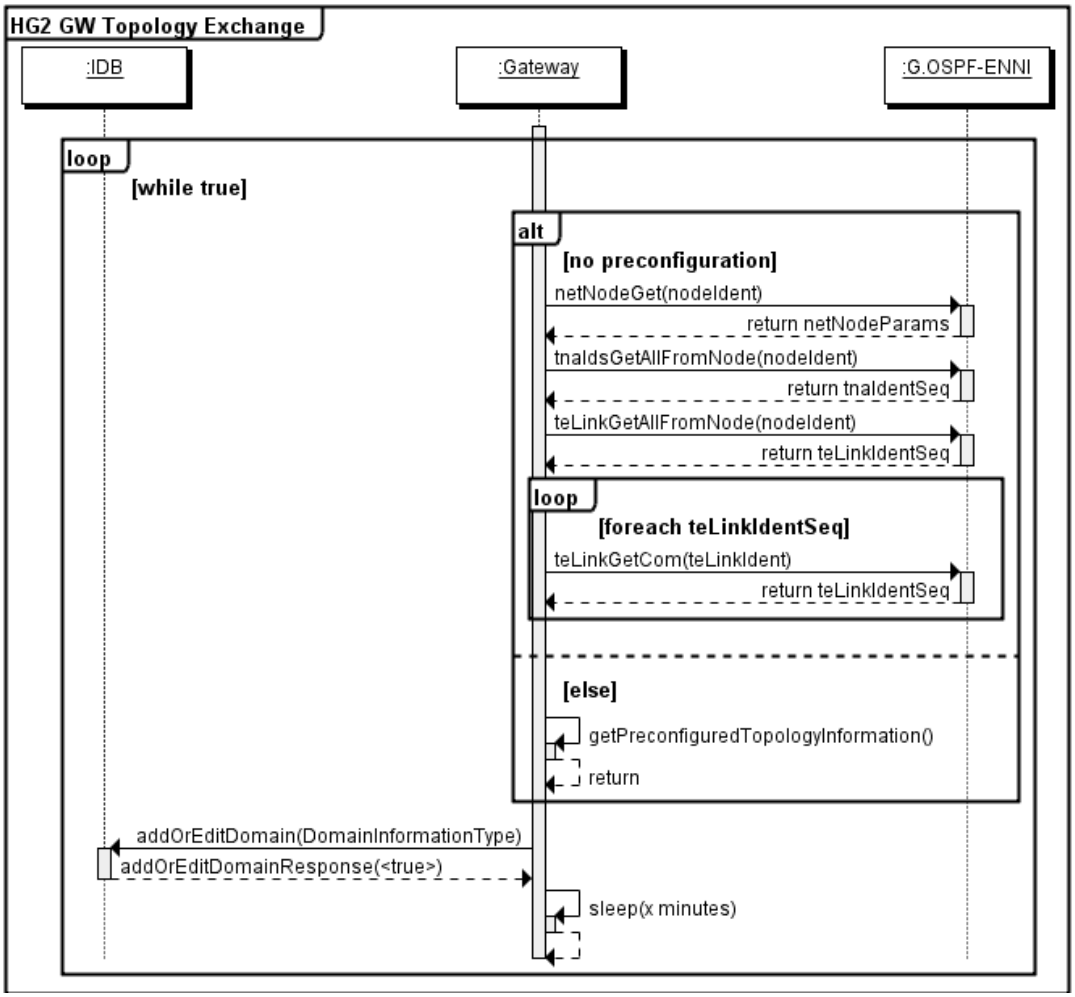

Figure 2-10: Sequence Diagram of the HG^2 GW Topology Exchange – G^{2}MPLS to HARMONY

Pushing topology through the HG^2 GW gateway to a G^{2}MPLS domain requires calling various G.OSPF-ENNI Corba methods. It is important to dynamically add and delete TNAs and inter-domain TE-links, and also to avoid unnecessary information updating which could trigger additional G^{2}MPLS OSPF opaque LSA flooding. For this reason, firstly the TNA_prefix or inter-domain links are checked by enquiring G^{2}MPLS Routing Controller about topology. Then the list of TNAs and links gathered from Harmony IDB and G^{2}MPLS Routing Controller are compared. The same procedure is applied for checking inter-domain link parameters changes.

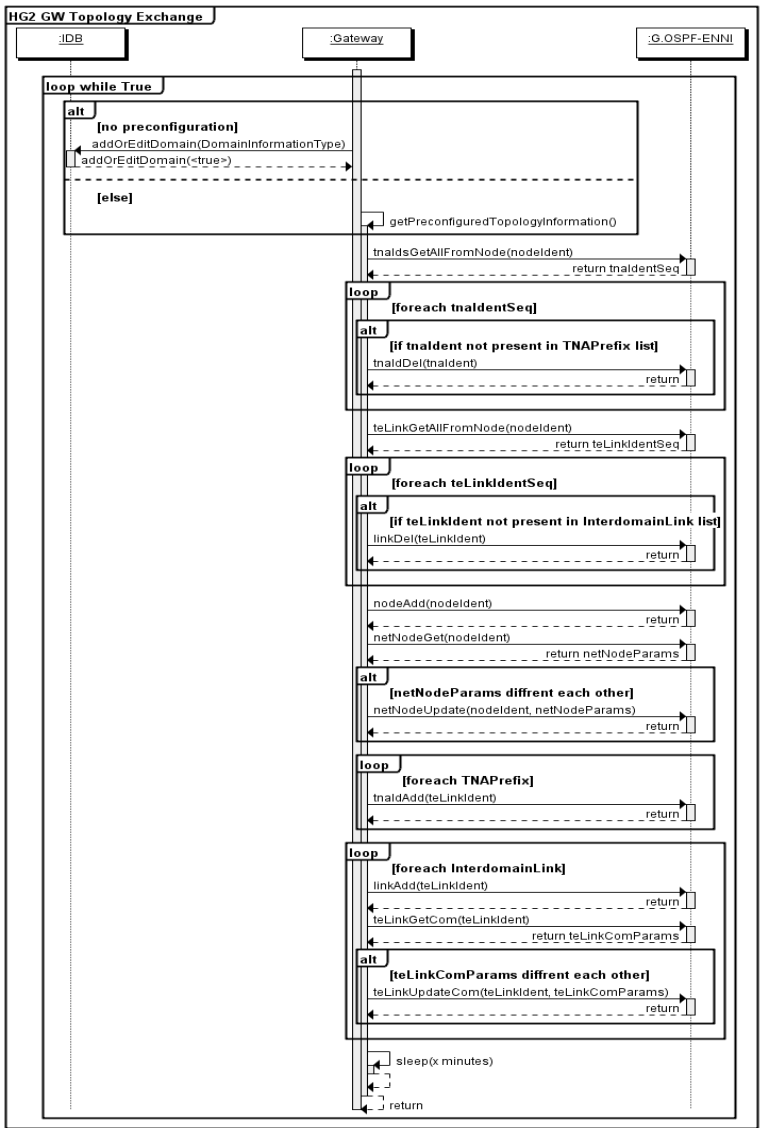

Figure 2-11: Sequence Diagram of the HG2 GW Topology Exchange – HARMONY to G^{2}MPLS

3 HG² GW Implementation

3.1 HG² GW Web Service

The signalling interface of the HG2 GW with the Harmony network service plane is established with one inter-domain broker. The gateway consists of a signalling component, which implements the methods defined in web service description language within the interface. The following tables show a detailed information about these functions.

3.1.1 Reservation Management

WSDL Operation Name	getReservation
Description	Retrieves the input by which a reservation request was made
Input parameters (XML type)	ReservationIdentifierType : String ServiceIdentifierType : integer
Output parameters (XML type)	GetReservationResponse : GetReservationResponseType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	getReservations
Description	Retrieves all existing reservations for the specified period of time
Input parameters (XML type)	PeriodStartTime : dateTime PeriodEndTime : dateTime
Output parameters (XML type)	*Sequence of* Reservation : GetReservationsComplexType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault

Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	cancelReservation
Description	Cancels a network resource reservation
Input parameters (XML type)	ReservationIdentifierType : String
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	getStatus
Description	Returns the status of a service
Input parameters (XML type)	Service : ServiceConstraintType JobIdentifierType : long
Output parameters (XML type)	ServiceStatus : ServiceStatusType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

3.1.2 Reservation Setup

WSDL Operation Name	isAvailable
Description	Checks whether the specified service is available
Input parameters (XML type)	Service: ServiceConstraintType JobIdentifierType : long
Output parameters (XML type)	DetailedResult : ConnectionAvailabilityType *optional* AlternativeStartTimeOffset : long
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault
Fault (XML type)	EndpointNotFoundFault : EndpointNotFoundFault

WSDL Operation Name	createReservation
Description	Creates the reservation of a path between 2 endpoints considering the specified constraints
Input parameters (XML type)	Service : ServiceConstraintType JobIdentifierType : long NotificationConsumerURL : string
Output parameters (XML type)	JobIdentifierType : long ReservationIdentifierType : long Detailed result : ConnectionAvailabilityType
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault
Fault (XML type)	EndpointNotFoundFault : EndpointNotFoundFault

3.1.3 Connection Management

WSDL Operation Name	activate
Description	Activates a service
Input parameters (XML type)	ReservationIdentifierType : long ServiceIdentifierType : integer
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

WSDL Operation Name	bind
Description	Create binding between NRPS endpoint and application endpoint
Input parameters (XML type)	ReservationIdentifierType : long ServiceIdentifierType : integer ConnectionIdentifierType : integer EndpointID : EndpointIdentifierType
Output parameters (XML type)	Success : boolean
Fault (XML type)	UnexpectedFault : UnexpectedFault
Fault (XML type)	InvalidRequestFault : InvalidRequestFault
Fault (XML type)	OperationNotAllowedFault : OperationNotAllowedFault

3.1.4 From Harmony to G^{2}MPLS

Signalling messages between Harmony and G^{2}MPLS can be sent on both directions. When messages are sent from Harmony to G^{2}MPLS the signalling component of the translator in the Harmony side acts as a server/consumer. The gateway receives the incoming requests from the adjacent inter-domain broker and forwards it to the translator module of the gateway, which then maps the requests to G^{2}MPLS calls.

3.1.5 From G^{2}MPLS to Harmony

In the opposite direction (G^{2}MPLS to Harmony), the signalling module acts as a client/producer. The translator component of the gateway receives the requests from G^{2}MPLS and translates them. After this mapping, it forwards the new requests to the reservation WS proxy. The WS proxy then forwards the requests to the necessary network service plane entities located in the Harmony side.

3.2 HG² GW Corba Client/Servant

The signalling interface of the HG² GW with the G²MPLS Control Plane is established with the Network Call Controller (NCC) on a Border G²MPLS node. The interface offered by the NCC is flexible enough to support the plugging of a generic G.E-NNI GW, i.e. it can enable the G²MPLS to be interfaced to virtually any other Control Plane or Network Resource Provisioning Plane architecture, in a peering fashion.

The interaction with the NCC is mediated, in both directions, by a couple of interfaces in the CallController Corba IDL (*CallController.idl*):

- The *CallController::Mgmt* interface
- The *CallController::EastWest* interface

The parameters used in the methods of these interfaces are specified by the *G²mplsTypes* module (*G²mplsTypes.idl*).

The following discussion is strictly focused on the interfacing of an external gateway to these NCC Corba interfaces. A more comprehensive discussion and details are reported in D2.3.

These two interfaces allow an external gateway to plug into the G² Call signalling (e.g. to originate or receive it) with different levels of complexity. In the simplest usage scenario, a GW can issue a Call setup and just poll to check for Call completion. In a more complex and realistic scenario (i.e. that used for interworking with Harmony), the GW will plug into each of the 3 Call signalling tiers, and will be able to know and control the progress of the Call setup / teardown.

The *Mgmt* interface is only served by the NCC, i.e. the gateway is just a client of it. On the contrary, the *EastWest* interface is served and accessed by both the NCC and the gateway, in a bidirectional fashion.

The methods of the two interfaces that are used for gateway purposes are shown in Code 3-1 and Code 3-2, respectively, and explained in the following subsections.

The usage of the two interfaces is intended to be as much symmetrical as possible, i.e. in the Harmony to G²MPLS direction and in the G²MPLS to Harmony direction. Nevertheless, some differences in usage is necessary, as explained in the two subsections below.

```
interface Mgmt {

       typedef sequence<g2mplsTypes::callIdent>            callIdentSeq;
       typedef sequence<g2mplsTypes::recoBundleIdent>      recoBundleIdentSeq;
       typedef sequence<g2mplsTypes::lspIdent>             lspIdentSeq;


       boolean
       callCreate(inout g2mplsTypes::callIdent             id,
                  in    g2mplsTypes::callParams            callInfo,
                  in    g2mplsTypes::recoveryParams        recoveryInfo,
                  in    g2mplsTypes::lspParams             lspInfo,
                  in    g2mplsTypes::aaaParams             aaaInfo,
                  in    g2mplsTypes::actorInfo             actor)
             raises(Types::InternalProblems);

       boolean
       callDestroy(in g2mplsTypes::callIdent               id,
                   in g2mplsTypes::actorInfo               actor)
             raises(Types::InternalProblems, Types::CannotFetch);

       boolean
       callSetTna(in g2mplsTypes::callIdent                id,
                  in g2mplsTypes::resourcePosition         pos,
                  in g2mplsTypes::tnaResource              tnaRes,
                  in g2mplsTypes::actorInfo                actor)
             raises(Types::InternalProblems, Types::CannotFetch);

       boolean
       callAddEroPart(in g2mplsTypes::callIdent            id,
                      in g2mplsTypes::eroSeq               eroItem,
                  in g2mplsTypes::actorInfo                actor)
             raises(Types::InternalProblems, Types::CannotFetch);

       boolean
       callSetUp(in g2mplsTypes::callIdent                 id,
                 in g2mplsTypes::actorInfo                 actor)
             raises(Types::InternalProblems, Types::CannotFetch);

       boolean
       callSetDown(in g2mplsTypes::callIdent               id,
                   in g2mplsTypes::actorInfo               actor)
             raises(Types::InternalProblems, Types::CannotFetch);
```

```
        callIdentSeq getCalls()
                raises(Types::InternalProblems);

        boolean
        callGetDetails(in g2mplsTypes::callIdent            id,
                       out g2mplsTypes::callParams          callInfo,
                       out g2mplsTypes::recoveryParams      recoveryInfo,
                       out g2mplsTypes::lspParams           lspInfo,
                       out g2mplsTypes::actorInfo           actor,
                       out g2mplsTypes::statesBundle        states,
                       out recoBundleIdentSeq               recoBundles)
                raises(Types::InternalProblems, Types::CannotFetch);

        boolean
        callGetTna(in  g2mplsTypes::callIdent               id,
                   in  g2mplsTypes::resourcePosition        pos,
                   out g2mplsTypes::tnaResource             tnaRes)
                raises(Types::InternalProblems, Types::CannotFetch);

};
```

Code 3-1: NCC CorbaController *Mgmt* interface.

```
interface EastWest {

        boolean
        callCreate(inout g2mplsTypes::callIdent                 id,
                         in g2mplsTypes::callParams             callInfo,
                         in g2mplsTypes::recoveryParams         recoveryInfo,
                         in g2mplsTypes::lspParams              lspInfo,
                         in g2mplsTypes::aaaParams              aaaInfo,
                         in g2mplsTypes::actorInfo              actor)
                raises(Types::InternalProblems);

        boolean
        callSetTna(in g2mplsTypes::callIdent                    id,
                      in g2mplsTypes::resourcePosition          pos,
                      in g2mplsTypes::tnaResource               tnaRes,
                      in g2mplsTypes::actorInfo                 actor)
                raises(Types::InternalProblems, Types::CannotFetch);

        boolean
        callSetGnsTna(in g2mplsTypes::callIdent                 id,
                      in g2mplsTypes::resourcePosition          pos,
                      in g2mplsTypes::gridParams                gnsTna,
                      in g2mplsTypes::actorInfo                 actor)
                raises(Types::InternalProblems, Types::CannotFetch);

        boolean
        callDestroy(in g2mplsTypes::callIdent                   id,
                       in g2mplsTypes::actorInfo                actor)
                raises(Types::InternalProblems, Types::CannotFetch);

        boolean
        callRaiseMsgEvent(in g2mplsTypes::callIdent             id,
                          in g2mplsTypes::callMessageType       msgType,
                          in string                             reason,
                          in g2mplsTypes::actorInfo             actor)
                raises(Types::InternalProblems);
```

```
};
```

Code 3-2: NCC CorbaController *EastWest* interface.

3.2.1 From Harmony to G²MPLS

In the direction Harmony to G²MPLS, the actions of the gateway through the two CallController interfaces flow according to the following:

1. The G² Call is created via *Mgmt::callCreate()*. Some of the Call and LSP parameters are mapped from the Harmony service request; others will use configured values (e.g. Switching Capability, type of restoration, etc.). A crucial Call parameter is the Call type, which will be discussed below.

2. The endpoints of the G² Call are then configured via *Mgmt::callSetTna()*. This will allow to set the ingress and egress TNA + (optionally) specific resources (i.e. a specific DataLink in the TNA, and a specific label). Both TNAs and related resources are derived from the Harmony service request. The configuration of endpoints is limited to *network* TNAs (i.e. it does not include GNS TNAs), since GNS-augmented TNAs are not handled by Harmony.

3. Once the G² Call is equipped, the GW will start its setup with the *Mgmt::callSetUp()* method. As a consequence of this, the NCC will perform basic actions (e.g. authorize the request via the GAAA-TK, route the G² Call towards the specified egress TNA, etc.), and then issue a G.7713 *SetupRequest* message to the next NCC.

4. From now on, the GW keeps synchronized with the Call signalling tiers by means of the *EastWest* interface. The *EastWest::callRaiseMsgEvent()* method will inform the GW of the successful reception of signalling messages (including errors).

    a. The message event in a successful Call setup concerns the *SetupIndication*.

    b. In case of error, an event concerning a *CallSigError* message is received

5. On a *SetupIndication* event, the GW can confirm the G² Call setup raising a *SetupConfirm* at the NCC, using the same method *EastWest::callRaiseMsgEvent().* Once this has been done successfully, the GW can consider its Call as *Active*, i.e. the Call and all the related LSPs, traversing the various domains, are up & running. The GW can now return to Harmony and close its service request.

In case of any error in the steps above, the GW needs to take specific actions. In case of a pre-setup failure (i.e. of a *Mgmt::callSetTna()*), the GW needs to clean things at the NCC using *Mgmt::callDestroy()*. In case of a very early failure (i.e. of a *Mgmt::callCreate()*) no cleaning is needed. In case of a failed signalling (i.e. reception of a *CallSigError* event), the G² Call FSMs at the NCCs will automatically converge to a clean state.

3.2.2 From G²MPLS to Harmony

In the direction G²MPLS to Harmony, the gateway sleeps on the *Mgmt::EastWest* interface, waiting for an incoming G² Call signalling. On that event, the gateway will take a number of actions, as in the following:

1. When a new G² Call reaches the GW, the *EastWest::callCreate()* method is invoked at the GW by the related NCC. With this method, the NCC transfers all its knowledge about the new Call (i.e. Call, Recovery and LSP parameters).

2. The knowledge transfer is completed with a set of *EastWest::callSetTna()* and *EastWest::callSetGnsTna()* invocations by the NCC to the GW, targeting both the ingress and egress endpoints, in terms of both TNAs (+ resources) and GNS TNAs. Actually, the passed GNS TNA is purely informational for the GW, since Harmony will not be able to resolve GNS TNA specifications. However, in case of anycast G² Call specification at the source, the GNS TNA has already been mapped into a TNA by NCC-1 at the originating G.UNI. Thus, the egress GNS TNA at this *EastWest* interface is accompanied by an egress TNA.

3. Once the G² Call information has been completely transferred to the GW, the NCC issues an *EastWest::callRaiseMsgEvent()* concerning the incoming *SetupRequest*.

4. From now on, the GW keeps synchronized with the G² Call signalling tiers. As a first step, it will propagate the Call request to its Harmony side, and wait for the result.

    a. In case of successful Harmony service setup, the GW will return an *EastWest::callRaiseMsgEvent()* invocation with a *SetupIndication* event, thus allowing the G² Call signalling to proceed in the G²MPLS CP.

    b. In case of errors from the Harmony side, the GW will return an *EastWest::callRaiseMsgEvent()* invocation with a *CallSigError* event, thus propagating it into the G² Call signalling and causing its failure.

5. Finally, the NCC will come back to the GW with an *EastWest::callRaiseMsgEvent()* invocation concerning a *SetupConfirm*. This indicates that the G² Call and related LSPs are up & running.

3.2.3 Usage of G² Call Type

The Call type in the Call Parameters specifies the behaviour of the G² Call at its boundaries, and it is valid in both gateway directions. It is used in the invocation of both *Mgmt::callCreate()* and *EastWest::callCreate()*.

The following list explains the meaning of each Call type:

- **CALLTYPE_AUTO**. In some cases (e.g. call creation by a G.UNI-GW), the Call type can be left unspecified, and it will be automatically set by the network side. This value indicates this condition. Not used in the HG²-GW case.

- **CALLTYPE_SPC**. It indicates a Soft Permanent G² Call, i.e. created by NMS on the originating NCC, signalled, and terminating on an NCC; aka CALLTYPE_aMGTzMGT (see below). Not used in the HG²-GW case.

- **CALLTYPE_PC**. Permanent G² Call. Valid for G² Calls completely created by NMS. Not used in the HG²-GW case.

- **CALLTYPE_SC**. Switched G² Call, i.e. a G² Call originated by signalling (at the G.UNI), and signalled. Similar to the following values, but no gateways are attached at the two ends of the Call: each signalling tier bounces automatically to the next one within the CCCs. Not used in the HG²-GW case.

- **CALLTYPE_aUGWzUGW**. G.UNI-GW originated G² Call, terminating on a far-end G.UNI-GW. Each signalling tier is exported to the G.UNI-GW (it does not bounce automatically). Not used in the HG²-GW case.

- **CALLTYPE_aMGTzEGW**. A Mgmt-originated G² Call (e.g. by some NMS), terminating on a G.E-NNI GW (e.g. the HG²-GW). The signalling tiers bounce automatically on the originating side, but are exported to the GW on the terminating side. This type could be used in the HG²-GW case for those G² Call that are planned and setup via management, targeting a TNA which is known to be out of the G²MPLS controlled domains (e.g. belonging to a Harmony-managed domain).

- **CALLTYPE_aEGWzMGT**. The opposite of the previous type: A G² Call originated by a G.E-NNI GW (e.g. the HG²-GW), but terminating on an NCC. The signalling tiers bounce automatically on the terminating side, but are exported to the GW on the originating side. This type could be applied to a G² Call entering G²MPLS from Harmony, and targeting a TNA that, according to G²MPLS knowledge, is not attached to a G.UNI or a G.E-NNI.

- **CALLTYPE_aUGWzEGW**. A G.UNI-GW originated G² Call, terminating on a G.E-NNI GW (e.g. the HG²-GW). The signalling tiers are exported to GWs on both sides. This type could be used in a quite common HG²-GW scenario: a G² Call is originated by a G.UNI-GW (either in an anycast fashion – i.e. with an egress GNS TNA – or not – i.e. with a TNA), and NCC-1 determines that the egress TNA (either specified or derived from the GNS TNA) is out of the G²MPLS controlled domains; thus meaning that the Call signalling will need to sink on a G.E-NNI GW (e.g. HG²-GW) interface.

- **CALLTYPE_aEGWzUGW**. The opposite of the previous type: a G.E-NNI GW originated G² Call, terminating on a G.UNI-GW. The signalling tiers are exported to GWs on both sides. This type could be used in a quite common HG²-GW scenario: a G² Call is created by the HG²-GW, targeting a TNA that is known to be crossed by a G.UNI interface; thus meaning that the Call signalling will need to sink on a G.UNI-GW interface.

- **CALLTYPE_aEGWzEGW**. The “man in the middle” case: a G² Call is originated by a G.E-NNI GW, and will terminate on a G.E-NNI.GW. The signalling tiers are exported to GWs on both sides. This type is used when a G²MPLS network segment lies between two non-G²MPLS segments. A G² Call is created by the HG²-GW, targeting a TNA that is known to be out of G²MPLS controlled domains; thus meaning that the Call signalling will need to sink on a G.E-NNI GW (e.g. HG²-GW) interface.

3.3 HG² GW Translator

3.3.1 Signalling mappings

The translation of parameters and calls from one side to the other is made at the HG^2 GW translator. Some parameters are directly translated since they are used on both domains. Others have to be statically configured at the HG^2 GW and added to the corresponding calls.

3.3.2 Signalling mappings

Correspondence between the Harmony signalling methods and the G^{2}MPLS signalling methods is shown in next table. This correspondence is mapped at the translator for creating connections, deleting connections and getting connection status.

Harmony method	G^{2}MPLS calls
createReservation(parameters)	CallController::Mgmt::createCall(parameters) CallController::Mgmt::callSetTna(parameterss) CallController::Mgmt::callSetUp(parameters) *****wait() CallController::EastWest::callRaiseMsgEvent(<SetupConfirm>)
cancelReservation(parameters)	CallController::Mgmt::callSetDown(callId) *****wait()
getStatus(parameters)	CallController::Mgmt::callGetDetails(callId)
getReservations(parameters)	CallControler::Mgmt::getCalls()

Table 3-1: Signalling method mappings

Each method has the following parameter translation:

Create reservation

Harmony parameters	G2MPLS parameters
serviceID	g2mplsTypes::callIdent->localId
startTime	g2mplsTypes::callParams->timeInfo
duration	g2mplsTypes::callParams->timeInfo
sourceEndpoint	g2mplsTypes::lspIdent->srcNodeId
targetEndpoint	g2mplsTypes::lspIdent->dstNodeId
minBw	g2mplsTypes::lspParams->bw

Table 3-2: Create Reservation parameter translation

These are the basic parameters in a structured way. It must be noticed that a service could contain one or more connections and each connection can have one or more target endpoints (point-to-point, point-to-multipoint)

Cancel reservation

Harmony parameters	G2MPLS parameters
reservationID: String	callId

Table 3-3: Cancel reservation parameter translation

GetStatus

Harmony parameters	G2MPLS parameters
reservationID: String	callId

Table 3-4: Get status parameter translation

3.3.3 Routing mappings

Harmony and G^{2}MPLS topology object bidirectional translation is performed at each HG^2 GW. The parameters that are present in both domains are directly mapped, like the inter-domain link bandwidth. G^{2}MPLS Domain parameters needed by Harmony IDB are statically configured in the gateway under a domain identifier and vice versa. The following tables show the direct translation of parameters. Mandatory parameters that can't be mapped should be configured statically at the HG^2 GW.

Harmony parameters	G2MPLS parameters
DomainId	nodeIdent

Table 3-5: Translation of domain identifiers

Harmony parameters	G2MPLS parameters
SourceEndpoint LinkId DestinationDomain	localNodeId localId remoteNodeId remoteId localRcId remoteRcId

Table 3-6: Translation of inter-domain link identifiers

Harmony parameters	G2MPLS parameters
TNAPrefix	rc node tna prefix

Table 3-7: Translation of TNA identifiers

4 Conclusions

This deliverable has focussed on interoperability between G²MPLS and Harmony to enable automatic setup of inte-domain connections in Phosphorus overlay mode. It provided insight on the interworking architectures and interoperability scenarios between G²MPLS and Harmony system. It also defined the signalling and routing specifications of the HG^2 GW (Harmony - G²MPLS gateway) and methods that have been implemented to provide this gateway.

The descriptions in this deliverable have already led to a working prototype, delivered jointly by WP2 and WP1. Results of running the prototype will be evaluated and enable us to further specify and improve the specifications of the gateway and supporting software modules.

5 References

[PH-WP2-D2.1] Phosphorus deliverable D2.1, “The Grid-GMPLS Control Plane architecture”.

[PH-WP2-D2.2] Phosphorus deliverable D2.2, “Routing and Signalling Extensions for the Grid-GMPLS Control Plane”.

[PH-WP2-D2.6] Phosphorus deliverable D2.6, “Deployment models and solutions of the Grid-GMPLS Control Plane”.

[PH-WP2-D2.7] Phosphorus deliverable D2.7, “Grid-GMPLS network interfaces”.

[PH-WP2-D2.3] Phosphorus deliverable D2.3, “Grid-GMPLS high level system design”.

[PH-WP2-D2.4] Phosphorus deliverable D2.4, “Report on Grid-GMPLS Control Plane functional tests”.

[PH-WP1-D1.3] Phosphorus deliverable D1.3, “NRPS southbound interfaces for standard GMPLS control plane”

[PH-WP1-D1.4] Phosphorus deliverable D1.4. “Definition and development of the Network Service Plane and northbound interfaces development”

6 Acronyms

ARGON	Allocation and Reservation in Grid-enabled Optic Networks
ASON	Automatically Switched Optical Network
DRAC	Dynamic Resource Allocation Controller
E-NNI	External Network-to-Network Interface
G.OUN	Grid O-UNI
G2MPLS	Grid-enabled GMPLS
G-E.NNI	Grid E-NNI
G-I.NNI	Grid I-NNI
GMPLS	Generalized Multi Protocol Label Switching
GNS	Grid Network Services
HG^2 GW	Harmony G^{2}MPLS GateWay
HSI	Harmony Service Interface
IETF	Internet Engineering Task Force
I-NNI	Internal Network-to-Network Interface
LSA	Link State Advertisement
LSP	Label Switched Path
NCC	Network Call Controller
NMS	Network Management System
NREN	National Research and Education Network
NRPS	Network Resource Provisioning System
NSP	Network Service Plane
O-UNI	Optical User Network Interface
PCE	Path Computation Element
RSVP	Resource Reservation Protocol
TNA	Transport Network Address
UCLP	User Controlled LightPaths
WS	Web Service
SOA	**Service-Oriented Architecture**
HAL	**Harmony Adaptation Layer**
IDB	**? InterDomain Broker?**
IDB-R	**IDB for Routing**
IDB-S	**IDB for Signalling**
E-NNI	**External Network-to-Network Interface**

RSVP	**Resource reSerVation Protocol**
OSPF	**Open Shortest Path First**
DCM	**Distributed Call and Connection Management**
RSVP-TE	**Resource Reservation Protocol - Traffic Engineering**
CCC	**Calling/Called Party Call Controller**
NCC	**Network Call Controller**
RC	**Routing Controller**
SCN	**Signalling Communication Network**
WSDL	**Web Service Description Language**