034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.2.3

Grid-GMPLS high-level system design

Due date of deliverable: 2008-03-31
Actual submission date: 2008-03-31
Document code: Phosphorus-WP2-D2.3

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable: **Nextworks (NXW)**

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 document reports the software architecture of the main network elements of the Grid-enabled GMPLS Control Plane.
Functional entities are specified in terms of interfaces, both internal to a G^{2}MPLS network element and external (i.e. towards other peering network elements).
Moreover, for each components of a G^{2}MPLS network element, a detailed breakdown of functionalities, data model, finite state machines and exported APIs is provided in terms of code/pseudo-code excerpts, in order to assemble a generalized but possibly detailed guide for the G^{2}MPLS software developers.

Project:	Phosphorus
Deliverable Number:	D.2.3
Date of Issue:	31/03/08
EC Contract No.:	034115
Document Code:	Phosphorus-WP2-D2.3

List of Contributors

Giacomo Bernini	NXW	Damian Parniewicz	PSNC
Gino Carrozzo	NXW	Artur Juszczyk	PSNC
Nicola Ciulli	NXW	Eduard Escalona	UESSEX
Giodi Giorgi	NXW	Reza Nejabati	UESSEX
Francesco Salvestrini	NXW	Adetola Orepdope	UESSEX
Adam Kaliszan	PSNC	Oliver Wäldrich	FHG-SCAI

Table of Contents

0	Executive Summary	13
1	Objectives and Scope	14
2	Terminology	17
3	High-level system design of G^{2}MPLS network elements	18
3.1	G^{2}MPLS Edge Controller	22
3.1.1	Main functionalities	22
3.1.2	External interfaces	23
3.1.3	Internal interfaces	24
3.2	G^{2}MPLS Core Controller	26
3.2.1	Main functionalities	26
3.2.2	External interfaces	26
3.2.3	Internal interfaces	27
3.3	G^{2}MPLS Border Controller	30
3.3.1	Main functionalities	30
3.3.2	External interfaces	30
3.3.3	Internal interfaces	31
3.4	G.UNI Gateway (G.UNI-GW)	34
3.4.1	Main functionalities	34
3.4.2	External interfaces	34
3.4.3	Internal interfaces	35
3.5	G.E-NNI Gateway (G.ENNI-GW)	36
4	Transport Network Resource Controller (TNRC)	37
4.1	TNRC basics	37
4.2	TNRC data model	38
4.2.1	TNRC_Master instance	39
4.2.2	TNRC_AP instance	41
4.2.3	Eqpt instance	42
4.2.4	Board instance	43

Project: Phosphorus
Deliverable Number: D.2.3
Date of Issue: 31/03/08
EC Contract No.: 034115
Document Code: Phosphorus-WP2-D2.3

4.2.5 Port instance 45
4.2.6 Resource instance 46
4.2.7 Plugin Container (Pcontainer) instance 48
4.2.8 Plugin instance 48
4.2.9 ApiQueue instance 51
4.2.10 Action instance 51
4.2.11 XC instance 53
4.3 TNRC Abstract Part 55
4.3.1 TNRC Abstract Part configuration API 55
4.3.2 TNRC Abstract Part external API 57
4.3.3 TNRC Abstract Part action specific API 59
4.4 TNRC Specific Part 60
4.4.1 TNRC Specific Part API 60
4.5 TNRC Action FSM 60
4.5.1 Example transitions 66

5 Link Resource Manager (LRM) 69
5.1 LRM basics 69
5.2 LRM Data Model 69
5.2.1 LRM instance 70
5.2.2 SCN Interface instance 71
5.2.3 Control Channel instance 71
5.2.4 Adjacency instance 72
5.2.5 TE-Link instance 72
5.2.6 Data-Link instance 74
5.3 LRM configuration API 75
5.4 LRM external API 77

6 SCN Gateway (SCNGW) 79
6.1 SCNGW basics 79
6.2 SCNGW client 81
6.3 SCNGW server 84
6.3.1 SCNGW server data structures 84
6.3.2 SCNGW server external API 87

7 G^2.RSVP-TE 89

7.1 G^2.RSVP-TE data model 90
7.1.1 G^2.RSVPTE instance 91
7.1.2 Session instance 92
7.1.3 LSP instance 93
7.1.4 Interface instance 97
7.2 G^2.RSVP-TE internal API 98
7.3 G^2.RSVP-TE external API 101
7.4 G^2.RSVP-TE LSP FSM 103
7.4.1 Example transitions 111
7.5 G^2.RSVP-TE parsing and formatting 112

8 Call Controllers 117
8.1 CC shared objects and functions (xCC) 117
8.1.1 xCC data model 118
8.1.2 xCC (CCC/NCC) External API 120
8.1.3 xCC Signalling Interfaces 124
8.2 G^2.NCC – The Grid-GMPLS Network Call Controller 128
8.2.1 G^2.NCC basics 128
8.2.2 G^2.NCC software overview 128
8.2.3 G^2.NCC data model 130
8.2.4 G^2.NCC Call FSM 132
8.3 G^2.CCC – The Grid-GMPLS Client Call Controller 139
8.3.1 G^2.CCC basics 139
8.3.2 G^2.CCC software overview 139
8.3.3 G^2.CCC data model 141
8.3.4 G^2.CCC Call FSM 142

9 Recovery Controller (RC) 147
9.1 Recovery Controller basics 147
9.2 Recovery Controller software overview 148
9.3 Recovery Controller data model 149
9.4 RC Recovery Bundle FSM 151
9.5 Recovery Controller External APIs 158

10 G^{2}MPLS Path Computation Engine Routing Algorithm (G^2.PCE-RA) 162
10.1 G^2.PCE-RA basics 162

10.2 Topology view in G^2.PCE-RA 164
10.3 G^2.PCE-RA data model 166
10.3.1 G^2.PCE-RA instance 167
10.3.2 GNS calls 168
10.3.3 Connections 169
10.3.4 Topology 169
10.3.5 Nodes 170
10.3.6 TNAs 176
10.3.7 TE Links 176
10.4 G^2.PCE-RA internal API 179
10.4.1 Topology update in G^2.PCE-RA 179
10.4.2 Computation of routes in G^2.PCE-RA 186
10.5 G^2.PCE-RA external API 190
10.5.1 Topology API 190
10.5.2 Computation API 196

11 G.UNI-GW Adapter Design Specification 199
11.1 G.UNI-GW Adapter Transactions 199
11.1.1 WSAG – WS-G.UNI Adapter – G.UNI-C RSVP PC (Signalling) 200
11.2 G.UNI-GW adapter Implementation 206
11.2.1 File descriptions 207
11.3 Example 207

12 G.UNI and G.E-NNI RSVP-TE 208

13 G^2.OSPF-TE (I-NNI, E-NNI and UNI-N/C) 209

14 Software structure 210
14.1 Configuration process 210
14.1.1 The configuration process from the user perspective 210
14.1.2 The configuration process from the developer perspective 212
14.2 Process start-up and monitoring 212
14.3 Inter-process communication 214
14.3.1 omniORB 215
14.3.2 Quagga daemons and threads 216
14.3.3 omniORB integration in Phosphorus 216

14.4 G²MPLS base Python modules 218
14.5 Software daemons 220
14.5.1 lrmd 220
14.5.2 scngwd 220
14.5.3 tnrcd 221
14.5.4 ospfd 222
14.5.5 g2rsvpted 222
14.5.6 gunirsvpd 222
14.5.7 gennirsvpd 223
14.5.8 g2nccd 223
14.5.9 g2pcerad 223
14.5.10 lib 223
14.5.11 pyg2mpls 224

15 References 225

16 Acronyms 226

Appendix A Common types 230
A.1 Identifiers 230
A.2 Label identifier 231
A.3 TE-Link and Data Link 231
A.4 TNA identifier 231
A.5 Call, Recovery Bundle and LSP identifiers 232
A.6 GMPLS extensions 233
A.7 Grid extensions 236
A.7.1 Signalling-specific 236
A.7.2 Routing-specific 240
A.8 GNS call parameters 245
A.9 Recovery parameters 246
A.10 LSP parameters 246
A.11 ERO 247
A.12 LRM specific 247
A.13 TNRC specific 248
A.14 G².PCE-RA specific 248

Appendix B	Automatic FSM skeleton generation	251
B.1	Configuration file	251
B.2	Template file	252
B.3	Generated code	255
Appendix C	TNRC Specific Part for ADVA FSP 3000RE-II	266
C.1	API Data structures	266
C.2	Summary of TNRC_SP LSC ADVA API functions	267
C.3	Detailed specification of TNRC_SP LSC ADVA API functions	267
C.4	ADVA FSP 3000RE-II device	275
C.4.1	Overview	275
C.4.2	Implementation details	278
C.4.3	TL1 commands	285
C.4.4	TL1 autonomous messages	290
C.4.5	Error codes	291
Appendix D	TNRC Specific Part for Calient DiamondWave FiberConnect	297
D.1	Calient TNRC_SP Software Design	297
D.1.1	Data structures	297
D.1.2	Detailed specification of TNRC_SP FSC API functions	298
D.2	Calient TNRC_SP Software Implementation	305
D.2.1	TNRC_SP use-case scenarios	305
D.2.2	TNRC_SP_Calient Generic Descriptions	316

List of Figures

Figure 3-1: Generic functional decomposition of G²MPLS controllers. 18
Figure 3-2: G²MPLS network elements. 20
Figure 3-3: The G^2MPLS Network Control Plane with gateway functional elements. 21
Figure 3-4: Internal components of the G2MPLS Edge Controller. 25
Figure 3-5: Internal components of the G2MPLS Core Controller. 29
Figure 3-6: Internal components of the G2MPLS Border Controller. 33
Figure 3-7: G.UNI Gateway (G.UNI-GW) breakdown into main components. 35
Figure 4-1: TNRC data model. 39
Figure 4-2: TNRC actions finite state machine. 65
Figure 4-3: TNRC action FSM: example transitions in case of successfully make cross-connection. 67
Figure 4-4: action FSM: example transitions in case of successfully destroy cross-connection. 68
Figure 5-1: LRM Data Model. 70
Figure 6-1: SCNGW module structure. 80
Figure 7-1: The base G².RSVP-TE data model. 90
Figure 7-2: G^2MPLS LSP finite state machine 110
Figure 7-3: Example of G^2MPLS LSP signalling setup 112
Figure 7-4: parsing and formatting sketch. 113
Figure 8-1: The base xCC data model 119
Figure 8-2: G².NCC threads structure 130
Figure 8-3: G².NCC data model 131
Figure 8-4: G².NCC Call FSM. 138
Figure 8-5: G².CCC threads structure 140
Figure 8-6: G².CCC data model 141
Figure 8-7: G².CCC Call FSM. 146
Figure 9-1: RC threads structure 149
Figure 9-2: RC data model 150
Figure 9-3: Recovery Bundle FSM 157
Figure 10-1: The G^2.PCE-RA component break-down. 163
Figure 10-2: Mixed topology with three domains, inter and intra-domain te-links and Grid sites. 164
Figure 10-3: G^2.PCE-RA representation of the previous mixed topology. 165
Figure 10-4: The base G^2.PCE-RA data model. 166
Figure 10-5: G^2.PCE-RA constrained Dijkstra pseudo-code. 186
Figure 10-6: Actions on a callRoute(). 188
Figure 10-7: Actions on a computeDisjointRoute(). 189
Figure 10-8: Actions on a computeMaxDisjointRoutes(). 190
Figure 11-1: The GUNI-GW breakdown and transactions localization. 200
Figure 11-2: GUNI-GW operation flow. 206
Figure 14-1: Phosphorus G2MPLS code structure. 211
Figure 16-1: Test FSM. 265
Figure 16-2: ADVA FSP 3000RE-II device. 276
Figure 16-3: ADVA FSP 3000RE-II architecture. 277
Figure 16-4: ADVA FSP 3000RE-II eROADM connections configuration (AID are “bay-shelve-slot-port”). 278
Figure 16-5: TNRC SP ADVA sequence diagram for XC creation and fault notification. 280
Figure 16-6: TNRC SP ADVA sequence diagram for information retrieve (get_resource_list, get_resource_detail, get_label_list). 281

Figure 16-7: TNRC SP ADVA make xc finite state machine (‘Destroy xc’ is entry point to destroy xc finite state machine). 282
Figure 16-8: TNRC SP ADVA destroy xc finite state machine (‘Unreserve xc’ is entry point to unreserve xc finite state machine). 283
Figure 16-9: TNRC SP ADVA reserve xc finite state machine (‘Unreserve xc’ is entry point to unreserve xc finite state machine). 284
Figure 16-10: TNRC SP ADVA unreserve xc finite state. 285
Figure 16-11: Uses Case Diagram for the TNRC_SP. 307
Figure 16-12: Process and threads sequential diagram. 317
Figure 16-13: State Diagram for Make XC. 318
Figure 16-14: State Diagram for Destroy XC. 319
Figure 16-15: State Diagram for Reserve Resources. 319
Figure 16-16: State Diagram for Un-reserve Resources. 320

List of Tables

Table 3-1: G^2MPLS Edge Controller external interfaces. 23
Table 3-2: G^2MPLS Edge Controller internal interfaces. 24
Table 3-3: G^2MPLS Core Controller external interfaces. 27
Table 3-4: G^2MPLS Core Controller internal interfaces. 28
Table 3-5: G^2MPLS Border Controller external interfaces. 31
Table 3-6: G^2MPLS Border Controller internal interfaces. 32
Table 3-7: G.UNI Gateway external interfaces. 34
Table 3-8: G.UNI Gateway internal interfaces. 35
Table 4-1: TNRC breakdown in sub-modules. 38
Table 4-2: TNRC Action FSM: states. 63
Table 4-3: TNRC Action FSM: events and root events. 64
Table 6-1: SCNGW breakdown into two sub-modules. 80
Table 7-1: Mapping between internal and external G^2.RSVP-TE API 103
Table 7-2: G^2.RSVP-TE LSP FSM: states 108
Table 7-3: G2RSVP-TE LSP FSM: root events 109
Table 8-1: G^2.NCC Call FSM: states 136
Table 8-2: G^2.NCC Call FSM: root events 137
Table 8-3: G^2.CCC Call FSM: states 144
Table 8-4: G^2.CCC Call FSM: root events 145
Table 9-1: RC Recovery Bundle FSM: states 154
Table 9-2: RC Recovery Bundle FSM: root events 156
Table 14-1: SCNGW breakdown in sub-modules. 221
Table 14-2: TNRC breakdown in sub-modules. 222

0 Executive Summary

This document reports the high-level software architecture of the main network elements of the Grid-enabled GMPLS Control Plane. The final purpose is to provide a reference on the G^2MPLS software modules, their expected functionalities and core structure for the main use by Phosphorus developers.

The actual scope of the document is stated in section 1, which provides the guiding information on how to read and use the whole document.

Section 2 introduces into the used terminology and refers to the acronyms list in section 16.

Section 3 describes the actual and implemented high-level software design of the G^2MPLS controllers (Edge, Core and Border Controllers); the rest of the document goes beyond it in order to offer a needed insight on the developed software, according to the overall scope of the deliverable.

Sections 4 to 13 reports on the software design of each single component in the G^2MPLS stack: the TNRC, the LRM, the SCNGW, the G^2.RSVP-TE, the G^2.OSPF-TE the Call Controllers (CCC and NCC), the Recovery Controller, the G^2.PCE-RA, the G.UNI and G.E-NNI RSVPs, the G.UNI and G.E-NNI OSPFs, and the G.UNI GW.

Finally, section 14 discusses the software structure of the phosphorus-g2mpls open source code, including some software-architectural details on the stack.

A set of appendixes introduces further details that could have been cumbersome if aggregated in previous sections. Appendix A reports software details on the G^2MPLS data model. Appendix B describes the software utilities to automate and streamline the generation of protocol Finite State Machines starting from human-readable specifications. Finally, appendixes C and D reports the software design of the TNRC SPs for the reference equipment to be controlled by G^2MPLS in the Phosphorus project (ADVA FSP 3000RE-II and the Calient DiamondWave FiberConnect, respectively).

1 Objectives and Scope

Foreword: this deliverable just refers to architectural and functional concepts of GMPLS and G²MPLS, but it does not explain them in depth. For a support on the basics, please refer to the architectural and protocol-specific documents already delivered by WP2 (D2.1, D2.2, D2.7 and, for some matters, D2.6). Furthermore, in order to keep the document focused on its scope, all the references to standards (by IETF, ITU-T and OIF), be them either normative or informational, are not listed again here, but are pointed to their corresponding listing in D2.1, D2.2 and D2.7.

This document reports the high-level software design of the Grid-enabled GMPLS Control Plane stack. However, this document has been originally planned at the end of the first round of development process (i.e. before any integration and testing) and is intended to contain much more than just a high-level software design. Indeed, this document is a detailed report about the G²MPLS software developed in the first 18 months of the Phosphorus project. With this scope, it includes:

- a level of information that is comparable to an a posteriori high-level software design (i.e. already implemented and preliminarily tested), not adopting the formalism (e.g. UML) needed when the design is a priori, and have to drive subsequent developments. For a greater efficiency, this form of design was produced during the developments themselves, thus not decoupling the system design phase (and teams) from the development counterparts. In other words, the reported design already includes the upgrades and fixes that occur when it goes through the real development process.

- An insight in the developed software, with varying levels of detail (according to the needs of each particular piece of software). The view might range from a high-level functional description up to the discussion of specific algorithms (e.g. for the PCE).

- In general, any sort of information that could make this deliverable a solid reference document about the G²MPLS software in the future, both for who contributed to the developments, and who will be using and modifying the Open Source Software released after it.

For the latter reason (need to be a quick and effective reference for the code), the style and mood of this document is less tutorial and verbose than that adopted in the architectural deliverables (D2.1, D2.2, D2.7 and D2.6).

Details that go beyond the listed specifications will derive from the completion of the development activities and the release of the G^2MPLS prototypes. In fact, this document is intended as a basic general reference that will integrate the official G^2MPLS architectural documents (D2.1 and D2.6) and provide some detailed descriptions of the G^2MPLS prototypes that will be delivered by WP2. Further details will be provided by the planned prototype deliverables, D2.5 and D2.10.

The network elements are characterized based on their roles in the G^2MPLS network, their functionalities and the interfaces they expose, both externally (toward other network elements) and internally (between their different functional entities). For this reason, there is not a single G^2MPLS stack software architecture, but a number of them, one for each G^2MPLS node configuration and role in the network (see section 3). The proposed and implemented software structure is composed of functionally complete and independent modules, which allows flexible integration, gradual development and could potentially be extended with modules by other developers.

However, the functional modules and components (protocol daemons, inter-process communication, utilities, etc.) of all these architecture converge into a common set, which make up the so-called G^2MPLS stack. In other words, the G^2MPLS is not a running set of processes and threads, but the collection of the software pieces that, properly installed and configured, allow to create and run specific G^2MPLS Controllers (Edge, Core and Border, as specified in D2.1).

An additional note concerns the subjects (modules) of the reported software design (sections 4 to 13). As a matter of fact, the scope and content of the developed software and, consequently, of this document, go much beyond what planned for WP2 in the Phosphorus Description of Work. That planning assumed to start from an existing Open Source GMPLS stack; however, at the start of the developments no stack was matching the Phosphorus functional requirements. This condition was among the technical risks analyzed during the project setup, and a backup plan was already available, and promptly implemented: contributing extra effort to WP2 and developing the needed components to set an adequate starting point for the G^2 developments.

Building a house from scratch has some relevant benefits, ultimately. The backup plan has led to a GMPLS stack fully owned and mastered by the developers of the G^2 extensions. Furthermore, this stack was equipped with all the needed modules and utilities that allow to fulfil to a large extent the ASON and GMPLS architectural requirements.

Thus, some of the elements (protocol controllers) can be easily identified as the name, functions and architectural placements are perfectly aligned with the ASON architecture; examples are the LRM, the G.RSVP-TE, the G.OSPF-TE. However, other elements have no counterparts in the ASON or GMPLS specifications, since they are basic and founding software modules that derive from high-level requirements set by ASON or GMPLS (e.g. the SCN Gateway is implied by the requirement that the Transport Network and the Signalling Communication Network are decoupled in GMPLS).

For this very reason, these modules had to be documented in a sufficiently detailed way, in order to provide a usable and effective tool to approach the complexity of the G^2MPLS stack.

Finally, this deliverable is not a closed document. The G²MPLS software stack will evolve in the next months: some activities are still going on (i.e. A2.2.4 – integration with the NSP/NRPSes and A2.2.3 – Integration with AAI[1]), and the system testing activities (A2.1.7) might introduce relevant upgrades. A new version of D2.3 might be produced, in order to incorporate the significant evolutions in the software after the official issue date (M18).

[1] See the G².CCC and G².NCC Call FSMs at Section 8 for the software hooks dedicated to the interaction with the Phosphorus AAI.

2 Terminology

No specific terminology is introduced by this document, which refers to Deliverable D2.1, D2.2, D2.6 and D2.7 for any specific terms used.

One note about a terminology issue: the Grid-capable Optical User Network Interface has been termed in previous WP2 deliverables as “G.OUNI”, in accordance with the terminology used in related OGF-GHPN documents. Since, during the course of time, OGF-GHPN has upgrade this term to a more general “G.UNI”, the WP2 documents has started following this new naming accordingly. Thus, in the whole set of past and future WP2 deliverables, the terms G.OUNI and G.UNI are used to refer, indifferently, to the Grid-capable User Network Interface (with or without, respectively, a specific focus on the exported optical services). In other words, the two terms refer to equivalent User-Network Interfaces, for what concerns the grid and network services exported.

A full list of the abbreviations used in this document is provided in Section 16.

3 High-level system design of G^2MPLS network elements

The generic functional decomposition of the stack components of the G^2MPLS Network Control Plane (valid for the G^2MPLS Edge, Core and Border Controllers, see below) is reported in Figure 3-1.

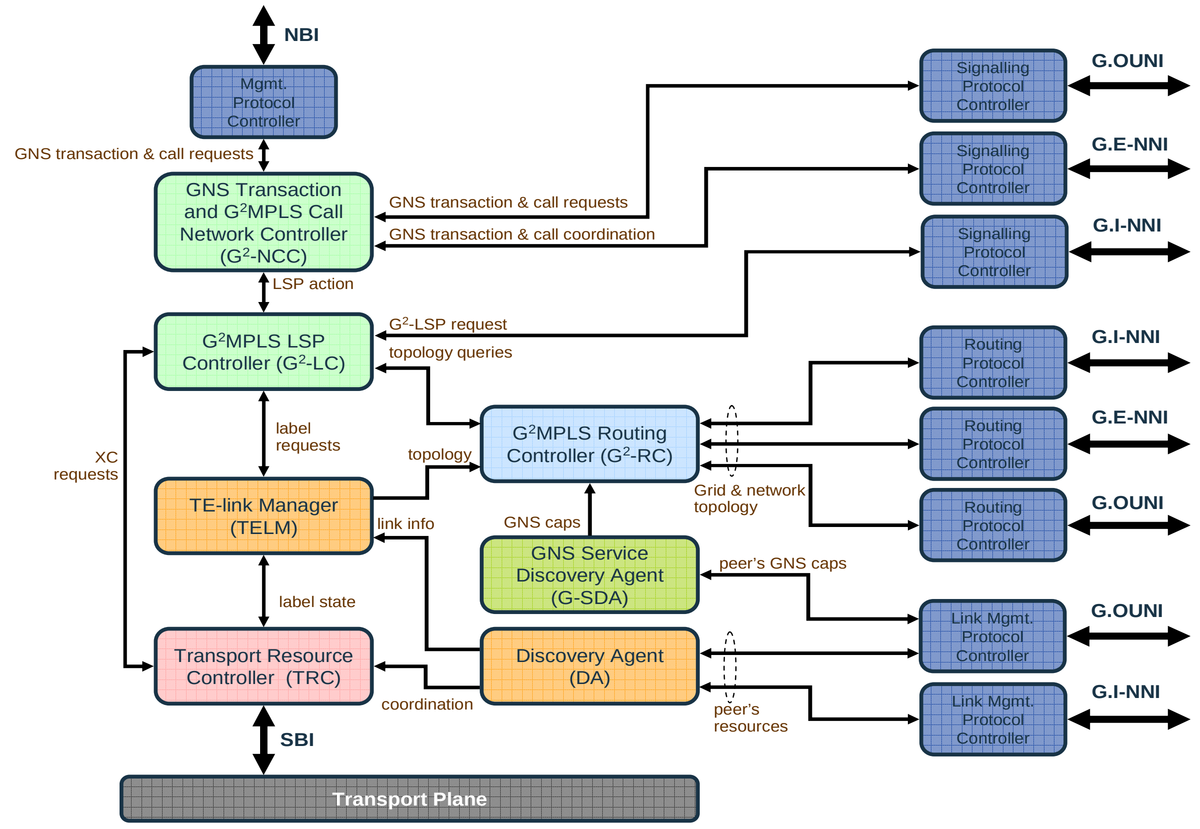

Figure 3-1: Generic functional decomposition of G^2MPLS controllers.

The software implementation of these components has been carried out starting from the v0.99.7 routing suite [QUAGGA-DOC], as explained in section 14.

Different types of network elements are identified in a G^{2}MPLS domain, depending on the role and functionalities they provide. Three of the elements are network-side, the fourth is a functional "G^2 cluster" at the customer premises:

- *G^{2}MPLS Edge Controllers*, which operate at the edge of the G^{2}MPLS domain and interface to the Control Plane user[2].
- *G^{2}MPLS Core Controllers*, which implements the functionalities of an internal node of the domain like an LSR in a GMPLS network.
- *G^{2}MPLS Border Controllers*, which operate on the domain boundary and interface the G^{2}MPLS with other domains of the same or different technology and control/provisioning architecture (e.g. NSP-NRPSes, AutoBAHN).
- *G^{2}MPLS UNI-C*: this node is the client-side counterpart of the G^{2}MPLS Edge Controller and is made up of a composition of the Edge Controller modules, plus two specialized ones (the G.UNI-GW and the Client Call Controller). Differently for the network-side G^{2}MPLS controllers, these modules could also be delocalized at different hardware platforms (e.g. the G.UNI-GW in one box, and the CCC in one other, with the rest of the UNI-C protocols). For this reason, this section does not propose or impose a specific software architecture for the whole set, but the document focuses on the single components.

Each controller is discussed in a separate subsection in the remainder of this section.

The localization of these network elements is shown in Figure 3-2, as well as the identification of the main network reference points of the controllers [PH-WP2-D2.1, PH-WP2-D2.2, PH-WP2-D2.6].

The Grid layer is typically WS-based and the choice of WS-Agreement technology has been adopted also for the Network Service Plane (which controls the NRPS layer, see D1.1 and D1.2) and the GÉANT2 BoD system (see GN2-JRA3 BoD specification documents, e.g. DJ3.4.1,2). For this reason, some form of translation from the WS context to G^{2}MPLS signalling and vice versa are needed at the external network reference points of the G^{2}MPLS NCP, i.e. the Grid-capable Optical User-Network Interface (G.UNI) and the Grid-capable External Network-Network Interface (G.E-NNI). For this purpose, two additional architectural elements are part of the G^{2}MPLS network model (ref. Figure 3-2 and Figure 3-3):

- The G.UNI gateway
- The G.E-NNI gateway

[2] In the G^{2}MPLS framework, the user is principally a Grid site with an instance of middleware issuing/receiving requests for Grid Network Services. However, the G^{2}MPLS user can fall back to a standard ASON/GMPLS user issuing just Network Service requests and in this case G^{2}MPLS control plane falls back to a GMPLS Control Plane.

The gateways are aimed to provide the needed bridging functionality between the two frameworks and preserve the core G^{2}MPLS/GMPLS signalling and routing procedures by concentrating in single points the adaptation functions.

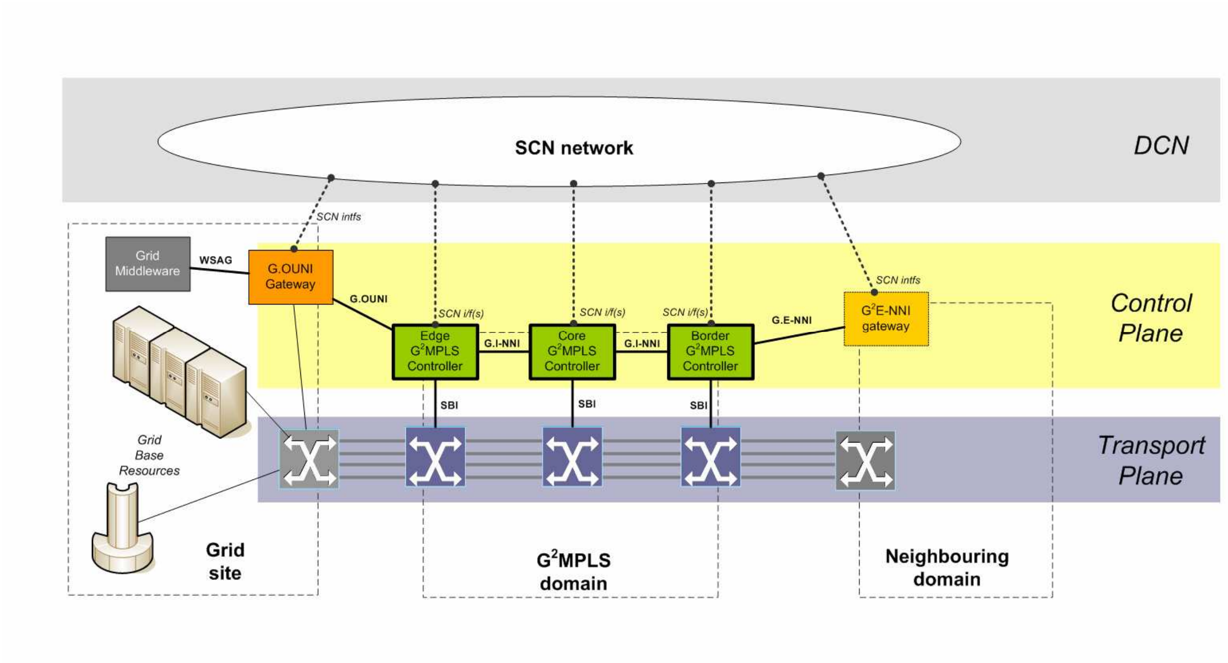

Figure 3-2: G²MPLS network elements.

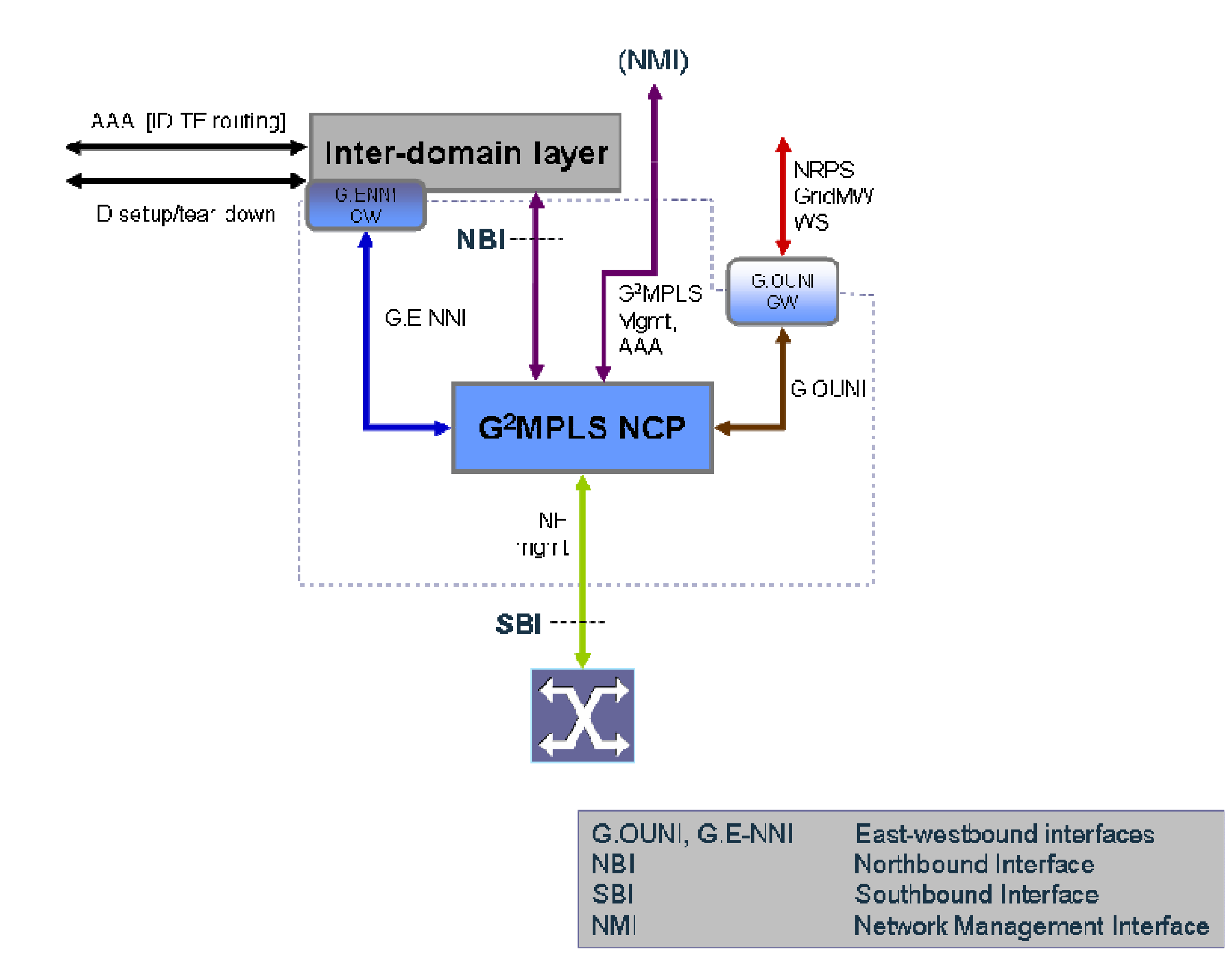

Figure 3-3: The G²MPLS Network Control Plane with gateway functional elements.

G²MPLS network elements are interconnected through network interfaces specified in [PH-WP2-D2.1, PH-WP2-D2.2, PH-WP2-D2.7]. In details:

- **G.UNI**, i.e. the Grid Optical User-Network Interface that supports Grid and network signalling and discovery between the Grid site and the G²MPLS domain.
- **G.I-NNI**, i.e. the Grid Internal Node-Node Interface (G.I-NNI) that supports the routing and signalling procedures between adjacent nodes.
- **G.E-NNI**, i.e. the Grid External Network-Network Interface that propagates Grid and network topology information across different Control Plane domains and supports the inter-domain signalling mechanisms.
- **SBI**, i.e. the Southbound Interface that retrieves resource status from the specific Transport Plane and translates Control Plane actions into appropriate configurations of those resources.

- **G.NBI**, i.e. the Northbound Interface that connect the G^2MPLS to the Grid layer and is based on WS agreements technologies.

Each G^2MPLS controller is connected to other G^2MPLS controllers through the SCN. Therefore, each G^2MPLS controller has a number (at least one) SCN interfaces on top of which SCN adjacencies are established with G^2MPLS controller that are adjacent on the Transport Plane but may be not adjacent on the DCN. This functionality is generally referred to as management of the dualism between Transport Network and Signalling Network. See D2.1 for further details.

3.1 G^2MPLS Edge Controller

3.1.1 Main functionalities

The G^2MPLS edge controller is the entry point of the G^2MPLS domain and, therefore, it is responsible for:

- the termination and control of a signalling session incoming through the UNI and initiated by an attached Grid client (G.UNI-C)
- the progression and control of a G.UNI signalling session towards an attached Grid client (G.UNI-C)
- the control of the G^2MPLS Call setup and its segment breakdown (with the scope of the domain in which it operates)
- the control of the end-to-end recovery of a call segment (inter-domain recovery is left for further studies)
- the flooding of Grid and network routing information, in terms of
  - local TE-link information directly generated
  - Grid resource availabilities received through the G.UNI by the attached G^2MPLS user
  - remote network and Grid information learned by peer routing controllers
- the computation of end-to-end explicit routes for a call and its segments. Routes are as much as possible complete and strict at least for the domain in which the Edge Controller operates, while they could be sparse and in case loose, depending on the available information published by neighbouring G^2MPLS domains
- [optional[3]] the flooding of inter-domain Grid and network routing information, in terms of
  - reception (feed-up) of topology (Grid and network) information from the domain in which it operates (level 0)
  - flooding of routing information with peering inter-domain routing controllers
  - dissemination (feed-down) of the summarized topology information about neighbouring domains towards the base routing instances operating in its domain (level 0).
- the retrieval of information (amount, status and alarms) on the Transport Network resources for G^2MPLS use in the equipment it is attached to
- the configuration (cross-connection) of Transport Network resources in the equipment it is attached to

[3] The functionality is optional because just one node in the domain configured as RC

- the control of the G^2MPLS data model (i.e. TE-links, Data-links, Control channels and SCN interfaces) in accordance with the node configuration and the Transport Network resources availabilities retrieved by the equipment
- [optional] the storage and control of the persistent data (TE-links, Calls, LSPs, resource status, etc.) across case of node restart.

These functionalities are implemented by the software components depicted in Figure 3-4.

3.1.2 External interfaces

Interface	Peer	Directionality	Main action (s)
SBI	TN equipment	in/out	▪ retrieval of information on transport resources (e.g. ports, wavelengths, ▪ configurations on transport resources (e.g. cross-connections, protections, etc.) ▪ alarm reporting on configured resources
SCN interface	adjacent G^2MPLS controller	in/out	▪ establish and maintain the adjacency between pairs of G^2MPLS controllers ▪ send/receive protocol SDUs
G.UNI	G^2MPLS user (Grid site with middleware)	in/out	▪ **signalling** o setup and monitoring of G.UNI calls ▪ **routing** o learning of Grid resource availabilities by the attached G^2MPLS user o publication of remote Grid resource availabilities learned by other routing controllers peering in the G^2MPLS domain
G.I-NNI	G^2MPLS core controller	in/out	▪ **signalling** o control (setup and recovery) of I-NNI call segments ▪ **routing** o learning of node external Grid and network (single domain and multi-domain) topology resource availabilities
[optional] G.E-NNI	G^2MPLS peering Routing Controllers	in/out	▪ **routing** o publication and learning of inter-domain Grid and network topology information

Table 3-1: G^2MPLS Edge Controller external interfaces.

Project: Phosphorus
Deliverable Number: D.2.3
Date of Issue: 31/03/08
EC Contract No.: 034115
Document Code: Phosphorus-WP2-D2.3

3.1.3 Internal interfaces

Peers	Directionality	Main action
G^2.NCC – TNRC	in/out	▪ head-end/tail-end resource configuration (cross-connection or protection among internal labels and “external” labels selected on ingress/egress TNAs) ▪ asynchronous notification of status change
G^2.NCC – G^2.RSVP-TE	in/out	▪ connection setup ▪ connection recovery (particularly restoration)
G^2.NCC – G.UNI RSVP	in/out	▪ G^2MPLS call setup
G^2.NCC – G^2.PCERA	in/out	▪ requests for call explicit routing (single-domain or inter-domain) completion ▪ requests for end-to-end call rerouting (in case of e2e crankback or recovery)
G^2.PCERA – G^2.OSPF-TE (level 0)	in	▪ topology information (single-domain or summarized multi-domain) on Grid and network resources ▪ topology updates
G^2.RSVP-TE – TNRC	in/out	▪ resource configuration (cross-connection or protection among labels) ▪ asynchronous notification of status change
G^2.RSVP-TE – LRM	in/out	▪ resource selection (data-link or label) ▪ local TE-link status update
G^2.RSVP-TE – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.RSVP-TE – G^2.PCERA	in/out	▪ requests for ERO completion ▪ requests for local-to-egress ERO computation (in case of crankback)
G.UNI RSVP – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G.UNI RSVP – TNRC	in/out	▪ resource configuration (cross-connection or protection among labels) ▪ asynchronous notification of status change
LRM – TNRC	in/out	▪ Update lists of data links and labels for bundling purposes ▪ check status of a resource (data-link or label) ▪ asynchronous notification of status change at runtime for bundling update
LRM – SCNGW	out	▪ update bindings between TE-links and Control Channels and between Control Channels and SCN interfaces
G^2.OSPF-TE (level 0) – LRM	in	▪ local TE-link update (all TE information)
G^2.OSPF-TE (level 0) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 0) – G^2.OSPF-TE (level 1)	out	▪ send and keep updated inter-domain topology data (feed-up)
G^2.OSPF-TE (level 1) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 1) – G^2.OSPF-TE (level 0)	out	▪ send and keep updated inter-domain topology data (feed-down)

Table 3-2: G^2MPLS Edge Controller internal interfaces.

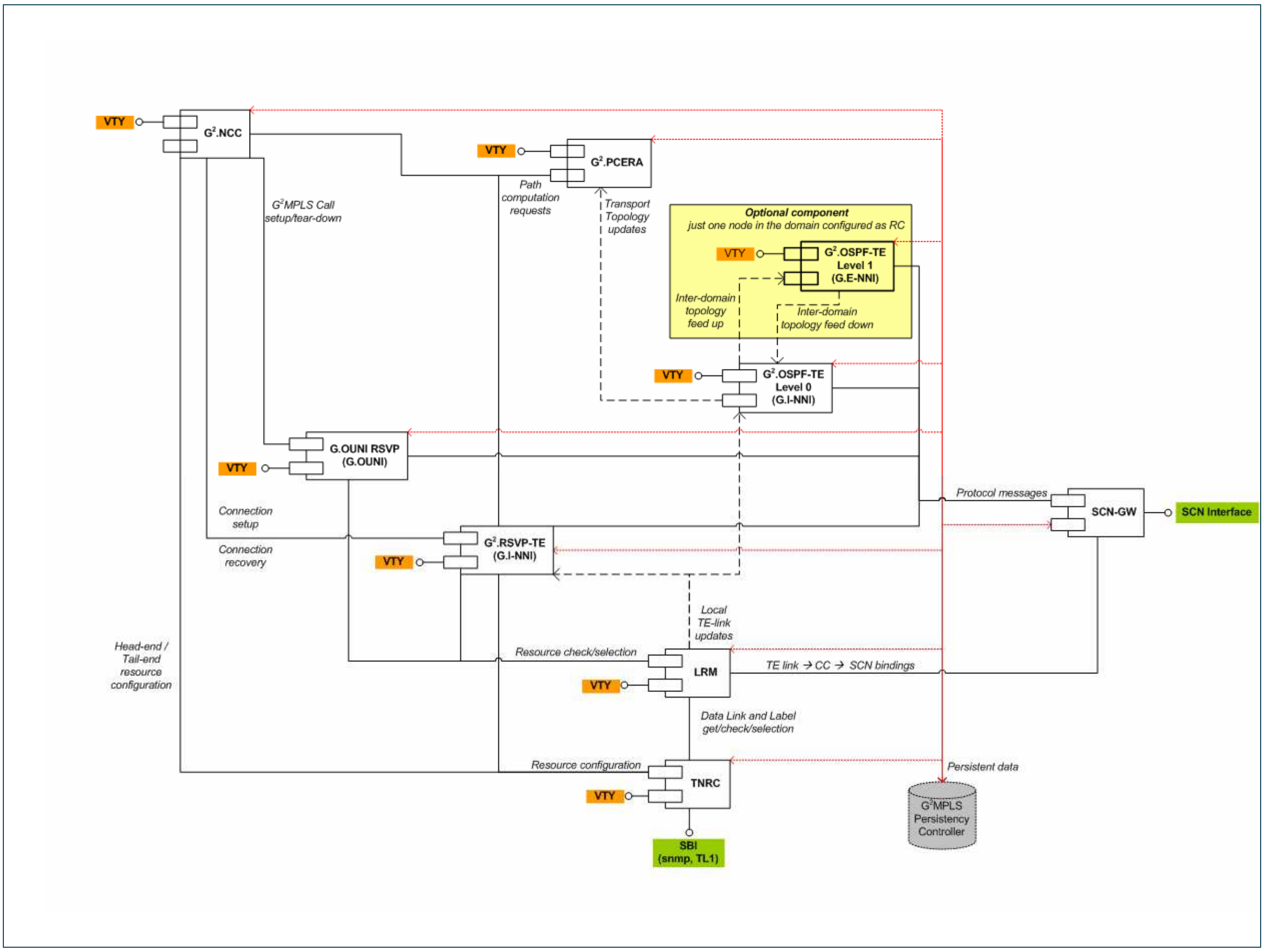

Figure 3-4: Internal components of the G2MPLS Edge Controller.

3.2 G^2MPLS Core Controller

3.2.1 Main functionalities

The G^2MPLS core controller is similar to a GMPLS LSR and, therefore, it is responsible for:

- the progression and control of a G.I-NNI signalling session towards an specified session destination
- the control of the local crankback for a failing LSP
- the flooding of Grid and network routing information, in terms of
  - local TE-link information directly generated
  - remote network and Grid information learned by peer routing controllers
- the completion of sparse or loose Explicit Routes, depending on the available information published by neighbouring G^2MPLS domains
- [optional[4]] the flooding of inter-domain Grid and network routing information, in terms of
  - reception (feed-up) of topology (Grid and network) information from the domain in which it operates (level 0)
  - flooding of routing information with peering inter-domain routing controllers
  - dissemination (feed-down) of the summarized topology information about neighbouring domains towards the base routing instances operating in its domain (level 0).
- the retrieval of information (amount, status and alarms) on the Transport Network resources for G^2MPLS use in the equipment it is attached to
- the configuration (cross-connection) of Transport Network resources in the equipment it is attached to
- the control of the G^2MPLS data model (i.e. TE-links, Data-links, Control channels and SCN interfaces) in accordance with the node configuration and the Transport Network resources availabilities retrieved by the equipment
- [optional] the storage and control of the persistent data (TE-links, Calls, LSPs, resource status, etc.) across case of node restart.

These functionalities are implemented by the software components depicted in Figure 3-5.

3.2.2 External interfaces

Interface	Peer	Directionality	Main action (s)
SBI	TN equipment	in/out	▪ retrieval of information on transport resources (e.g. ports, wavelengths, ▪ configurations on transport resources (e.g. cross-connections, protections,

[4] The functionality is optional because just one node in the domain configured as RC

			etc.) ▪ alarm reporting on configured resources
SCN interface	adjacent G^2MPLS controller	in/out	▪ establish and maintain the adjacency between pairs of G^2MPLS controllers ▪ send/receive protocol SDUs
G.I-NNI	G^2MPLS core controller	in/out	▪ **signalling** o control (setup and recovery) of I-NNI call segments ▪ **routing** o learning of node external Grid and network (single domain and multi-domain) topology resource availabilities
[optional] G.E-NNI	G^2MPLS peering Routing Controllers	in/out	▪ **routing** o publication and learning of inter-domain Grid and network topology information

Table 3-3: G^2MPLS Core Controller external interfaces.

3.2.3 Internal interfaces

Peers	Directionality	Main action
G^2.PCERA – G^2.OSPF-TE (level 0)	in	▪ topology information (single-domain or summarized multi-domain) on Grid and network resources ▪ topology updates
G^2.RSVP-TE – TNRC	in/out	▪ resource configuration (cross-connection or protection among labels) ▪ asynchronous notification of status change
G^2.RSVP-TE – LRM	in/out	▪ resource selection (data-link or label) ▪ local TE-link status update
G^2.RSVP-TE – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.RSVP-TE – G^2.PCERA	in/out	▪ requests for ERO completion ▪ requests for local-to-egress ERO computation (in case of crankback)
LRM – TNRC	in/out	▪ Update lists of data links and labels for bundling purposes ▪ check status of a resource (data-link or label) ▪ asynchronous notification of status change at runtime for bundling update
LRM – SCNGW	out	▪ update bindings between TE-links and Control Channels and between Control Channels and SCN interfaces
G^2.OSPF-TE (level 0) – LRM	in	▪ local TE-link update (all TE information)
G^2.OSPF-TE (level 0) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 0) – G^2.OSPF-TE (level 1)	out	▪ send and keep updated inter-domain topology data (feed-up)

G^2.OSPF-TE (level 1) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 1) – G^2.OSPF-TE (level 0)	out	▪ send and keep updated inter-domain topology data (feed-down)

Table 3-4: G^{2}MPLS Core Controller internal interfaces.

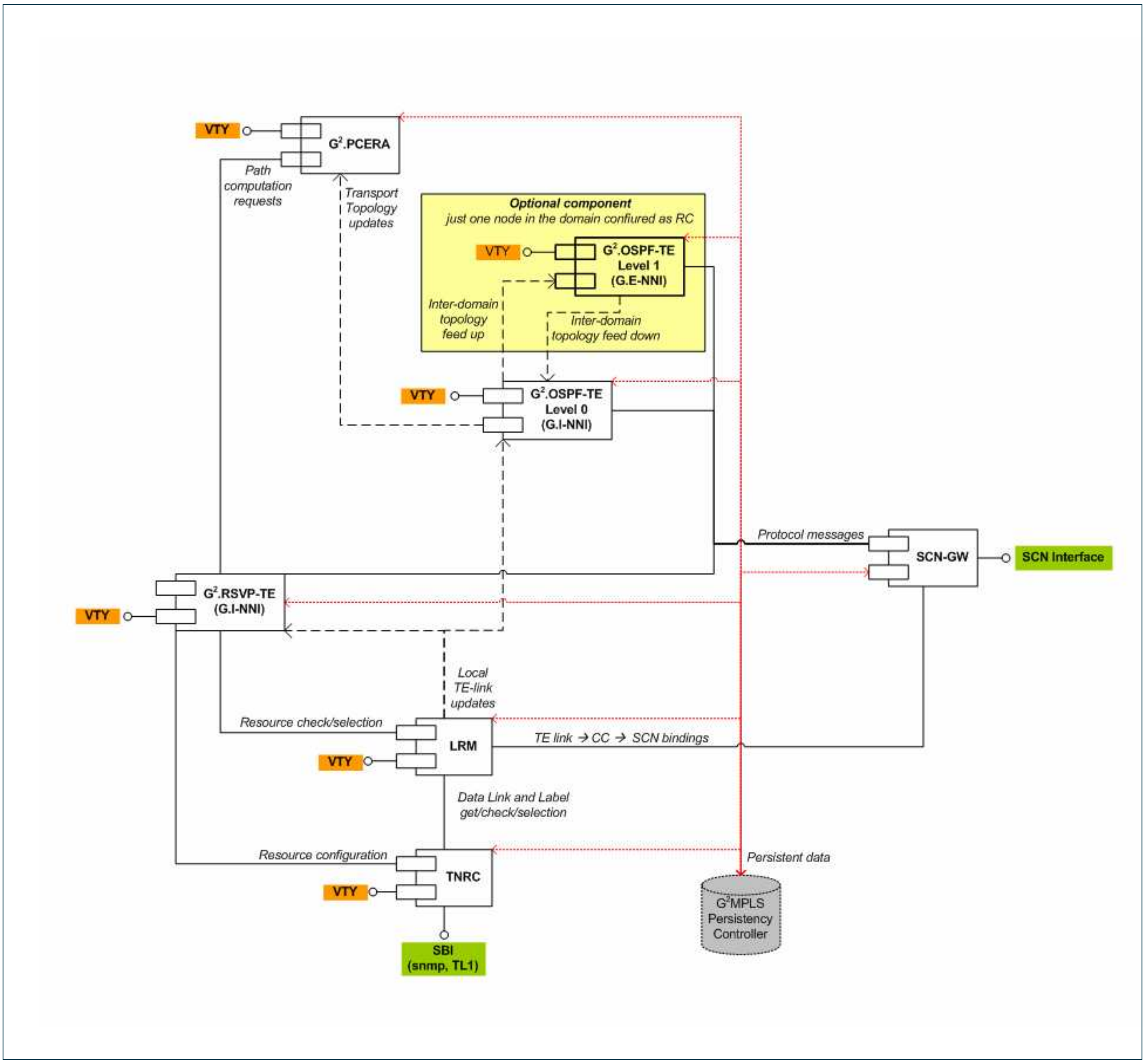

Figure 3-5: Internal components of the G2MPLS Core Controller.

3.3 G^{2}MPLS Border Controller

3.3.1 Main functionalities

The G^{2}MPLS border controller is the egress point of a G^{2}MPLS domain and, therefore, it is responsible for:

- the termination of a G^{2}MPLS call segment incoming through the I-NNI
- the control of the G^{2}MPLS Call setup
- the progression and control of a G.E-NNI signalling session towards an adjacent G^{2}MPLS border controller in another G^{2}MPLS domain
- the flooding of Grid and network routing information, in terms of
  - local TE-link information directly generated
  - remote network and Grid information learned by peer routing controllers
- the completion of sparse or loose Explicit Routes, depending on the available information published by neighbouring G^{2}MPLS domains
- [optional[5]] the flooding of inter-domain Grid and network routing information, in terms of
  - reception (feed-up) of topology (Grid and network) information from the domain in which it operates (level 0)
  - flooding of routing information with peering inter-domain routing controllers
  - dissemination (feed-down) of the summarized topology information about neighbouring domains towards the base routing instances operating in its domain (level 0).
- the retrieval of information (amount, status and alarms) on the Transport Network resources for G^{2}MPLS use in the equipment it is attached to
- the configuration (cross-connection) of Transport Network resources in the equipment it is attached to
- the control of the G^{2}MPLS data model (i.e. TE-links, Data-links, Control channels and SCN interfaces) in accordance with the node configuration and the Transport Network resources availabilities retrieved by the equipment
- [optional] the storage and control of the persistent data (TE-links, Calls, LSPs, resource status, etc.) across case of node restart.

These functionalities are implemented by the software components depicted in Figure 3-6.

3.3.2 External interfaces

Interface	Peer	Directionality	Main action (s)
SBI	TN equipment	in/out	▪ retrieval of information on transport resources (e.g. ports, wavelengths,

[5] The functionality is optional because just one node in the domain configured as RC

			▪ configurations on transport resources (e.g. cross-connections, protections, etc.) ▪ alarm reporting on configured resources
SCN interface	adjacent G^2MPLS controller	in/out	▪ establish and maintain the adjacency between pairs of G^2MPLS controllers ▪ send/receive protocol SDUs
G.I-NNI	G^2MPLS core controller	in/out	▪ **signalling** o control (setup and recovery) of I-NNI call segments ▪ **routing** o learning of node external Grid and network (single domain and multi-domain) topology resource availabilities
G.E-NNI	G^2MPLS peering Routing Controllers	in/out	▪ **signalling** o setup and monitoring of G.E-NNI calls ▪ **[optional] routing** o publication and learning of inter-domain Grid and network topology information

Table 3-5: G^2MPLS Border Controller external interfaces.

3.3.3 Internal interfaces

Peers	Directionality	Main action
G^2.NCC – TNRC	in/out	▪ head-end/tail-end resource configuration (cross-connection or protection among internal labels and “external” labels selected on ingress/egress TNAs) ▪ asynchronous notification of status change
G^2.NCC – G^2.RSVP-TE	in/out	▪ connection setup ▪ connection recovery (particularly restoration)
G^2.NCC – G.ENNI RSVP	in/out	▪ G^2MPLS call setup
G^2.NCC – G^2.PCERA	in/out	▪ requests for call explicit routing (single-domain or inter-domain) completion ▪ requests for end-to-end call rerouting (in case of e2e crankback or recovery)
G^2.PCERA – G^2.OSPF-TE (level 0)	in	▪ topology information (single-domain or summarized multi-domain) on Grid and network resources ▪ topology updates
G^2.RSVP-TE – TNRC	in/out	▪ resource configuration (cross-connection or protection among labels) ▪ asynchronous notification of status change
G^2.RSVP-TE – LRM	in/out	▪ resource selection (data-link or label) ▪ local TE-link status update
G^2.RSVP-TE – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages

G^2.RSVP-TE – G^2.PCERA	in/out	▪ requests for ERO completion ▪ requests for local-to-egress ERO computation (in case of crankback)
G.ENNI RSVP – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G.ENNI RSVP – TNRC	in/out	▪ resource configuration (cross-connection or protection among labels) ▪ asynchronous notification of status change
LRM – TNRC	in/out	▪ Update lists of data links and labels for bundling purposes ▪ check status of a resource (data-link or label) ▪ asynchronous notification of status change at runtime for bundling update
LRM – SCNGW	out	▪ update bindings between TE-links and Control Channels and between Control Channels and SCN interfaces
G^2.OSPF-TE (level 0) – LRM	in	▪ local TE-link update (all TE information)
G^2.OSPF-TE (level 0) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 0) – G^2.OSPF-TE (level 1)	out	▪ send and keep updated inter-domain topology data (feed-up)
G^2.OSPF-TE (level 1) – SCNGW	in/out	▪ send protocol messages ▪ receive protocol messages
G^2.OSPF-TE (level 1) – G^2.OSPF-TE (level 0)	out	▪ send and keep updated inter-domain topology data (feed-down)

Table 3-6: G^2MPLS Border Controller internal interfaces.

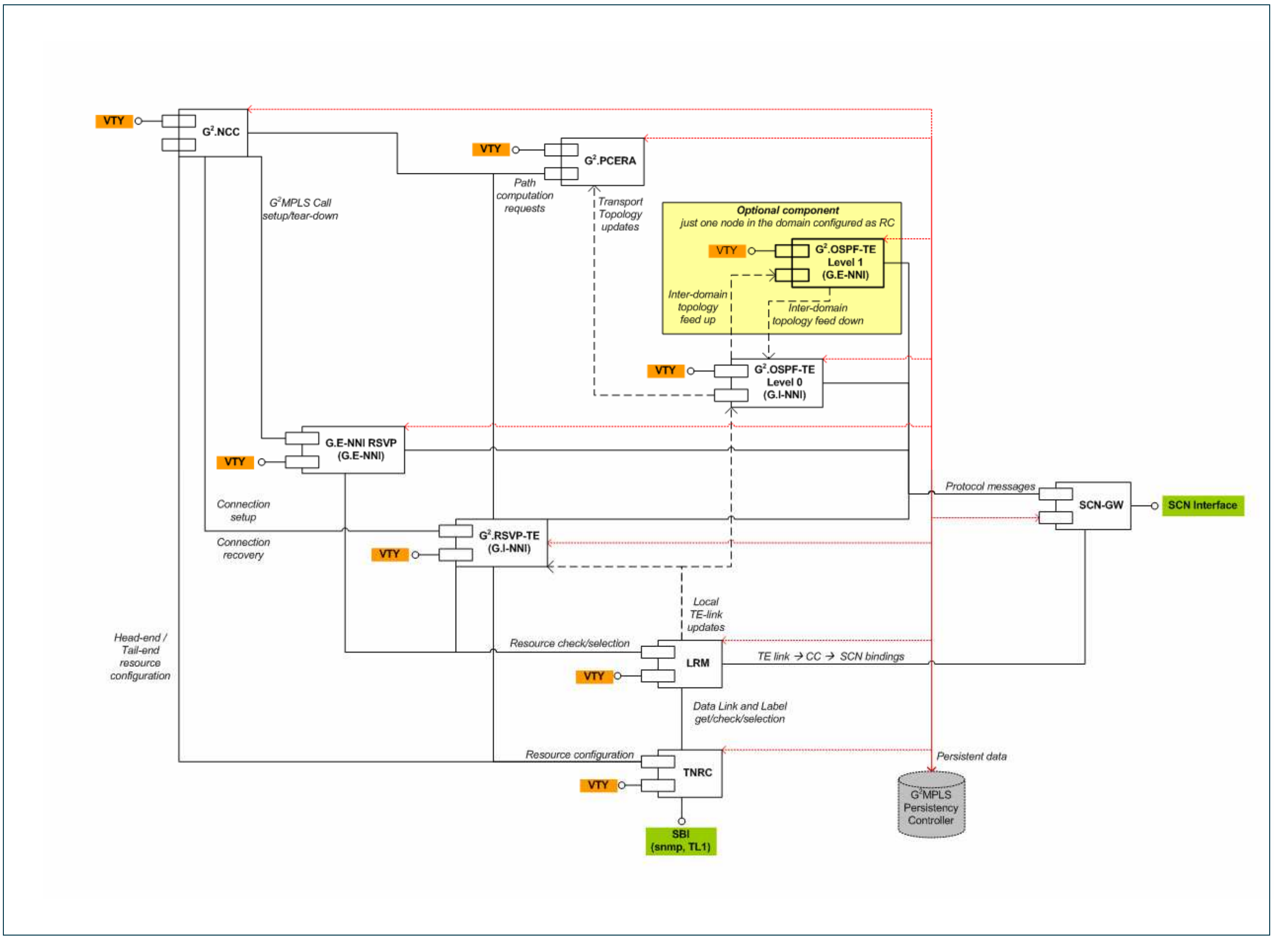

Figure 3-6: Internal components of the G2MPLS Border Controller.

3.4 G.UNI Gateway (G.UNI-GW)

3.4.1 Main functionalities

The G.UNI Gateway is the adapter between the G^2MPLS Control Plane and the Grid middleware. It is responsible for:

- the translation of WS-Agreement semantics on job request (JSDL) and resource availability (GLUE) into G.UNI syntax
- initiating/terminating a GNS transaction and related G^2MPLS calls
- the injection of Grid routing information into the G^2MPLS domain
- the learning and forward to the middleware of remote Grid routing information coming from the G^2MPLS domain
- the configuration (cross-connection) of Transport Network resources in the customer equipment attached to the G^2MPLS domain

These functionalities are implemented by the software components depicted in Figure 3-7.

3.4.2 External interfaces

Interface	Peer	Directionality	Main action (s)
G.UNI	G^2MPLS edge controller	in/out	▪ **signalling** o setup and monitoring of G.UNI calls ▪ **routing** o publication of local Grid resource availabilities o learning of remote Grid resource availabilities by the attached G^2MPLS user
G.NBI	Grid middleware (Grid broker)	in/out	▪ **WS-Agreement job creation** o setup and monitoring of jobs via JSDL ▪ **WS-Agreement resource information** o publication and learning of Grid resource information

Table 3-7: G.UNI Gateway external interfaces.

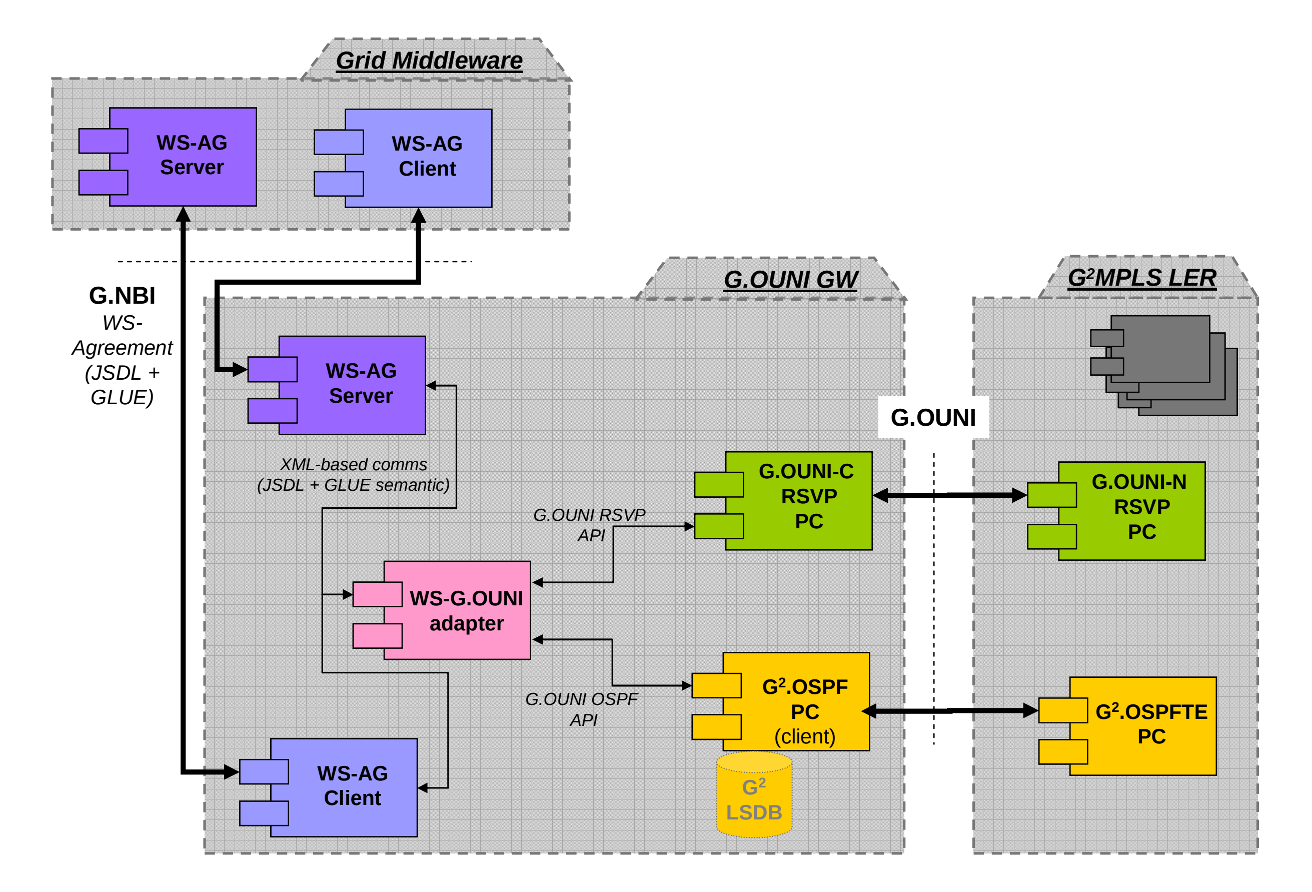

Figure 3-7: G.UNI Gateway (G.UNI-GW) breakdown into main components.

3.4.3 Internal interfaces

Peers	Directionality	Main action
WS-AG server – WS-G.UNI adapter	in/out	▪ translate WS-agreements on Grid job and resource availabilities into an XML schema
WS-G.UNI adapter – G.UNI RSVP	in/out	▪ G^2MPLS call setup
WS-G.UNI adapter – G^2.OSPF (client)	in/out	▪ push/pull Grid topology information

Table 3-8: G.UNI Gateway internal interfaces.

3.5 G.E-NNI Gateway (G.ENNI-GW)

The G.E-NNI GW is designed in the track of integration activities between G²MPLS, NSP/NRPSes and GN2-JRA3 AutoBAHN system. Its design and high-level software specification will be reported in related documents (D2.9).

4 Transport Network Resource Controller (TNRC)

The TNRC module is a separate process, not part of Quagga routing suite and is developed from scratch. It is integrated into Quagga framework according to Quagga daemon main structure (e.g. one master thread to manage the single thread daemon as pseudo multi-thread, the trace log system, the VTY interface, etc).

4.1 TNRC basics

The TNRC module is responsible for abstracting the technology specific details of the transport network resources for control plane use. The main functionalities of the Transport Network Resource Controller are:

- translation and maintenance of the bindings between the technology specific name space for transport resources (e.g. in DWDM equipments: <port, wavelength>; in TDM: <port, virtual container>; in Ethernet: <port, VLANs>) and the G2MPLS name space (<data-link, label>)
- translation between the technology specific configurations for transport resources (e.g. cross-connections, protections, etc.) and the G^2MPLS corresponding actions
- binding maintenance among the resources (e.g. cross-connections, bookings, protections/restorations, etc.).

The TNRC module is further broken down in two sub-modules as described in Table 4-1

module	sub-module	short description
TNRC (Transport Network Resource Controller)	TNRC-AP (TNRC Abstract Part)	Process offering a generic API for the configuration & monitoring of the TN resources. It will abstract the TN resource description, and provide an atomic grouping of actions that might be composed by a set of local management sub-actions on the equipment.

	TNRC-SP (TNRC Specific Part))	Lower part of the process, loaded as plug-in, and offering the upper part an API specific to the equipment considered. It will name resources based on the underlying TN technology and SwCap. The core part of the TNRC-SP is likely to be dependent on the controlled equipment (e.g. based on some proprietary SNMP MIB sub-tree supported for configuration and monitoring).

Table 4-1: TNRC breakdown in sub-modules.

The following sections will describe the TNRC data model, the TNRC Abstract Part and the generic TNRC Specific Part. Examples of TNRC SP design and implementations can be found in Appendix C.

4.2 TNRC data model

The TNRC data model is depicted in Figure 4-1.

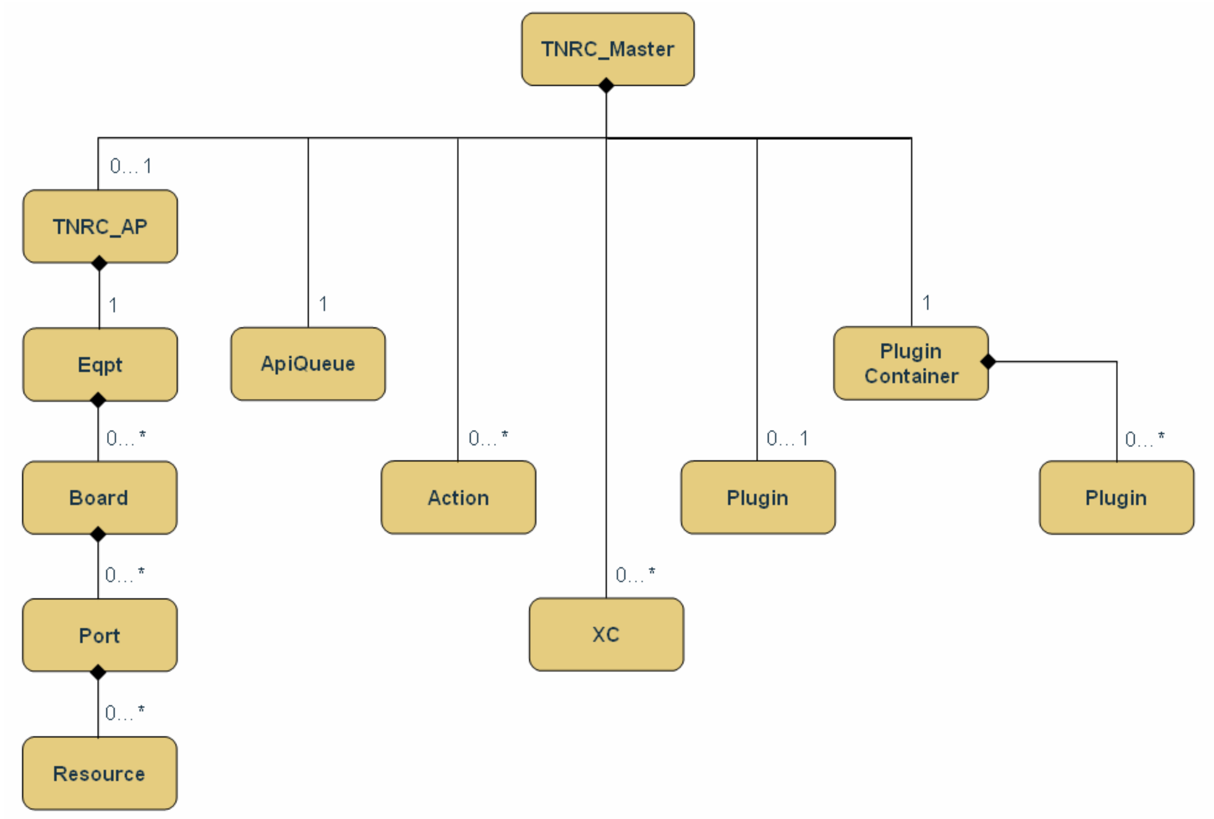

Figure 4-1: TNRC data model

4.2.1 TNRC_Master instance

The TNRC_Master instance is the root of the whole TNRC data model. When TNRC process starts, a global instance of TNRC_Master is created, all available plug-ins (representing all possible TNRC Specific Part) are loaded in a plug-in container, which is linked to the TNRC_Master.

The TNRC_Master instance is also linked to:

- the unique TNRC_AP instance (*tnrc_ap_*)
- the unique ApiQueue instance (*api_aq_)*
- a map of Action instances, representing all the actions either in execution or executed (*actions_*)
- a map of XC instances (*xcs_)*
- the unique Plugin instance installed (*plugin_*)

```
class TNRC_Master {
 public:
       static bool              init(void);
       static bool              destroy(void);

       static TNRC_Master     & instance(void);

       void                     init_vty(void);

       void                     pc(Pcontainer * PC);
       Pcontainer             * getPC();

       struct thread_master * getMaster();
       static TNRC::TNRC_AP * getTNRC();

       bool                     test_mode(void);
       void                     test_mode(bool val);
       char*                    test_file(void);
       void                     test_file(std::string loc);

       tnrcap_cookie_t          new_cookie();
       uint32_t                 new_xc_id();

       eqpt_type_t              getEqpt_type(eqpt_id_t id);

       static bool              attach_instance(TNRC::TNRC_AP * t);
       static bool              detach_instance(TNRC::TNRC_AP * t);

       Plugin*                  getPlugin(void);
       void                     installPlugin(Plugin * p);
       std::string              plugin_location(void);
       void                     plugin_location(std::string loc);

       bool                     api_queue_insert(api_queue_element_t * e);
       api_queue_element_t    * api_queue_extract(void);
       int                      api_queue_size(void);
       void                     api_queue_process(void);
       u_int                    api_queue_tot_request(void);

       void                     process_make_xc(api_queue_element_t * el);
       void                     process_destroy_xc(api_queue_element_t * el);
       void                     process_reserve_xc(api_queue_element_t * el);
       void                     process_unreserve_xc(api_queue_element_t * el);

       bool                     attach_action(tnrcap_cookie_t ck, Action * a);
       bool                     detach_action(tnrcap_cookie_t ck);
       Action                 * getAction (tnrcap_cookie_t ck);

       bool                     attach_xc(u_int id, XC * xc);
       bool                     detach_xc(u_int id);
       XC                     * getXC (u_int id);
       int                      n_xcs (void);

       // define iterator_actions
       DEFINE_MASTER_MAP_ITERATOR (actions, tnrcap_cookie_t, Action);
       // define iterator_xcs
       DEFINE_MASTER_MAP_ITERATOR (xcs, u_int, XC);

 protected:
```

```
        TNRC_Master& operator=(const TNRC_Master& j);
        TNRC_Master(const TNRC_Master & j);
        TNRC_Master(void);
        ~TNRC_Master(void);

 private:
        static TNRC_Master                  * instance_;

        static tnrcap_cookie_t                cookie_; //value for the next cookie
        static uint32_t                       xc_id_;  //value for the next Xc id

        static struct thread_master         * master_;
        //test mode
        bool                                  test_mode_;
        std::string                           test_file_;

        TNRC::TNRC_AP *                       tnrc_ap_; // TNRC_AP instance

        Pcontainer *                          PC_;    // pointer to Plugin container
        Plugin                              * plugin_;// pointer to installed Plugin
        std::string                           plugin_location_;

        ApiQueue                              api_aq_;
        std::map<tnrcap_cookie_t, Action *>   actions_; // actions in
                                                       // execution/executed

        std::map<u_int, XC *>                 xcs_; // XCs active or reserved

        static time_t                         start_time_;
};
```

Code 4-1: TNRC_Master class.

4.2.2 TNRC_AP instance

The TNRC_AP instance is the container of the TNRC abstraction of the Transport Network resources. It manages Eqpt, Board, Port and Resource instances offering an up-to-date image of the equipment resources status.

The most relevant fields are:

- a flag active when the link with equipment is down (*eqpt_link_down_*): in this case no further actions on the equipment can be accepted until restoring communication with equipment (in charge of TNRC Specific Part)
- a map of linked Eqpt instances (*eqpts_*). Even if only one Eqpt instance is accepted, there is a map to take in account of future upgradings

```
class TNRC_AP {
 public:
        TNRC_AP(void);
```

```
        ~TNRC_AP(void);

        bool       attach(EqptKey_t k, Eqpt * e);
        bool       detach(EqptKey_t k);

        Eqpt     * getEqpt(EqptKey_t    k);
        Board    * getBoard(EqptKey_t  e_id,
                            BoardKey_t b_id);
        Port     * getPort(EqptKey_t  e_id,
                           BoardKey_t b_id,
                         PortKey_t  p_id);
        Resource * getResource(EqptKey_t     e_id,
                               BoardKey_t    b_id,
                               PortKey_t     p_id,
                               ResourceKey_t l_id);

        int        n_eqpts(void);

        bool       eqpt_link_down(void);
        void       eqpt_link_down(bool val);

        // Defines eqpts_iterator
        DEFINE_DM_MAP_ITERATOR(eqpts, Eqpt);

 private:
        uint32_t dflt_RetransmitInterval_;
        time_t   tnrc_start_time_;
        time_t   current_time_;
        time_t   stats_reset_time_;
        time_t   shutdown_delay_;

        bool     eqpt_link_down_; // flag active if equipment link is down

        std::map<EqptKey_t, Eqpt *> eqpts_; // map of eqpts
};
```

Code 4-2: TNRC_AP class.

4.2.3 Eqpt instance

The Eqpt instance is the highest level of abstraction of the equipment resources, representing the equipment itself. There is only one Eqpt instance linked to the TNRC_AP instance.

The most relevant fields are:

- a unique identifier (*eqpt_id_)*
- the type of equipment (e.g. ADVA, Calient, etc.) (*type_*)
- operational state (*opstate_*)
- administrative state (*admstate_*)
- a map of linked Board instances (*boards_*)

```
class Eqpt {
 public:
       Eqpt(void);
       ~Eqpt(void);
       Eqpt(TNRC_AP *      tnrc,
            eqpt_id_t      id,
            g2mpls_addr_t  addr,
            eqpt_type_t    t,
            opstate_t      opst,
            admstate_t     admst,
            std::string    loc);

       bool             attach(BoardKey_t k, Board * b);
       bool             detach(BoardKey_t k);
       Board          * getBoard(BoardKey_t k);
       int              n_boards(void);


       eqpt_id_t        eqpt_id(void);
       g2mpls_addr_t    address(void);
       eqpt_type_t      type(void);

       opstate_t        opstate(void);
       void             opstate(opstate_t st);
       admstate_t       admstate(void);
       void             admstate(admstate_t st);

       const char     * location(void);

       // Defines boards_iterator
       DEFINE_DM_MAP_ITERATOR(boards, Board);

 private:
       TNRC_AP        * tnrc_ap_;              // TNRC_AP parent instance

       g2mpls_addr_t    address_;
       eqpt_id_t        eqpt_id_;
       eqpt_type_t      type_;                 // type of equipment

       opstate_t        opstate_;
       admstate_t       admstate_;

       std::string      location_name_;

       std::map<BoardKey_t, Board *> boards_; // map of boards
};
```

Code 4-3: Eqpt class.

4.2.4 Board instance

The most relevant fields of the Board instance are:

- a unique board identifier for a given Eqpt (*board_id_*)
- switching capability of all Ports and Resources linked (*sw_cap_*)

- encoding type of all Ports and Resources linked (*enc_type_*)
- operational state (*opstate_*)
- administrative state (*admstate_*)
- a map of linked Port instances (*ports_*)

```
class Board {
 public:
       Board(void);
       ~Board(void);
       Board(Eqpt *     e,
             board_id_t id,
             sw_cap_t   sw_cap,
             enc_type_t enc_type,
             opstate_t  opst,
             admstate_t admst);

       Eqpt       * eqpt();

       bool         attach(PortKey_t k, Port * p);
       bool         detach(PortKey_t k);
       Port*        getPort (PortKey_t k);
       int          n_ports(void);

       board_id_t   board_id(void);

       sw_cap_t     sw_cap(void);
       enc_type_t   enc_type(void);

       opstate_t    opstate(void);
       void         opstate(opstate_t st);
       admstate_t   admstate(void);
       void         admstate(admstate_t st);

       // Defines ports_iterator
       DEFINE_DM_MAP_ITERATOR(ports, Port);

 private:
       Eqpt       * eqpt_;      // eqpt parent instance
       board_id_t   board_id_;

       sw_cap_t     sw_cap_;    // switching capability
       enc_type_t   enc_type_;  // encoding type

       opstate_t    opstate_;
       admstate_t   admstate_;

       std::map<PortKey_t, Port *> ports_; // map of ports
};
```

Code 4-4: Board class.

4.2.5 Port instance

The Port instance is the TNRC abstraction for the data link in the G^2MPLS space. The triple Eqpt/Board/Port in fact identifies a single data link on the equipment.

The most relevant fields are:

- a unique port identifier for a given Board (*port_id_*)
- the protection type for this data link (*prot_type_*)
- operational state (*opstate_*)
- administrative state (*admstate_*)
- total available bandwidth on the data link (*max_bw_*)
- maximum reservable bandwidth on the data link (*max_res_bw_*)
- unreserved bandwidth per priority on the data link (*unres_bw_*). This parameter is updated each time a linked Resource is involved in some action on the equipment.
- minimum reservable bandwidth per LSP on the data link (*min_lsp_bw_*)
- maximum reservable bandiwdth per LSP and per priority on the data link (*max_lsp_bw_*)
- a map of linked Resource instances (*resources_*)

```
class Port {
 public:
      Port(void);
      ~Port(void);
      Port(Board *           b,
           port_id_t         id,
           int               flags,
           g2mpls_addr_t     rem_eq_addr,
           port_id_t         rem_port_id,
           opstate_t         opst,
           admstate_t        admst,
           uint32_t          bandwidth,
           gmpls_prottype_t protection);

      Board              * board();

      bool                 attach(ResourceKey_t k, Resource * r);
      bool                 detach(ResourceKey_t k);
      int                  n_resources (void);

      Resource           * getResource(ResourceKey_t k);

      port_id_t            port_id (void);
      int                  port_flags (void);
      g2mpls_addr_t        remote_eqpt_address (void);
      port_id_t            remote_port_id (void);

      opstate_t            opstate (void);
      void                 opstate(opstate_t st);
      admstate_t           admstate(void);
      void                 admstate(admstate_t st);
```

```
      uint32_t            max_bw(void);
      uint32_t            max_res_bw(void);
      avail_bw_per_prio_t unres_bw(void);
      uint32_t            min_lsp_bw(void);
      avail_bw_per_prio_t max_lsp_bw(void);
      void                upd_unres_bw(label_t label);

      gmpls_prottype_t    prot_type(void);

      // Defines resources_iterator
      DEFINE_DM_MAP_COMP_ITERATOR(resources, Resource, myCompareResource);

 private:
      Board             * board_;         // parent board instance

      port_id_t           port_id_;
      int                 port_flags_;   // bit mask describing the port behaviour
      g2mpls_addr_t       remote_eqpt_address_;
      port_id_t           remote_port_id_;

      opstate_t           opstate_;
      admstate_t          admstate_;

      gmpls_prottype_t    prot_type_;   // Protection type


      uint32_t            max_bw_;      // total available bandwidth
      uint32_t            max_res_bw_;  // max reservable bandwidth
      avail_bw_per_prio_t unres_bw_;    // unreserved  bandwidth per priority
      uint32_t            min_lsp_bw_;  // minimum reservable bandiwdth
      avail_bw_per_prio_t max_lsp_bw_;  // maximum reservable bandiwdth per priority

      // map of resources
      std::map<ResourceKey_t, Resource *, myCompareResource> resources_;
};
```

Code 4-5: Port class.

4.2.6 Resource instance

The Resource instance is the lowest level of abstraction of the equipment resources, representing a single label associated to a data link.

The most relevant fields are:

- a unique label identifier for a given Port (*label_id_*)
- operational state (*opstate_*)
- administrative state (*admstate_*)
- label state (*state_*)
- a map of advance reservation for this label (*reservations_*), stored as [start time, end time] couples

```
class Resource {
 public:
       Resource(void);
       ~Resource(void);
       Resource(Port *        p,
                int           tp_fl,
                label_t       id,
                opstate_t     opst,
                admstate_t    admst,
                label_state_t st);

       bool            attach(struct timeval start, struct timeval end);
       bool            detach(struct timeval start);
       bool            check_label_availability(struct timeval start,
                                                struct timeval end);

       Port          * port();
       int             tp_flags(void);
       label_t         label_id(void);

       opstate_t       opstate(void);
       void            opstate(opstate_t st);
       admstate_t      admstate(void);
       void            admstate(admstate_t st);
       label_state_t   state(void);
       void            state(label_state_t st);

       //define iterator advance reservations
       typedef std::map<struct timeval,  struct timeval, myCompareTime>::
             iterator iterator_reservations;

       iterator_reservations begin_reservations(void)
       iterator_reservations end_reservations(void);

 private:
       Port          * port_;           // parent port pointer

       int             tp_flags_;
       label_t         label_id_;

       opstate_t       opstate_;
       admstate_t      admstate_;
       label_state_t   state_;

       // Advance Reservation Calendar
       std::map<struct timeval, struct timeval, myCompareTime> reservations_;
};
```

Code 4-6: Resource class.

4.2.7 Plugin Container (Pcontainer) instance

The Plugin Container (Pcontainer) instance is unique and is created at the boot of the TNRC process. It is a container of all available plug-ins, that are loaded in the Plugin Container at the boot.

```
class Pcontainer {
 public:
      Pcontainer(void) {};
      ~Pcontainer(void){};

      bool     attach(std::string name, Plugin *p);
      bool     detach(std::string name);
      Plugin * getPlugin(std::string name);

      // Defines iterator_plugins
      DEFINE_PIN_MAP_ITERATOR(plugins, std::string, Plugin);

 private:
      std::map<std::string, Plugin *> plugins_; // map of plugins
};
```

Code 4-7: Pcontainer class.

4.2.8 Plugin instance

The Plugin class is the abstract interface of the TNRC Specific Part. Each TNRC Specific Part for for a given equipment type (ADVA, Calient, etc.) implements his own interface, inheriting the following Plugin class.

One and only one plug-in can be installed in the TNRC_Master instance, in fact one TNRC process manges one and only one equipment.

The most relevant fields are:

- a unique plug-in name (*name_*)
- a flag for bidirectional cross-connections support (*xc_bidir_support_*)

```
class Plugin {
public:
      Plugin(void) {};
      ~Plugin(void){};
      Plugin(std::string name);

      std::string            name(void);
      tnrcsp_handle_t        new_handle(void);

      bool                   xc_bidir_support(void);
```

```
        virtual wq_item_status     wq_function(void *d) = 0;
        virtual void               del_item_data(void *d) = 0;
        virtual tnrcapiErrorCode_t probe(std::string location) = 0;

        virtual tnrcsp_result_t
                tnrcsp_make_xc(tnrcsp_handle_t *          handlep,
                               tnrc_port_id_t             portid_in,
                               label_t                    labelid_in,
                               tnrc_port_id_t             portid_out,
                               label_t                    labelid_out,
                               xcdirection_t              direction,
                               tnrc_boolean_t             isvirtual,
                               tnrc_boolean_t             activate,
                               tnrcsp_response_cb_t       response_cb,
                               void *                     response_ctxt,
                               tnrcsp_notification_cb_t   async_cb,
                               void *                     async_ctxt) = 0;

        virtual tnrcsp_result_t
                tnrcsp_destroy_xc(tnrcsp_handle_t *    handlep,
                                  tnrc_port_id_t       portid_in,
                                  label_t              labelid_in,
                                  tnrc_port_id_t       portid_out,
                                  label_t              labelid_out,
                                  xcdirection_t        direction,
                                  tnrc_boolean_t       isvirtual,
                                  tnrc_boolean_t       deactivate,
                                  tnrcsp_response_cb_t response_cb,
                                  void *               response_ctxt) = 0;

        virtual tnrcsp_result_t
                tnrcsp_reserve_xc(tnrcsp_handle_t *    handlep,
                                  tnrc_port_id_t       portid_in,
                                  label_t              labelid_in,
                                  tnrc_port_id_t       portid_out,
                                  label_t              labelid_out,
                                  xcdirection_t        direction,
                                  tnrcsp_response_cb_t response_cb,
                                  void *               response_ctxt) = 0;

        virtual tnrcsp_result_t
                tnrcsp_unreserve_xc(tnrcsp_handle_t *    handlep,
                                    tnrc_port_id_t       portid_in,
                                    label_t              labelid_in,
                                    tnrc_port_id_t       portid_out,
                                    label_t              labelid_out,
                                    xcdirection_t        direction,
                                    tnrcsp_response_cb_t response_cb,
                                    void *               response_ctxt) = 0;

        virtual tnrcsp_result_t
                tnrcsp_register_async_cb(tnrcsp_event_t *events) = 0;

protected:
        std::string         name_;
        tnrcsp_handle_t     handle_;
        bool                xc_bidir_support_;

        struct work_queue * wqueue_;
```

```
};
```

Code 4-8: Plugin class.

The pure virtual methods in the Plugin class are the TNRC Specific Part API exposed toward TNRC Abstract Part to communicate with the equipment. These methods, implemented by each TNRC Specific Part plug-in, are:

- *tnrcsp_make_xc()*: create (or activate a reserved) cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks have been carried out: poisitively if the cross-connection has been requested and started o the equipment, else negatively
  - later, when the cross-connection has been completed, the TNRC Specific Part will come back using the response callback (*response_cb*) and context (*response_ctxt*),and delivering the result of the operation
  - any future event related to the cross-connection or one of its component will be reported with the asynchronus callback (*async_cb*)
- *tnrcsp_destroy_xc()*: destroy an existent cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks have been carried out: poisitively if the cross-connection removal has been requested and started o the equipment, else negatively
  - later, when the cross-connection removal has been completed, the TNRC Specific Part will come back using the response callback (*response_cb*) and context (*response_ctxt*),and delivering the result of the operation
- *tnrcsp_reserve_xc()*: reserve a cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks have been carried out: poisitively if the cross-connection reservation has been requested and started o the equipment, else negatively
  - later, when the cross-connection reservation has been completed, the TNRC Specific Part will come back using the response callback (*response_cb*) and context (*response_ctxt*),and delivering the result of the operation
- *tnrcsp_unreserve_xc()*: unreserve an existent reserved cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks have been carried out: poisitively if the cross-connection unreservation has been requested and started o the equipment, else negatively
  - later, when the cross-connection unreservation has been completed, the TNRC Specific Part will come back using the response callback (*response_cb*) and context (*response_ctxt*),and delivering the result of the operation
- *tnrcsp_register_async_cb()*: report about events on ports; it's invoked asynchronously by the TNRC Specific Part, based on underlying event report mechanism

4.2.9 ApiQueue instance

The ApiQueue instance is unique and is created at the boot of the TNRC process. It contains a queue in which are stored all the action requests coming from the upper layers, through the TNRC Abstract Part external API.

Each time an action is executed (either successfully or unsuccessfully), an action request is extracted from the queue and executed.

```
class ApiQueue {
 public:
       ApiQueue(void);
       ~ApiQueue(void) {};

       bool                    insert(api_queue_element_t * e);
       api_queue_element_t *   extract(void);

       int                     size(void);
       u_int                   tot_request (void);

 private:
       u_int                   tot_req_;   // total number of action requests

       std::queue<api_queue_element_t *> queue_;   // queue of actions to execute
};
```

Code 4-9: ApiQueue class.

4.2.10 Action instance

The Action instance is the basic item in the TNRC data model dedicated to equipment resource requests management. An Action instance is created each time that a “create” (make/reserve cross-connection) action request is extracted from ApiQueue and ready to communicate with the equipment. Otherwise when a “destroy” action (destroy/unreserve cross-connection) is extracted from ApiQueue, no new Action instance is created, and correspondent make/reserve Action instance is retrieved to post an event on its FSM instance.

It has the following relevant fields:

- a unique identifier generated for the client requested the action (*ap_cookie_*)
- a unique identifier generated by TNRC Specific Part (*sp_handle_*)
- type of action (make/destroy or reserve/unreserve cross-connection) (*action_type_*)
- a pointer to the (unique) installed plug-in (*plugin_*)
- a pointer to the Action FSM instance for this action (*FSM_*)
- atomic action in execution (*atomic_*)
- queue containing all the atomic actions for this action *(atomic_actions_*)
- queue containing the atomic actions to do (*atomic_todo_*)

- queue containing the atomic actions already done (*atomic_done_*)

```
class Action {
public:
	Action (void) {};
	~Action (void){};

	Plugin * plugin();

	Action * atomic();
	void     atomic(Action * at);

	tnrcap_cookie_t ap_cookie(void);

	void            sp_handle(tnrcsp_handle_t h);
	tnrcsp_handle_t sp_handle(void);

	long   resp_ctxt(void);
	void   resp_ctxt(long ctxt);

	long   async_ctxt(void);

	tnrc_action_type_t action_type(void);
	void               action_type(tnrc_action_type_t type);

	void              prel_check(tnrcsp_result_t pc);
	tnrcsp_result_t   prel_check(void);

	void            eqpt_resp(tnrcsp_result_t res);
	tnrcsp_result_t eqpt_resp(void);

	void have_atomic(bool atomic);
	bool have_atomic(void);

	bool have_atomic_todo(void);
	bool have_atomic_todestroy(void);

	bool wait_answer(void);
	void wait_answer(bool val);

	//atomic actions to do management
	void     pop_todo();
	Action * front_todo(void);
	void     push_todo(Action * at);
	int      todo_size(void);

	//atomic actions done management
	void     pop_done();
	Action * front_done(void);
	void     push_done(Action * at);
	int      done_size(void);
	void     swap_action_type(void);

	int  n_retry(void);
	void n_retry_inc();

	virtual void fsm_start(void) = 0;
```

```
        virtual void fsm_post(fsm::TNRC::virtFsm::root_events_t  ev,
                              void *                             ctxt,
                              bool                               queue = false) = 0;

        //define iterator_atomic_actions
        DEFINE_QUEUE_ITERATOR(atomic_actions, Action);
        //define iterator_atomic_done
        DEFINE_QUEUE_ITERATOR(atomic_done, Action);

protected:
        tnrcap_cookie_t             ap_cookie_;
        tnrcsp_handle_t             sp_handle_;

        long                        resp_ctxt_;
        long                        async_ctxt_;

        tnrc_action_type_t          action_type_;

        tnrcsp_result_t             prel_check_;

        tnrcsp_result_t             eqpt_resp_;

        bool                        have_atomic_;
        bool                        have_atomic_todo_;

        bool                        wait_answer_;
        int                         n_retry_;

        Plugin                    * plugin_;

fsm::TNRC::virtFsm *        FSM_;           // Action FSM instance

        Action                    * atomic_;        // atomic action in execution

        std::deque<Action *>        atomic_actions_; // queue of atomic actions
        std::deque<Action *>        atomic_todo_;    // queue of atomic actions todo
        std::deque<Action *>        atomic_done_;    // queue of atomic actions done
};
```

Code 4-10: Action class.

4.2.11 XC instance

An XC instance is created each a time a make or reserve cross-connection action is executed successfully. The XC instance is useful to correlate an executed action with the correspondent cross-connection on the equipment, allowing to manage easily any possible notification related to the cross-connection from the equipment.

An XC instance is deleted when correspondent cross-connection is either destroyed or unreserved on the equipment.

It has the following relevant fields:

- a unique cross-connection identifier (*id_*)
- identifier of the associated action (*cookie_*)
- resources (and ports) involved in the cross-connection (*portid_in_*, *portid_out_*, *labelid_in_*, *labelid_out_*)

```
class XC {
 public:
      XC(void) {};
      ~XC(void){};
      XC(u_int                     id,
         tnrcap_cookie_t           ck,
         tnrcap_xc_state_t         st,
         tnrc_port_id_t            portid_in,
         label_t                   labelid_in,
         tnrc_port_id_t            portid_out,
         label_t                   labelid_out,
         xcdirection_t             direction,
         long                      ctxt);

      u_int             id(void);

      tnrcap_cookie_t   cookie(void);
      void              cookie(tnrcap_cookie_t ck);

      tnrcap_xc_state_t state(void);
      void              state(tnrcap_xc_state_t st);

      tnrc_port_id_t    portid_in(void);
      tnrc_port_id_t    portid_out(void);
      label_t           labelid_in(void);
      label_t           labelid_out(void);
      xcdirection_t     direction(void);

      long              async_ctxt (void);
      void              async_ctxt (long ctxt);

 private:
      u_int             id_;          // id of the cross-connection
      tnrcap_cookie_t   cookie_;      // cookie of the associated action

      tnrcap_xc_state_t state_;       // state of the crossconnection

      tnrc_port_id_t    portid_in_;
      tnrc_port_id_t    portid_out_;
      label_t           labelid_in_;
      label_t           labelid_out_;
      xcdirection_t     direction_;

      long              async_ctxt_;
};
```

Code 4-11: XC class.

4.3 TNRC Abstract Part

The TNRC Abstract Part is the core of the TNRC module; it is implemented as a process integrated in the Quagga framework, and it is in charge of:

- bridging the semantics from the G^2MPLS space down to the equipment (through TNRC_SP)
    - G^2MPLS resource spec:
        - data link
        - label
    - lower layer resource spec (at TNRC_SP)
        - FSC
            - port
        - LSC
            - port
            - wavelength/waveband
        - TDM
            - port
            - Termination Point (TP)
        - L2SC
            - port
            - label
- decoupling the communication mechanism
- decomposing and serializing the operations that are atomic at the G^2MPLS level into a sequence of operations that are atomic at the equipment level
- maintaining a synchronized image of equipment resource status
- providing access to this mirrored information to upper G^2MPLS module
- handling the notifications rising from the equipment and correlating them to some G^2MPLS-level resource

The TNRC Abstract Part has three different APIs:

- configuration API (exposed to TNRC Specific Part)
- external API (exposed to external modules)
- action specific API (exposed to TNRC Specific Part)

4.3.1 TNRC Abstract Part configuration API

The configuration API is exposed to TNRC Specific Part, and is used to install the unique plug-in and to build an up-to-date image of the equipment .in TNRC Abstract Part. It is specified in *<sw_root>/tnrcd/tnrc_apis.h*.

```
tnrcapiErrorCode_t init_plugin(std::string name, std::string loc);

int                plugin_probe(struct thread *t);

tnrcapiErrorCode_t add_Eqpt(eqpt_id_t     id,
                            g2mpls_addr_t address,
                            eqpt_type_t   type,
                            opstate_t     opst,
                            admstate_t    admst,
                            std::string   location);

tnrcapiErrorCode_t add_Board(eqpt_id_t  eqpt_id,
                                   board_id_t id,
                                   sw_cap_t   sw_cap,
                                   enc_type_t enc_type,
                                   opstate_t  opst,
                                   admstate_t admst);

tnrcapiErrorCode_t add_Port(eqpt_id_t        eqpt_id,
                            board_id_t        board_id,
                            port_id_t         id,
                            int               flags,
                            g2mpls_addr_t     rem_eq_addr,
                            port_id_t         rem_port_id,
                            opstate_t         opst,
                            admstate_t        admst,
                            uint32_t          bandwidth,
                            gmpls_prottype_t protection);

tnrcapiErrorCode_t add_Resource(eqpt_id_t     eqpt_id,
                                 board_id_t    board_id,
                                 port_id_t     port_id,
                                 int           tp_fl,
                                 label_t       id,
                                 opstate_t     opst,
                                 admstate_t    admst,
                                 label_state_t st);
```

Code 4-12: TNRC Abstract Part configuration API.

The methods of the API are:

- *init_plugin()*: install the plug-in specified by *name* into TNRC_Master instance, and schedule the execution of *plugin_probe()*
- *plugin_probe()*: this is the core method of the configuration API. It's a wrapper of TNRC Specific Part plug-in method called *probe()* (see Code 4-8), that is responsible to create the image of equipment in the TNRC_AP instance through.the *add_Eqpt(),add_Board(),add_Port(),add_Resource()* methods
- *add_Eqpt()*: add an Eqpt instance in the data model
- *add_Board()*: add a Board instance in the data model
- *add_Port()*: add a Port instance in the data model
- *add_Resource()*: add a Resource instance in the data model

4.3.2 TNRC Abstract Part external API

The external API is exposed to external modules, and is used to accept new action requests and to provide access to the image of the equipment stored in the data model. It is specified in *<sw_root>/idl/tnrc.idl*.

```
#include "types.idl"
#include "g2mplsTypes.idl"

interface TNRC {
      exception InternalProblems { };
      exception CannotFetch      { };
      exception ParamError       { };

      boolean makeXC(out Types::uint32             cookie,
                     in  g2mplsTypes::DLinkId      dlinkIn,
                     in  g2mplsTypes::labelId      labelIn,
                     in  g2mplsTypes::DLinkId      dlinkOut,
                     in  g2mplsTypes::labelId      labelOut,
                     in  g2mplsTypes::xcDirection  direction,
                     in  Types::uint32             activate,
                     in  Types::uint32             rsrvCookie,
                     in  long                      responseCtxt,
                     in  long                      asyncCtxt)
            raises(InternalProblems, ParamError);

      boolean destroyXC(in Types::uint32  cookie,
                        in Types::uint32  deactivate,
                        in long           responseCtxt)
            raises(InternalProblems, ParamError);

      boolean reserveXC(out Types::uint32             cookie,
                        in  g2mplsTypes::DLinkId      dlinkIn,
                        in  g2mplsTypes::labelId      labelIn,
                        in  g2mplsTypes::DLinkId      dlinkOut,
                        in  g2mplsTypes::labelId      labelOut,
                        in  g2mplsTypes::xcDirection  direction,
                        in  Types::uint32             advanceRsrv,
                        in  long                      startTime,
                        in  long                      endTime,
                        in  long                      responseCtxt)
            raises(InternalProblems, ParamError);

      boolean unreserveXC(in Types::uint32  cookie,
                          in long           responseCtxt)
            raises(InternalProblems, ParamError);

      boolean getDLinkDetails(in  g2mplsTypes::DLinkId         dataLink,
                              out g2mplsTypes::DLinkParameters params)
            raises(InternalProblems, CannotFetch);

      boolean getLabelStatus(in  g2mplsTypes::DLinkId    localDataLink,
                             in  g2mplsTypes::labelId    label,
                             out g2mplsTypes::labelState labelState,
                             out g2mplsTypes::operState  opState)
            raises(InternalProblems, CannotFetch);

      boolean getLabelFromDLink(in  g2mplsTypes::DLinkId dataLink,
```

```
                                    out g2mplsTypes::labelId label)
             raises(InternalProblems, CannotFetch);
};
```

Code 4-13: TNRC Abstract Part external API IDL.

The methods of the API are:

- *makeXC()*: create (or activate a reserved) cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks on the data model have been carried out: positively if the request is consistent and queued in the ApiQueue instance, else negatively
  - later, when the cross-connection has been completed, the TNRC Abstract Part will come back using the *actionResponse()* method exposed by G².RSVP-TE external API (see section 7.3) and context (*responseCtxt*), delivering the result of the operation
- any future event related to the cross-connection or one of its component will be reported with the *actionNotify()* method exposed by G².RSVP-TE external API
- *destroyXC()*: destroy an existent cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks on the data model have been carried out: positively if the request is consistent and queued in the ApiQueue instance, else negatively
- later, when the cross-connection removal has been completed, the TNRC Abstract Part will come back using the *actionResponse()* method exposed by G².RSVP-TE external API and context (*responseCtxt*), delivering the result of the operation
- *reserveXC()*: reserve a cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks on the data model have been carried out: poisitively if the request is consistent and queued in the ApiQueue instance, else negatively
  - later, when the cross-connection reservation has been completed, the TNRC Abstract Part will come back using the *actionResponse()* method exposed by G².RSVP-TE external API and context (*responseCtxt*), delivering the result of the operation
  - if the advance reservation flag (*advanceRsrv*) is active and the preliminary checks on the data model have been carried out positively, a make cross-connection action is scheduled to be executed at *startTime*, and a destroy cross-connection one is scheduled to be executed at *endTime*
- *unreserveXC()*: unreserve an existent reserved cross-connection on the equipment, with the following behaviour:
  - it returns soon after the preliminary checks have been carried out: positively if the request is consistent and queued in the ApiQueue instance, else negatively
  - later, when the cross-connection unreservation has been completed, the TNRC Abstract Part will come back using the *actionResponse()* method exposed by G².RSVP-TE external API and context (*responseCtxt*), delivering the result of the operation
- *getDLinkDetails()*: method called to retrieve information about a data link (operational and administrative status, bandwidth parameters, switching capability, encoding type, etc.)

- *getLabelStatus()*: method called to retrieve the status (operational, administrative and label status) of the specified label associated to the specified data link
- *getLabelfromDLink()*: method called to pick a free label among all free labels associated to the specified data link

4.3.3 TNRC Abstract Part action specific API

The action specific API is exposed to TNRC Specific Part, and is used to provide a set of action result callbacks to be called by the Specific Part when the action has been completed on the equipment. It is specified in *<sw_root>/tnrcd/tnrc_apis.h*.

```
void make_xc_resp_cb(tnrcsp_handle_t handle,
                     tnrcsp_result_t result,
                     void *          ctxt);

void destroy_xc_resp_cb(tnrcsp_handle_t handle,
                        tnrcsp_result_t result,
                        void *          ctxt);

void notification_xc_cb(tnrcsp_handle_t        handle,
                        tnrcsp_resource_id_t ** failed_resource_listp,
                        void *                  cxt);

void reserve_xc_resp_cb(tnrcsp_handle_t handle,
                        tnrcsp_result_t result,
                        void *          ctxt);

void unreserve_xc_resp_cb(tnrcsp_handle_t handle,
                          tnrcsp_result_t result,
                          void *          ctxt);
```

Code 4-14: TNRC Abstract Part action specific API.

The methods of the API are:

- *make_xc_resp_cb()*: this method is registered as *response_cb* parameter when TNRC Specific Part API *tnrcsp_make_xc()* method is called by the Abstract Part. An appropriate event is posted to the action FSM and the data model is updated, according to the *result* value
- *destroy_xc_resp_cb()*: this method is registered as *response_cb* parameter when TNRC Specific Part API *tnrcsp_destroy_xc()* method is called by the Abstract Part. An appropriate event is posted to the action FSM and the data model is updated, according to the *result* value
- *notification_xc_cb()*: this method is registered as *async_cb* parameter when TNRC Specific Part API *tnrcsp_make_xc()* method is called by the Abstract Part. The data model is updated according to the event notified

- *reserve_xc_resp_cb()*: this method is registered as *response_cb* parameter when TNRC Specific Part API *tnrcsp_reserve_xc()* method is called by the Abstract Part. An appropriate event is posted to the action FSM and the data model is updated, according to the *result* value
- *unreserved_xc_resp_cb()*: this method is registered as *response_cb* parameter when TNRC Specific Part API *tnrcsp_unreserve_xc()* method is called by the Abstract Part. An appropriate event is posted to the action FSM and the data model is updated, according to the *result* value

4.4 TNRC Specific Part

The TNRC Specific Part is in charge of:

- implementing the specific actions on the hardware, by means of any available and suitable management interface
- decoupling the mechanism of the lower management interface from the upper layers (TNRC Abstract Part)
  - decoupling of blocking/unblocking sync/async communication
  - decoupling of objects or sessions identifiers
- perform any final translation from the semantics and object identifiers passed by TNRC Abstract Part into those needed to communicate with the hardware
- hide away from TNRC Abstract Part some unneeded peculiarities of the underlying transport network

There is a different TNRC Specific Part for each type of equipment (ADVA, Calient,.etc.). A single Specific Part is build via the implementation of a plug-in: this is done inheriting the Plugin class explained in the TNRC Data Model section (see Code 4-8) and implementing the pure virtual methods specified.

The TNRC Specific Part offers an API to the Abstract Part to execute the actions on the equipment.

4.4.1 TNRC Specific Part API

The TNRC Specific Part API consists of the set of methods exposed by the Plugin class (see Code 4-8), and implemented by each specific inherited plug-in.

These methods have already been explained in Section 4.2.8.

4.5 TNRC Action FSM

The main engine of TNRC Abstract Part is the finite state machine of the actions that are executed. Each Action can be the collection of a number of correlated AtomicActions, whose execution and success determines the success of the master Action. The FSM states and events are explained in Table 4-2 and Table 4-3, while the overall FSM picture with the transition events between states are shown in Figure 4-2.

```
#
#  XXX FSM definition
#
#    st/ev    eventI   event1   event2   event3
#
#    stateI     -      state1   state2   state3
#    state1     -      state1     -        -
#    state2     -        -      state2     -
#    state3     -        -        -      state3

{ FSM }

name = TNRC
definition-file = tnrc_action.def
include-name = tnrc_action.h
# start-state = stateX [optional]
graphviz-file = tnrc_action.dot

#
# Events
#
#
# rootEvent = derivedEvent1, derivedEvent2, ...
#

{ Events }

ActionCreate             =  evActionCreate
AtomicActionOk           =  evAtomicActionOk     , evAtomicActionNext,  evActionEndUp,
               evActionEndDown
AtomicActionKo           =  evAtomicActionKo     , evAtomicActionRetry,
evAtomicActionIncomplete, evAtomicActionAbort
ActionNotification       =  evAction Notification
ActionDestroy            =  evActionDestroy      , evActionPending
ActionRollback           =  evActionRollback
AtomicActionTimeout      =  evAtomicActionTimeout
AtomicActionRetryTimer   =  evAtomicActionRetryTimer
AtomicActionDownTimeout  =  evAtomicActionDownTimeout
EqptDown                 =  evEqptDown

#
# States
#
# state = state1   [The first state is the start one if start-state is not set]
#       eventX -> dstState
#
# state = state2
#       eventY -> dstState
#

{ States }

State = Down
      evActionCreate             -> Creating
```

```
State = Creating
      evAtomicActionNext          -> Creating
      evActionPending             -> Dismissed
      evAtomicActionIncomplete    -> Incomplete
      evActionEndUp               -> Up
      evActionDestroy             -> Down
      evAtomicActionKo            -> Down
      evEqptDown                  -> Down

State = Incomplete
      evEqptDown                  -> Down
      evActionRollback            -> Closing

State = Dismissed
      evEqptDown                  -> Down
      evAtomicActionKo            -> Down
      evAtomicActionTimeout       -> Down
      evAtomicActionOk            -> Incomplete
      evAtomicActionIncomplete    -> Incomplete

State = Up
      evActionNotification        -> Up
      evActionDestroy             -> Closing

State = Closing
      evAtomicActionNext          -> Closing
      evAtomicActionRetry         -> Closing
      evAtomicActionRetryTimer    -> Closing
      evAtomicActionDownTimeout   -> Closing
      evAtomicActionAbort         -> Down
      evActionEndDown             -> Down
```

Code 4-15: TNRC Abstract Part action FSM.

state	short description
Down	Initial state of the FSM; none of the AtomicAction has been run yet
Creating	The Action has been created and all the component AtomicActions are executed
Dismissed	The Action has been stopped while going Up, and received a command to destroy itself; but the current AtomicAction is still waiting for a response from the TNRC-SP, and thus the equipment Agent
Incomplete	The Action has been stopped while going Up, and received a command to destroy itself; but the current AtomicAction is not already waiting for a response from the equipment Agent (e.g. its request has not been ack-ed yet and can be silently dismissed)
Up	The Action has successfully run all the component AtomicAction and is now established in an idle state
Closing	The Action is rewinding its “ready” component AtomicActions in order to undo all the atomic operations carried out until the moment when the Destroy command has been received

Table 4-2: TNRC Action FSM: states.

Event	Root event	short description
evActionCreate	**ActionCreate**	Start running the first AtomicAction in the Action
evAtomicActionOk	**AtomicActionOk**	The current AtomicAction has been positively answered by the equipment; do not run the next Atomic Action
evAtomicActionNext	**AtomicActionOk**	The current AtomicAction has been positively answered by the equipment; now run the next AtomicAction
evActionEndUp	**AtomicActionOk**	The current AtomicAction has been positively answered by the equipment, and this was the last AtomicAction in the Action; the Action should go idle into Up state
evActionEndDown	**AtomicActionOk**	the current AtomicAction has been positively answered by the equipment, and this was the last AtomicAction in the Action; the Action should go idle into Down state
evAtomicActionKo	**AtomicActionKo**	The current AtomicAction has been negatively answered by the equipment, and should not be re-issued
evAtomicActionRetry	**AtomicActionKo**	The current AtomicAction has been negatively answered by the equipment, and should be re-attempted later on (after a “retry” interval)
evAtomicActionIncomplete	**AtomicActionKo**	The current AtomicAction has been negatively answered by the equipment, and the Action should go Down; but some other AtomicActions have been successfully carried out before, thus those Action’s AtomicActions need to be rewinded before the Action can go Down
evAtomicActionAbort	**AtomicActionKo**	The current AtomicAction has been negatively answered by the equipment, and should not be reattempted anymore
evActionNotification	**ActionNotification**	The Action has received an asynchronous notification from the equipment about some of its related resources
evActionDestroy	**ActionDestroy**	The Action got a Destroy command, and none of its AtomicActions have been either carried out nor even sent to the equipment
evActionPending	**ActionDestroy**	The Action got a Destroy command, but the current AtomicAction is still waiting for a response from the equipment and, when ready, it might need to be rewinded before the Action can go Down
evActionRollback	**ActionRollback**	Start rewinding this Action from the point it has reached until now with its “ready” AtomicActions
evAtomiActionRetryTimer	**AtomiActionRetryTimer**	The “retry” timer has expired; it is time to reissue the request to the equipment about the currently rewinded AtomicAction

evEqptDown	**EqptDown**	A failure on the TNRC Specific Part – equipment link occurred
evAtomicActionTimeout	**AtomicActionTimeout**	(not used)
evAtomicActionDownTimeout	**AtomicActionDownTimeout**	(not used)

Table 4-3: TNRC Action FSM: events and root events.

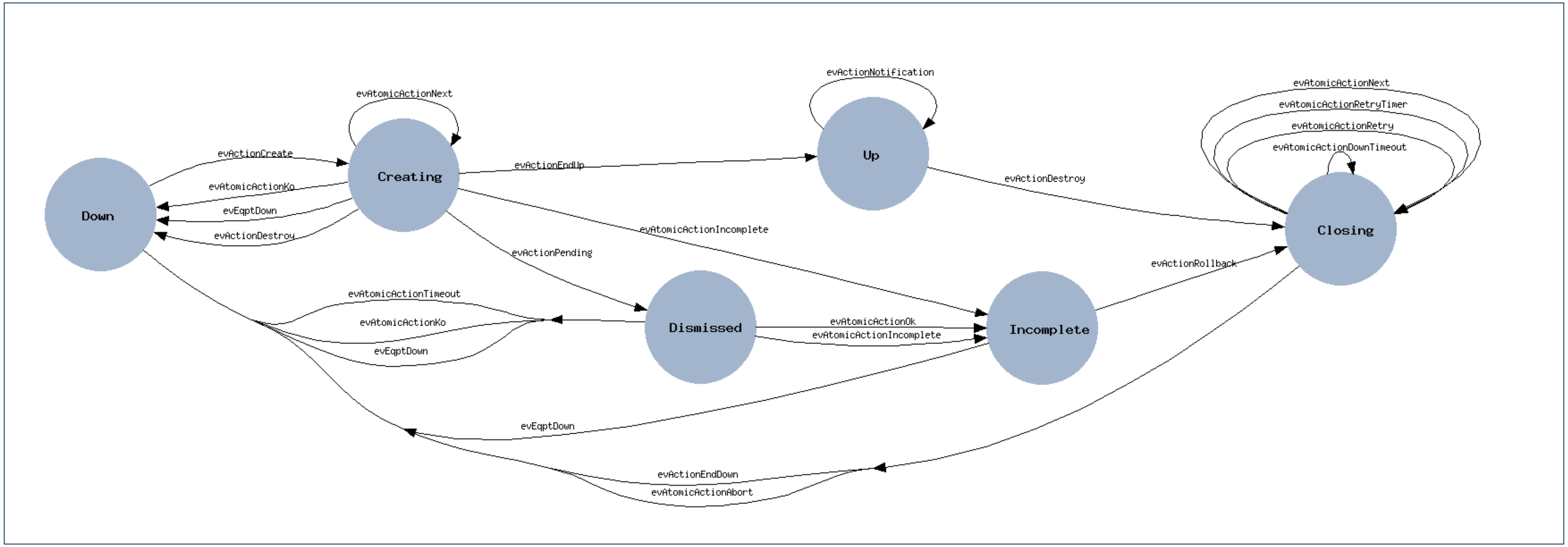

Figure 4-2: TNRC actions finite state machine.

4.5.1 Example transitions

In Figure 4-3 is showed an example of successfully make cross-connection action:

- the initial state is *Down*, an *evActionCreate* event is posted as soon as the Action instance is created.
- the TNRC Specific Part API *tnrcsp_make_xc()* method is called (for every atomic action) and the Action FSM goes to the *Creating* state
- the equipment executes correctly all the atomic actions, and the Specific Part (through the Abstract Part action specific API) post an *evAtomicActionNext* event for each atomic action
- when all atomic are executed, an *evActionEndUp* event is posted and the Action FSM goes to *Up* state, meaning that the cross-connection is correctly done

In Figure 4-4 is showed an example of successfully destroy cross-connection action:

- the initial state is *Up*, an *evActionDestroy* event is posted as soon as the destroy action request is extracted from the queue in the ApiQueue instance, and appropriate Action instance is retrieved
- the TNRC Specific Part API *tnrcsp_destroy_xc()* method is called (for every atomic action) and the Action FSM goes to the *Closing* state
- the equipment executes correctly all the atomic actions, and the Specific Part (through the Abstract Part action specific API) post an *evAtomicActionNext* event for each atomic action
- when all atomic are executed, an *evActionEndDown* event is posted and the Action FSM goes to *Down* state, meaning that the cross-connection is correctly removed. The Action instance is destroyed.

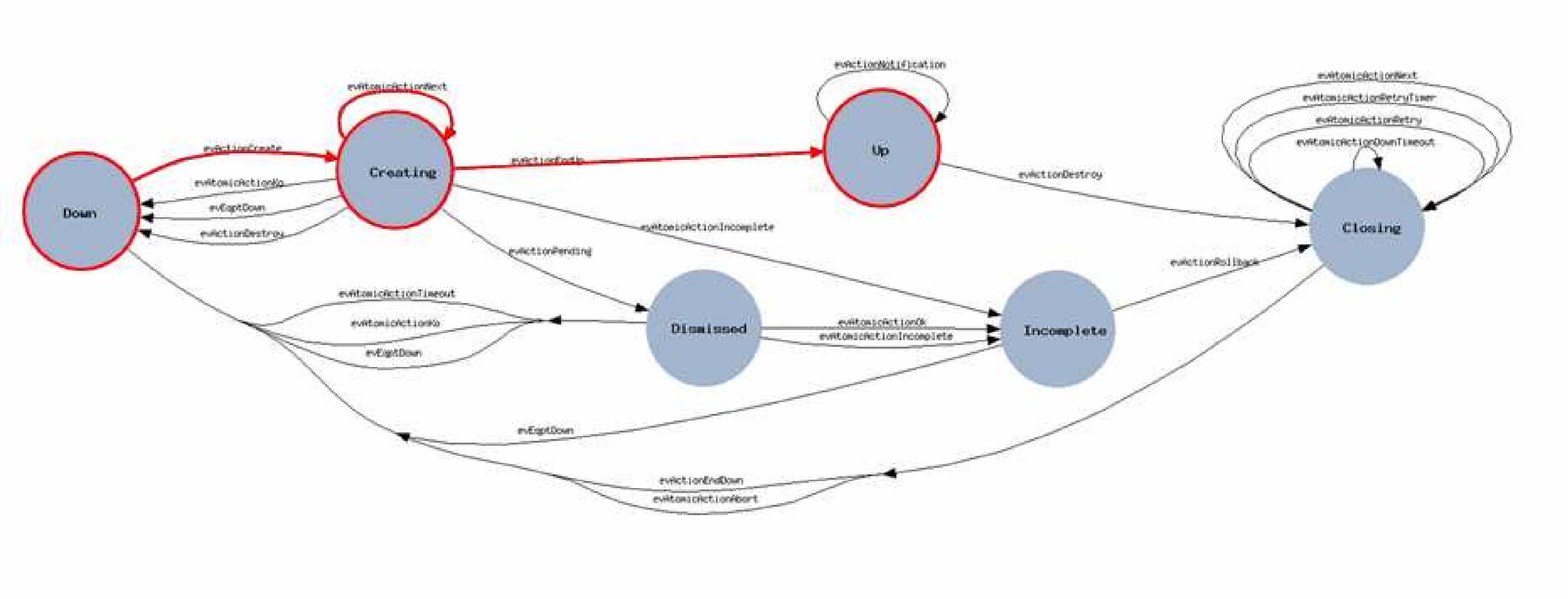

Figure 4-3: TNRC action FSM: example transitions in case of successfully make cross-connection.

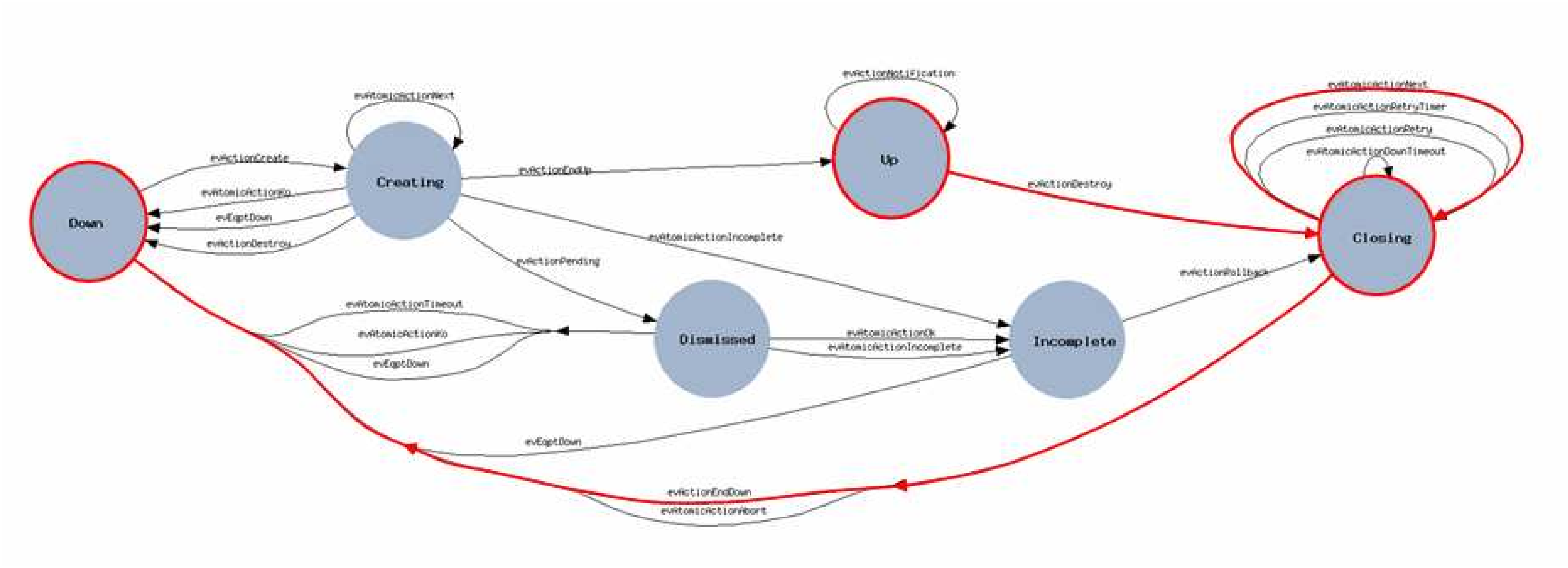

Figure 4-4: action FSM: example transitions in case of successfully destroy cross-connection.

5 Link Resource Manager (LRM)

The LRM module is a separate process, not part of Quagga routing suite and is developed from scratch. It is integrated into Quagga framework according to Quagga daemon main structure (e.g. one master thread to manage the single thread daemon as pseudo multi-thread, the trace log system, the vty interface, etc).

5.1 LRM basics

This LRM module is responsible for the management of the relationships among TE-Links, Data-Links, Control Channels and SCN Interfaces. The TE-links are the result of a bundling procedure applied to a number of physical component Data-Links with the eligibility for being part of the same logical construct.

The functionalities of the LRM comprise:

- Selection and allocation/de-allocation of resources (<Data-link, label>) in TE-link for signaling purposes,
- Management of the TE-link status and bundling information for topology purposes.

The LRM module exposes interfaces to gunirsvpd, G^2.RSVP-TE, TNRC, ospfd, SCNGW and g2nccd.

5.2 LRM Data Model

The LRM Data Model also holds the basic instances of nearly all the G^2MPLS items. Each external module remaps its own “view” or “instance” of a basic item (e.g. a Data-Links, ora TE-Link), but the basic item itself is maintained and hosted by the LRM module.

The LRM Data Model is depicted in Figure 5-1.

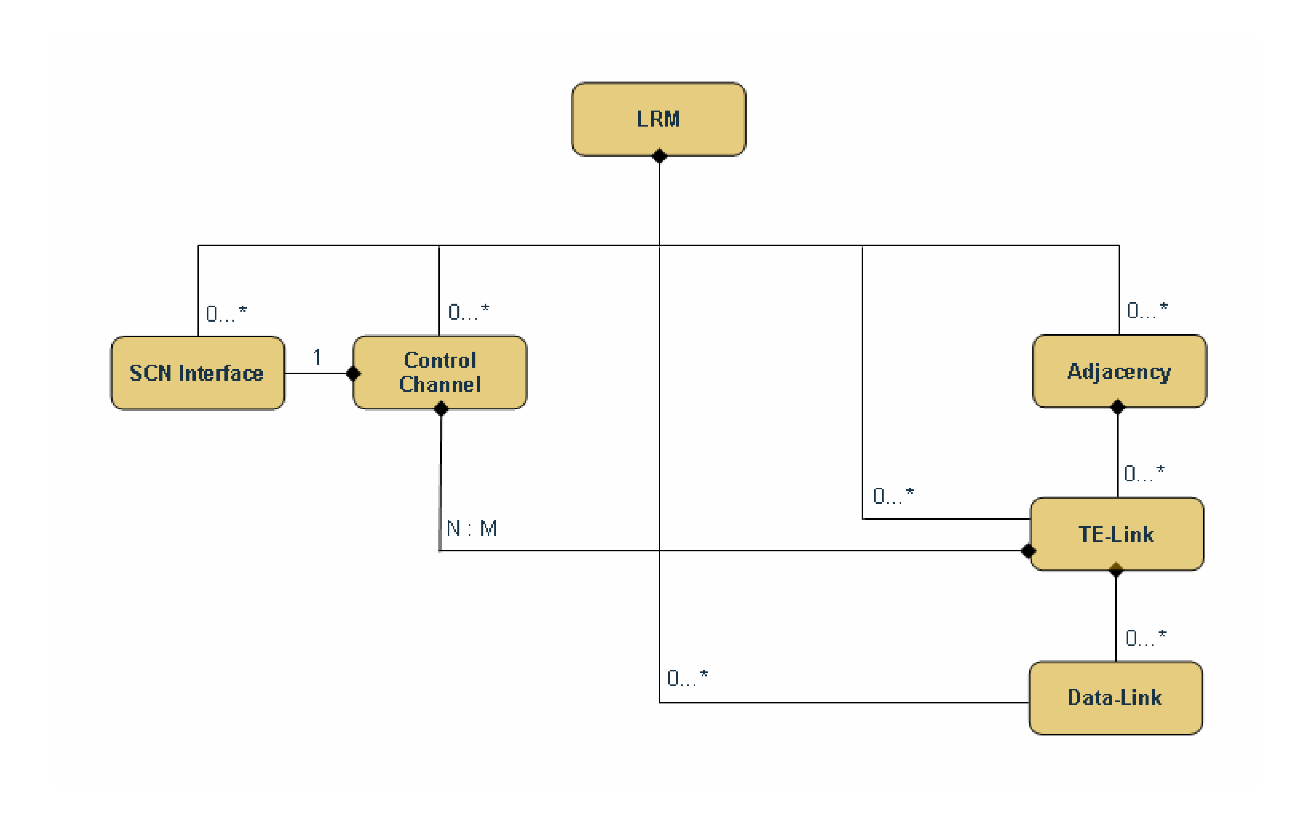

Figure 5-1: LRM Data Model.

5.2.1 LRM instance

The LRM instance is the root of the whole LRM Data Model. When LRM process starts, a global instance of LRM is created. It holds:

- a unique network address of G^{2}MPLS controller (*router_id*)
- all the SCN Interfaces instances (*scn_if_list*)
- all the Control Channel Instances (*cc_list*)
- all the Data-Link instances (*datalink_list*)
- all the TE-Link instances (*telink_list*)
- alle the Adjacency instances (*adj_list*)

```
typedef struct lrm {
        u_int32_t        router_id;
        struct zlist * scn_if_list;
        struct zlist * cc_list;
        struct zlist * datalink_list;
```

```
        struct zlist * telink_list;
        struct zlist * adj_list;
        uint32_t       telink_count;
} lrm_t;
```

Code 5-1: LRM instance

5.2.2 SCN Interface instance

The SCN Interface is the basic item for the Control Network management. The SCN Interface instances are created reading a configuration file containing the description of the entire data model. Each instance holds:

- the network address of the interface (*addr*)
- the type (broadcast/point-to-point) of the interface (*type*)
- operational state of the interface (*op_state*)
- administrative sate of the interface (*adm_state*)

```
typedef struct ctrl_interface {
g2mpls_addr_t     addr;
if_type_t         type;
opstate_t         op_state;
admstate_t        adm_state;
int               sync_status;
} ctrl_intf_t;
```

Code 5-2: SCN interface instance

5.2.3 Control Channel instance

The Control Channel is a fundamental item in the Control Network management, and represent the binding of two (local and remote) SCN Interfaces in the Control Network. The Control Channel instances are created reading a configuration file containing the description of the entire data model. Each instance holds:

- a unique local identifier of the Control Channel (*cc_id*)
- the remote identifier of the Contro Channel (*rem_cc_id*)
- local SCN Interface address (*lcl_scn_addr*)
- remote SCN Interface address (*rem_scn_addr*)

```
typedef struct control_channel {
u_int32_t        cc_id;              /* local  and node-unique CC id */
u_int32_t        rem_cc_id;          /* remote and node-unique CC id */
```

```
g2mpls_addr_t    lcl_scn_addr;
g2mpls_addr_t    rem_scn_addr;
opstate_t        op_state;
admstate_t       adm_state;
struct zlist   * te_link_list;       /*  te-links managed by this CC */
       int              sync_status;
} cc_t;
```

Code 5-3: Control Channel instance

5.2.4 Adjacency instance

The Adjacency is the highest level item of the Transport Netwok part of the data model. The Adjacency instances are created each time a new TE-Link instance not linked to any Adjacency is created. Each instance holds:

- a unique local identifier of the Adjacency ()
- the network address of the remote G^{2}MPLS controller (its *router_id* parameter of LRM instance) (*adj_addr*)
- the type of the Adjacency (INNI/ENNI/UNI) (*link_type*)
- the list of all associated TE-Link instances

```
typedef struct adj {
       u_int32_t        adj_id;
       u_int32_t        adj_addr;
       adj_type_t       link_type;
       struct zlist   * tel_list;
} adj_t;
```

Code 5-4: Adjacency instance.

5.2.5 TE-Link instance

The TE-Link is the basic routable item of the data model. The TE-Link instances are created reading a configuration file containing the description of the entire data model.

The most relevant fields are:

- local address (*lcl_id*)
- remote address (*rem_id*)

- the network address of the remote G^2MPLS controller (its *router_id* parameter of LRM instance) (*rem_node_id*)
- operational state (*op_state*)
- administrative state (*adm_state*)
- the type of the Adjacency (*adj_type*)
- the TE metric (*te_metric*)
- the list SRLG the TE-Link instance belongs to (*SRLG_ids*)
- the switching capability (*swcap*). This parameter must be the same for all associated Data-Link instances
- the encoding type (*enctype*). This parameter must be the same for all associated Data-Link instances
- total available bandwidth (*max_bw*)
- maximum reservable bandwidth (*max_res_bw*)
- unreserved bandwidth per priority (*avail_bw_per_prio*)
- minimum reservable bandwidth per LSP (*min_LSP_bw*)
- maximum reservable bandiwdth per LSP and per priority (*max_LSP_bw*)
- the list of all associated Data-Link instances (*dl_list*). All the above bandwidth parameters are a bundle of the correspondent parameters of the associated Data-Link instances
- a pointer to the parent Adjacency instance (*adj*)
- the list of all associated Control Channel instances (*cc_list*)

```
typedef struct _te_link {
       g2mpls_addr_t lcl_id;
       g2mpls_addr_t rem_id;
       u_int32_t     rem_node_id; /* rem_id MUST be contained */
       char          tel_name[20 + 1]; /* name of TEL */
       int           tel_key;     /* used for internal purposes */
       opstate_t     op_state;
       admstate_t    adm_state;
       /* Summary (after bundling) or configured TE info */
       adj_type_t     adj_type;
       u_int32_t      te_metric;
       u_int32_t      link_color;
       struct zlist * SRLG_ids;

       sw_cap_t       swcap;      /* switching capability */
       enc_type_t     enctype;     /* encoding type */
       u_int32_t      max_bw;
       u_int32_t      max_res_bw;
       /* unreserved bw per priority */
       u_int32_t      avail_bw_per_prio[MAX_BW_PRIORITIES];
       /*max of max LSP per priority p bw of component links*/
       u_int32_t      max_LSP_bw[MAX_BW_PRIORITIES];
       u_int32_t      min_LSP_bw;

       struct zlist * dl_list;   /* list of data links into te-link */
       adj_t *        adj;
       cc_t *         assoc_cc;  /* cc associated whith this te-link */
       struct zlist * cc_list;   /* list of CCs for this TEL */
       u_int32_t      num_cc_up; /* number of available CCs in up */
```

```
      int             sync_status;
} te_link_t;
```

Code 5-5: TE-Link instance.

5.2.6 Data-Link instance

The Data-Link is the lowest level item of the Transport Netwok part of the data model. The Data-Link instances are created reading a configuration file containing the description of the entire data model. Each instance holds:

- local Transport Network address (*lcl_id*)
- remote Transport Network address (*rem_id*)
- operational state (*op_state*)
- administrative state (*adm_state*)
- the switching capability (*swcap*)
- the encoding type (*enctype*)
- total available bandwidth (*max_bw*)
- maximum reservable bandwidth (*max_res_bw*)
- unreserved bandwidth per priority (*avail_bw_per_prio*)
- minimum reservable bandwidth per LSP (*min_LSP_bw*)
- maximum reservable bandiwdth per LSP and per priority (*max_LSP_bw*)

```
typedef struct datalink {
g2mpls_addr_t    lcl_id;
g2mpls_addr_t    rem_id;
opstate_t        op_state;
admstate_t       adm_state;
sw_cap_t         swcap;      /* switching capability */
enc_type_t       enctype;    /* the encoding type of this data link */
u_int32_t        max_bw;
u_int32_t        max_res_bw;
u_int32_t        avail_bw_per_prio[MAX_BW_PRIORITIES];
u_int32_t        max_LSP_bw[MAX_BW_PRIORITIES];
u_int32_t        min_LSP_bw;
} datalink_t;
```

Code 5-6: Data Link instance.

5.3 LRM configuration API

The LRM configuration API is used to build the LRM data model starting from the configuration file containing its description.

It is specified in *<sw_root>/lrmd/lrm_core.h*.

```
int      lrm_set_rid(lrm_t * lrm, u_int32_t rid);

/* CTRL IF related functions */
int   scn_if_add(lrm_t * lrm, g2mpls_addr_t addr, if_type_t intf_type);
int   scn_if_del(lrm_t * lrm, g2mpls_addr_t addr);
int   scn_if_ena(lrm_t * lrm, g2mpls_addr_t addr);
int   scn_if_dis(lrm_t * lrm, g2mpls_addr_t addr);


/* CC related functions */
int   control_channel_add(lrm_t *            lrm,
                              u_int32_t          cc_id,
                              g2mpls_addr_t      lcl_scn,
                      g2mpls_addr_t      rem_scn);
int   control_channel_del(lrm_t * lrm, u_int32_t cc_id);
int   control_channel_ena(lrm_t * lrm, u_int32_t cc_id);
int   control_channel_dis(lrm_t * lrm, u_int32_t cc_id);
int   control_channel_up(lrm_t * lrm, u_int32_t cc_id);   /* static-LMP */
int   control_channel_down(lrm_t * lrm, u_int32_t cc_id); /* static-LMP */


/* DATA LINK related functions */
int   data_link_add(lrm_t * lrm, g2mpls_addr_t dl_id, g2mpls_addr_t rem_dl_id);
int   data_link_del(lrm_t * lrm, g2mpls_addr_t dl_id);
int   data_link_ena(lrm_t * lrm, g2mpls_addr_t dl_id);
int   data_link_dis(lrm_t * lrm, g2mpls_addr_t dl_id);


/* TE-LINK related functions */
int   te_link_add(lrm_t *       lrm,
                    g2mpls_addr_t tel_id,
                    g2mpls_addr_t r_tel_id,
                    u_int32_t     adj_rid,
                    adj_type_t    adj_type);
int   te_link_del(lrm_t * lrm, g2mpls_addr_t tel_id);
int   te_link_ena(lrm_t * lrm, g2mpls_addr_t tel_id);
int   te_link_dis(lrm_t * lrm, g2mpls_addr_t tel_id);
int   te_link_bind_cc(lrm_t * lrm, g2mpls_addr_t tel_id, u_int32_t cc_id);
int   te_link_unbind_cc(lrm_t * lrm, g2mpls_addr_t tel_id, u_int32_t cc_id);
int   te_link_push_dl(lrm_t * lrm, g2mpls_addr_t tel_id, g2mpls_addr_t dl_id);
int   te_link_pop_dl(lrm_t * lrm, g2mpls_addr_t tel_id, g2mpls_addr_t dl_id);
int   te_link_set_te_metric(lrm_t *       lrm,
                                g2mpls_addr_t tel_id,
                                u_int32_t     te_metric);
int   te_link_set_link_color(lrm_t *       lrm,
                                g2mpls_addr_t tel_id,
                                u_int32_t     colotmask);
```

```
int   te_link_add_srlg_id(lrm_t *        lrm,
                            g2mpls_addr_t  tel_id,
                            u_int32_t      SRLG_id);
int   te_link_rem_srlg_id(lrm_t *        lrm,
                            g2mpls_addr_t  tel_id,
                            u_int32_t      SRLG_id);
```

Code 5-7: LRM configuration API.

The methods of the API are:

- *scn_if_add()*: add a new SCN Interface instance in the data model, probing the specified interface by means of *ioctl()* system call. Advertise SCNGW module through its external API of this addition
- *scn_if_del()*: delete an existent SCN Interface instance from the data model. Advertise SCNGW module through its external API of this deletion
- *scn_if_en()*: set the administrative state of the SCN Interface istance to ENABLED
- *scn_if_dis()*: set the administrative state of the SCN Interface instance to DISABLED

- *control_channel_add()*: add a new Control Channel instance in the data model. Advertise SCNGW module through its external API of this addition
- *control_channel_del()*: delete an existent Control Channel instance from the data model. Advertise SCNGW module through its external API of this deletion
- *control_channel_en()*: set the administrative state of the Control Channel instance to ENABLED
- *control_channel_dis()*: set the administrative state of the Control Channel instance to DISABLED
- *control_channel_up()*: set the operational state of the Control Channel instance to UP
- *control_channel_down()*: set the operational state of the Control Channel instance to DOWN

- *data_link_add()*: add a new Data-Link instance in the data model, checking if this is consistent with TNRC Abstract Part image of the equipment (through its external API)
- *data_link_del()*: delete an existent Data-Link instance from the data model
- *data_link_en()*: set the administrative state of the Data-Link instance to ENABLED
- *data_link_dis()*:set the administrative state of the Data-Link instance to DISABLED

- *te_link_add()*: add a new TE-Link instance in the data model. Advertise SCNGW module through its external API of this addition
- *te_link_del()*: delete an existent TE-Link instance from the data model. Advertise SCNGW module through its external API of this deletion
- *te_link_en()*:set the administrative state of the TE-Link instance to ENABLED
- *te_link_dis()*:set the administrative state of the TE-Link instance to DISABLED
- *te_link_bind_cc()*: bind the specified Control Channel instance to specified TE-link Instance
- *te_link_unbind_cc()*: unbind the specified Control Channel instance from specified TE-link Instance
- *te_link_push_dl()*: associate the specified Data-Link instance to specified TE-link Instance

- *te_link_pop_dl()*:disassociate the specified Data-Link instance from specified TE-link Instance
- *te_link_set_te_metric()*: set the metric for specified TE-link Instance

1.) *te_link_set_link_color()*: set the link color for specified TE-link Instance
2.) *te_link_add_srlg_id()*: add a SRLG to the specified TE-Link list of SRLGs

- *te_link_rem_srlg_id()*: remove a SRLG from the specified TE-Link list of SRLGs

5.4 LRM external API

The LRM external API is used to allow external modules to retrieve information about LRM data model. It is specified in *<sw_root>/idl/lrm.idl*.

```
#include "types.idl"
#include "g2mplsTypes.idl"

interface LRM {

       exception InternalProblems      { };
       exception UnknownTELinkIdentity { g2mplsTypes::TELinkId id; };
       exception UnknownDLinkIdentity  { g2mplsTypes::DLinkId  id; };
       exception UnknownTELink         { };
       exception UnknownDLink          { };
       exception UnknownAdjId          { };
       exception UnknownNodeId         { };
       exception NoTELinks             { };

       void localDLinkIdFromRemoteDLinkId(in  g2mplsTypes::nodeId     nodeId,
                                          in  g2mplsTypes::DLinkId     remoteDLink,
                                          out g2mplsTypes::DLinkId     localDLink,
                                          out g2mplsTypes::operState   operState,
                                          out g2mplsTypes::adminState adminState)
              raises (InternalProblems, UnknownDLink, UnknownNodeId);

       g2mplsTypes::TELinkId TELinkFromDLink(in g2mplsTypes::DLinkId datalink)
              raises (InternalProblems, UnknownDLink);

       g2mplsTypes::DLinkId DLinkFromTELink(in g2mplsTypes::TELinkId telink)
              raises (InternalProblems, UnknownTELink);

       void TELinksData(inout g2mplsTypes::TELinkDataSeq telinks)
              raises (InternalProblems, UnknownTELinkIdentity);

       void DLinksData(inout g2mplsTypes::DLinkDataSeq datalinks)
              raises (InternalProblems, UnknownDLinkIdentity);

       g2mplsTypes::TELinkIdSeq allTELinkIds()
              raises (InternalProblems);

       g2mplsTypes::TELinkDataSeq allTELinks(in g2mplsTypes::adjType type)
              raises (InternalProblems, NoTELinks);

       g2mplsTypes::nodeId nodeId()
              raises (InternalProblems);
```

```
        void scngw_isup()
                raises (InternalProblems);
};
```

Code 5-8: LRM external API.

The methods of the API are:

- *localDLinkIdFromRemoteDLinkId()*: retrieve local Data-Link address for specified remote Data-Link address. This method return also the Data-Link instance administrative and operational state
- *TELinkFromDLink()*: retrieve parent TE-Link local address for the specified Data-Link instance
- *DLinkFromTELink()*: get a Data-Link instance local address among specified TE-Link instance list of associated Data-Links
- *TELinksData()*: get TE-Link instance parameters for specified TE-Link instance
- *DLinksData()*: get Data-Link instance parameters for specified Data-Link instance
- *allTELinkIds()*: get all TE-Links instance local address
- *allTELinks()*: get all TE-Links instance local address for a specified Adjacency type
- *nodeId()*: get the network address of G^2MPLS controller (*router_id* parameter of the LRM instance)
- *scngw_isup()*: this method is called by SCNGW module to start the synchronization phase in the communication with LRM module (see SCNGW server external API)

6 SCN Gateway (SCNGW)

The SCNGW module is not part of Quagga routing suite and is developed from scratch. It is integrated into Quagga framework according to Quagga daemon main structure (e.g. one master thread to manage the single thread daemon as pseudo multi-thread, the trace log system, the vty interface, etc).

6.1 SCNGW basics

The SCNGW module has the role to manage the dualism between the Transport Network and the Control Network. It's a kind of socket manager responsible of mapping TN resources (TE-links, well known by G^{2}MPLS protocols) into SCN resources (control i/fs, unknown by G^{2}MPLS protocols). The main functionalities of the SCN Gateway are:

- maintain the bindings between TE-links, Control Channels and SCN interfaces
- send the G^{2}MPLS protocols' SDUs on the appropriate Control Channels
- dispatch received SDUs (from network) to the correct G^{2}MPLS protocol

SCNGW exposes interface to G2.RSVP-TE, G.UNI-GW, G.E-NNI RSVP, G.I-NNI RSVP (G^{2}MPLS protocols) and LRM. For these purposes, the module is broken down into two sub-modules:

module	sub-module	short description
SCNGW (SCN Gateway)	SCNGW client (SCNGWC)	Library offering a wrapped socket API, to be linked by each protocol wanting communication across the SCN. It acts as an access i/f to the SCNGW server, and has 2 channels with it: 1 for data, 1 for control (e.g. open/close sockets, etc.)

	SCNGW server (SCNGWS)	Separate process (i.e. a socket manager) handling (tunnelled) communication through the SCN for one or more clients. It maps TN resources (TE links) into SCN resources (control i/fs) via the TE links <-> CCs association.

Table 6-1: SCNGW breakdown into two sub-modules.

The overall structure of the SCNGW module is depicted in Figure 6-1.

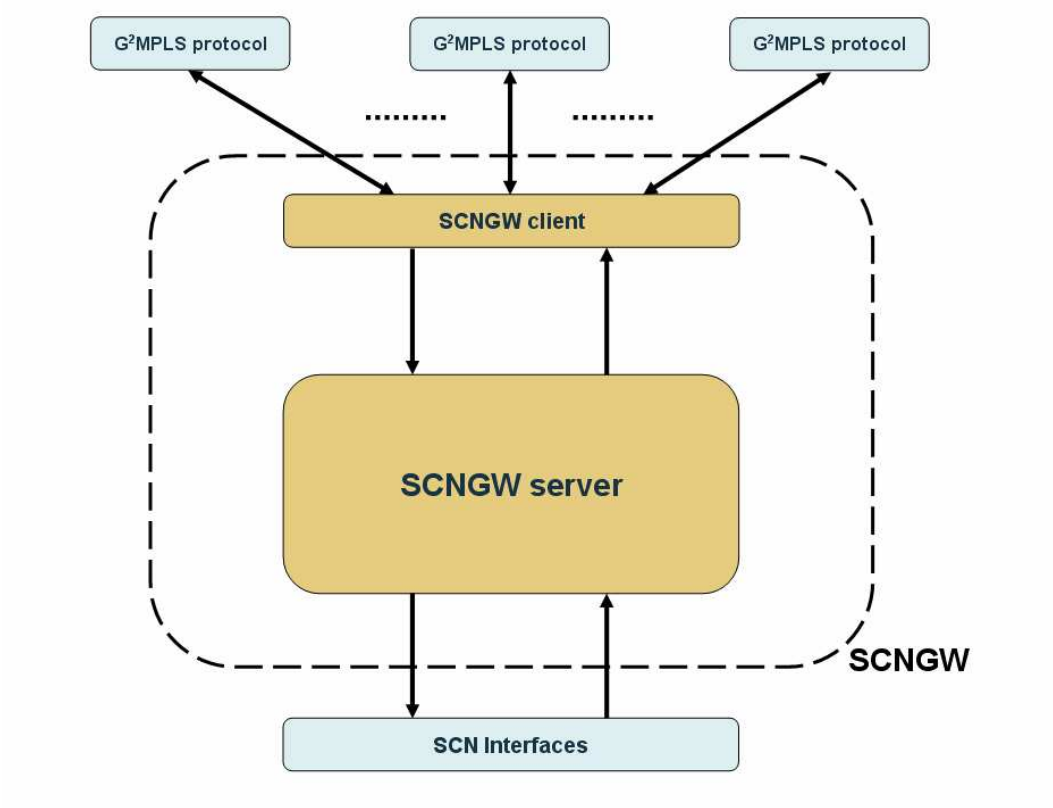

Figure 6-1: SCNGW module structure.

6.2 SCNGW client

The client part of SCNGW is responsible of the communication with the G^{2}MPLS protocols; it is a library that wraps the standard socket functions and exposes an API each protocol can use to interact with the SCNGW server, that is the core of the SCNGW module.

```
#ifndef IPPROTO_OSPFIGP
#define IPPROTO_OSPFIGP         89
#endif /* IPPROTO_OSPFIGP */

#ifndef IPPROTO_RSVP
#define IPPROTO_RSVP            46
#endif /* IPPROTO_RSVP */

#define OSPF_PORT            61089
#define RSVP_PORT            61046

#define NO_TUNNEL                0 /* want no encapsulation in SCNGW server */
#define TUNNEL                   1 /* want encapsulation in SCNGW server */

#define WANT_NO_ACK              0 /* want no response on packet from SCNGW */
#define WANT_ACK                 1 /* want response on packet from SCNGW */

/* Protoypes */
extern int scngwc_init           (int interface_type,
                                      int protocol,
                                      int encap,
                                      int want_ack);

extern int scngwc_sendmsg        (int            sock,
                                     const void *   sdu,
                                     u_int16_t      sdu_size,
                                     struct in_addr src_addr,
                                     struct in_addr dst_addr,
                                     int            flags,
                                     int *          unread_packets);

extern int scngwc_stream_recvmsg (void *       sdu,
                                     int          sock,
                                     struct ip ** iph,
                                     size_t       size);

extern void scngwc_close         (int sock);
```

Code 6-1: SCNGW client API.

The interaction between the protocols and the SCNGW client takes place in three different actions:

- initialization
- exchanging of the SDUs
- closing

The functions of the API are:

- *scngwc_init()*: this function is responsible for the initialization phase, opening the TCP socket between the client and the server part of the SCNGW module that will serve the considered G^2MPLS protocol. It also includes a registration of the protocol to the SCNGW server. The protocol that want to interact with SCNGW has to call this function only once, declaring
  - *protocol*: what kind of protocol it is (e.g. OSPF or RSVP)
  - *interface_type*: the adjacency type (e.g. INNI or ENNI or UNI)
  - *encap*: if it wants his SDUs be encapsulated by SCNGW server
  - *want_ack*: if it wants an acknowledgment by SCNGW server of the transmission of the SDU on the network
- *scngwc_sendmsg()*: this function is responsible for the exchanging of the SDUs phase, in the direction G^2MPLS protocol→ SCNGW client. If the protocol wants his SDUs encapsulated, it also builds the first IP packet header for the specified protocol SDU. When a protocol has to send its SDU has to specify
  - *sock:* file descriptor returned by *scngwc_init()*
  - *sdu:* pointer to the buffer containing the SDU
  - *sdu_size:* length of the SDU (bytes)
  - *src_addr:* address of the source TE-link, used to build the IP packet header and (in the SCNGW server) to retrieve the correct SCN interface)
  - *dst_addr:* address of the destination TE-link, used to build the IP packet header and (in the SCNGW server) to retrieve the correct SCN interface)
  - *flags:* flags to be used by the SCNGW server when sending the SDU on the network
  - *unread_packets:* flag valorized by SCNGW (out parameter) that specify if there any unread packets for the protocol
- *scngwc_recvmsg()*: this function is responsible for the exchanging of the SDUs phase, in the direction SCNGW client→G^2MPLS protocol. If the protocol wants his SDUs encapsulated, it also remove the last

IP packet header of the incoming (from SCNGW server) packet, offering to the protocol the only SDU. The protocol has to specify

- *sdu:* pointer to the buffer for the incoming SDU
- *sock:* file descriptor returned by *scnwgc_init()* and that is set by the incoming SDU
- *iph:* pointer to the buffer for the incoming IP packet header
- *size:* length of the buffer specified by the parameter *sdu*

- *scngwc_close()*: this function is responsible of the closing phase, opening the TCP socket between the client and the server part of the SCNGW module. The G^2MPLS protocol has to specify
  - *sock*: file descriptor returned by *scnwgc_init()*

For each G^2MPLS protocol, the client and the server part of SCNGW module communicates each other through a different socket. To improve this communication, each time there is a protocol SDU to be sent/received by SCNGW client from SCNGW server and viceversa, a specific SCNGW overhead is added to the entire message exchanged, containing some useful information about TE-links and SDU.

```
#define PACKET_MESSAGE          1U
#define ACK_MESSAGE             2U
#define NACK_MESSAGE            3U
#define REGISTRATION_MESSAGE    4U

/* structure containing the SCNGW header parameters */
struct scngw_hdr {
       u_int32_t           msg_type;  /* Message type */
       u_int32_t           hdr_len;   /* header length (bytes) */
       u_int32_t           sdu_len;   /* SDU length (bytes) */
       u_int32_t           msg_id;    /* Message ID */
       u_int32_t           flags;     /* flags used by protocols */
       u_int32_t           src_addr;  /* TE-link local address */
       u_int32_t           dst_addr;  /* TE-link remote address */
       u_int32_t           cc;        /* Control channel */
};
```

Code 6-2: SCNGW header structure.

This overhead allows to simply identify the exchanged message type, and to retrieve basic information about the TE-links' addresses and the size of the SDU without reading the specific IP header packet fields.

6.3 SCNGW server

The server part of the SCNGW is the core of this module. It is in charge of sending the SDU of the G^{2}MPLS protocol on the correct SCN interface (for the specified couple source/destination TE-links), and receiving packets from the network dispatching the contained SDUs to the appropriate protocol.

To do that, SCNGW server maintains:

- an up-to-date association between TE-links/Control Channels/SCN interfaces through a communication with LRM module
- a list of all registered G^{2}MPLS protocols

When a registered G^{2}MPLS protocol has to send its SDU:

- SCNGW client send to SCNGW server the SDU (with the IP packet header added if requested by the G^{2}MPLS protocol) adding the SCNGW header
- SCNGW server receives the message, read the SCNGW header and bind the SDU to the correct registered G^{2}MPLS protocol. Put the message in an internal queue of messages (associated with the specific protocol) to send on the network.
- SCNGW server extracts first message from the queue, retrieves the appropriate Control Channel and SCN interfaces for specified TE-links, add the last IP packet header and finally send the packet on the correct SCN interface

When SCNGW server receives a packet from network:

- SCNGW server retrieves the SCN interface of the incoming packet
- SCNGW server fetches the appropriate registered protocol ("owner" of the incoming packet) basing on the associations TE-links/Control Channels/SCN interfaces
- SCNGW server extracts first message from the queue, and send it to the G^{2}MPLS protocol (through SCNGW client), adding the SCNGW header

6.3.1 SCNGW server data structures

The SCNGW server data structures are specified in *<sw_root>/scngwd/scngws.h*.

```
#include "stream.h"
#include "linklist.h"
#include "scngws_packet.h"

/* SCNGWs  master for system wide configuration and variables. */
struct scn_master {
```

```
        /* SCNGW thread master. */
        struct thread_master *master;
        struct zlist *client_list;

        /* Thread for END_SYNC timeout in LRM communication */
        struct thread *timer_thread;

        /* Status of the connection with LRM */
        int lrm_conn_status;

        /* Timeout for synchronization phase with LRM (sec) */
        long int timeout_lrmsync;

        /* SCNGWs start time. */
        time_t start_time;
};

/* Structure for the SCNGWs client. */
struct scn_client {

        /* Client protocol */
        int proto;
        /* Client interface type */
        int interface_type;
        /* Encapsulation */
        int encap;
        /* ACK / NO ACK */
        int want_ack;
        /* Number of packets sent */
        int pckts_sent;
        /* Number of packets received */
        int pckts_rcvd;

        /* Socket */
        int fd_cl;
        int fd_net;

        /* Input buffers*/
        struct stream *ibuf_cl;
        /* Output queues. */
        struct scngws_fifo *obufq_cl;
        struct scngws_fifo *obufq_net;

        /* threads. */
        struct thread *t_read_cl;
        struct thread *t_write_cl;
        struct thread *t_read_net;
        struct thread *t_write_net;

};
/* Structure containing one SCN-if */
struct scnif {

        /* Status of SCN-if */
        int status:
        /*local SCN-if address*/
        struct in_addr loc_addr;
```

```
};

/* Structure containing one TE-link/CC association */
struct tel_cc_assoc {

       /* Status of the TE-link */
       int status;
       /* INNI / ENNI / UNI */
       u_int16_t interface_type;
       /* key used to update association */
       u_int32_t key;
       /* local TE-link address*/
       struct in_addr tel_loc;
       /* remote TE-link address*/
       struct in_addr tel_rem;
       /* control channels (id) associated*/
       struct zlist *cclist;
};

/* Structure containing one CC/SCN-if association */
struct cc_scnif_assoc {
       /* Status of the CC */
       int status;
       /* control channel id*/
       u_int32_t cc_id;
       /*local SCN-interface address*/
       struct in_addr scnif_loc;
       /*remote SCN-interface address*/
       struct in_addr scnif_rem;
};

/* SCN-if structure */
struct scn_if_addrs {
       /*local SCN-interface address*/
       struct in_addr loc_addr;
       /*remote SCN-interface address*/
       struct in_addr rem_addr;
       int mtu;
};
```

Code 6-3: SCNGWS data structures.

The *scn_client* structure identifies a registered G^{2}MPLS protocol. The registration is done by SCNGW client when the protocol is in the initialization phase of the communication with the client part. This structure contains information about the parameters specified by the protocol (adjacency type, encapsulation, etc.), the file descriptor of the sockets opened toward SCNGW client and the network, buffers and queues for internal packets storage.

In the *scn_master* structure is stored the list of all registered protocols, used to retrieve the appropriate protocol when a packet is receveid on a certain SCN interface.

The *tel_cc_assoc*, *cc_scnif_assoc* and *scnif_addrs* structures identify respectively a singular TE-link/Control Channelsl association, Control Channel/SCN interfaces association and a couple of local/remote SCN interface addresses. These structures are created and updated by the communication with the LRM module. A list of all these associations is maintained in SCNGW server ass global variable.

6.3.2 SCNGW server external API

The API for the communication with the LRM module is specified in *<sw_root>/idl/scngw.idl*.

```
#include "types.idl"
#include "g2mplsTypes.idl"

interface SCNGW {
       exception SyncErr { };
       exception InternalProblems { };
       exception CCNotFound { };

       void begin_sync(in long scnif_count,
                       in long cc_count,
                       in long telink_count)
             raises(SyncErr);

       void end_sync()
             raises(SyncErr);

       void sync_fatal_error();

       void scnif_add(in g2mplsTypes::addr addr)
             raises(SyncErr);

       void scnif_delete(in g2mplsTypes::addr addr)
             raises(SyncErr);

       void tel_cc_assoc_add(in TELCC_Add_AssocSeq assocs)
             raises(InternalProblems, SyncErr);

       void tel_cc_assoc_update(in long key_id,
                                in UpdateSeq updates)
             raises(SyncErr);

       void tel_cc_assoc_delete(in TELCC_Delete_AssocSeq assocs)
             raises(SyncErr);

       void cc_scnif_assoc_add(in CC_Add_AssocSeq assocs)
             raises(InternalProblems, SyncErr);

       void cc_scnif_assoc_update(in long               cc_id,
                                  in g2mplsTypes::addr  local_addr,
                                  in g2mplsTypes::addr  remote_addr)
             raises (CCNotFound, SyncErr);

       void cc_scnif_assoc_delete(in CC_Delete_AssocSeq assocs)
             raises(SyncErr);
```

```
};
```

Code 6-4: SCNGW server external API IDL.

When the SCNGW server process starts, the *scngw_is_up()* LRM module external API function is called, and a synchronization phase starts. During this phase LRM send all the TE-links/Control Channels/SCN interfaces associations to SCNGW server (through the external API specified above). If something goes wrong during the synchronization of the associations, SCNGW server deletes all the associations created and call again *scngw_is_up()*, to restart the synchronization. When the synchronization ends correctly, SCNGW server is ready to use the associations to send the protocols' packets through the appropriate Control Channels.

The LRM module can add, delete or update some association simply calling, out of synchronization, an external API function.

The external API functions are:

- *begin_sync()*: start of the synchronization of all associations (to be called specifying the number of associations to send)
- *end_sync()*: end of the synchrionization of all associations
- *sync_fatal_error()*: fatal error in synchronization (to be called after 5 in a row unsuccessfully synchronization)
- *scnif_add()*: add a couple of local/remote SCN interfaces (to be called either in synchronization phase or to add a new association)
- *scnif_delete()*: delete an existent couple of local/remote SCN interfaces (to be called out of synchronization phase)
- *tel_cc_assoc_add()*: add a TE-link/Control Channel association (to be called either in synchronization phase or to add a new association)
- *tel_cc_assoc_update()*: update an existent TE-link/Control Channel association (to be called out of synchronization phase)
- *tel_cc_assoc_delete()*: delete an existent TE-link/Control Channel association (to be called out of synchronization phase)
- *cc_scnif_assoc_add()*: add a Control Channel/SCN interface association (to be called either in synchronization phase or to add a new association)
- *cc_scnif_assoc_update()*: update an existent Control Channel/SCN interface association (to be called out of synchronization phase)
- *cc_scnif_assoc_delete()*: delete an existent Control Channel/SCN interface association (to be called out of synchronization phase)

7 G^2.RSVP-TE

The G^2.RSVP-TE module is the RSVP-TE signalling protocol extended with GMPLS TE and Grid-GMPLS extensions. This module implements the I-NNI signalling between G^{2}MPLS nodes and it is responsible for LSPs signalling.

It is compliant with the following IETF RFCs (see D2.1 and D2.2 for details):

- RFC 2205
- RFC 2961
- RFC 3209 / 3210
- RFC 3471
- RFC 3473
- RFC 3474
- RFC 3476
- RFC 3477

The g2rsvpted daemon is not originally part of Quagga routing suite and has been developed from scratch. The G^2.RSVP-TE protocol is integrated into the Quagga framework according to the Quagga daemon main structure (e.g. one master thread to manage the single thread daemon as pseudo multi-thread, the trace log system, the vty interface, etc etc).

Before starting the g2rsvpted daemon must:

- Initialize its own CORBA servants, i.e. NorthBound and TnrController interfaces (see Sec. 7.3).
- Initialize its CORBA clients toward tnrcd, nccd, rcd, g2pcera and lrmd.
- Set up the SCNGW client.

Therefore G^2.RSVP-TE protocol must start after the TNRC, NCC, RC, G^{2}PCERA, LRM and SCNGW modules.

7.1 G².RSVP-TE data model

The G^2.RSVP-TE data model is sketched in Figure 7-1.

The main class is the *G².RSVPTE*, the instance of the protocol, triggered by the VTY command or equivalent internal API. Once the protocol instance is created, is attached at the global G².RSVP-TE *Thread Master,* a singleton class in charge of handling both the G².RSVP-TE protocol instance and the Quagga structures.

This class links a list of interfaces and the various G².RSVP-TE sessions.

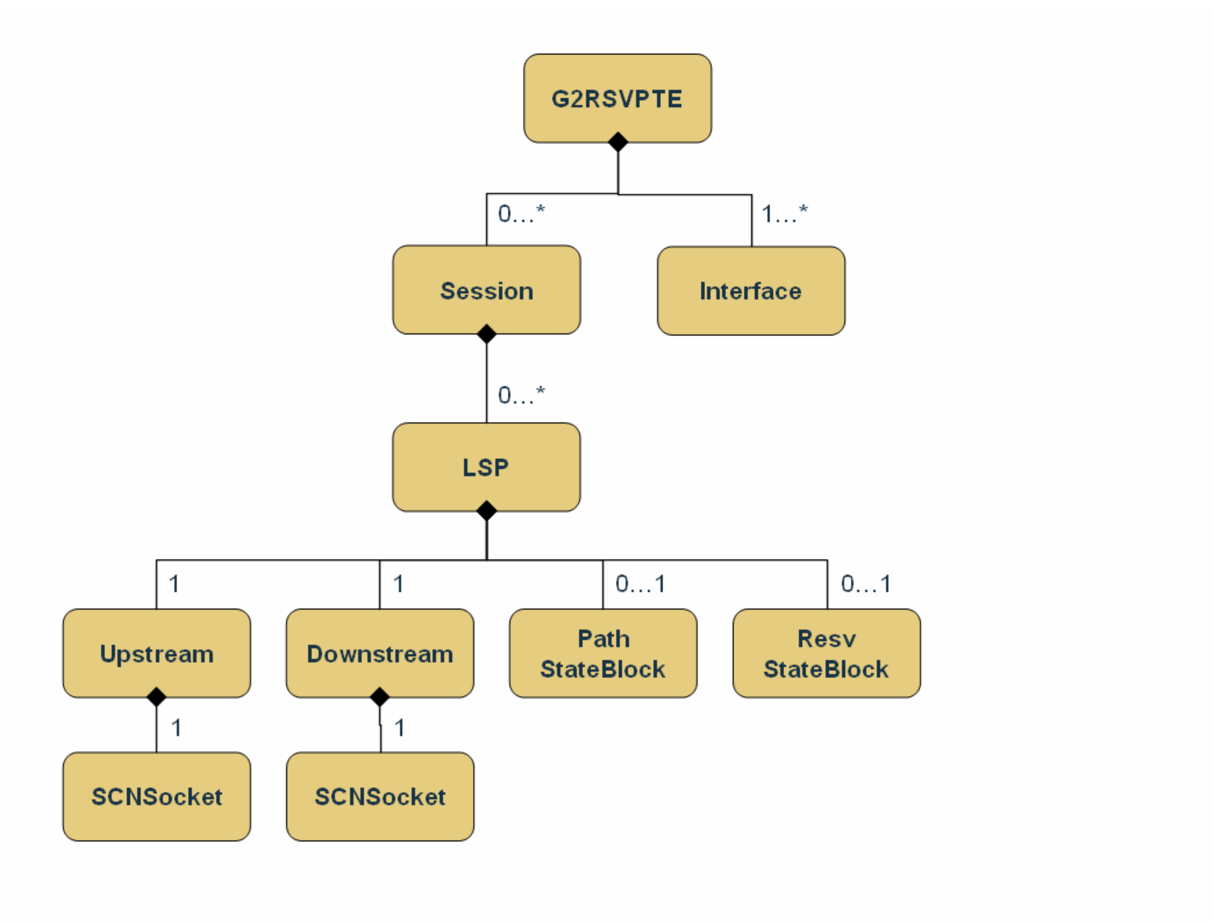

Figure 7-1: The base G².RSVP-TE data model.

7.1.1 G².RSVPTE instance

The *G2RSVPTE* instance is the root of the whole data model. At boot, each node in the network starts a G².RSVP-TE protocol instance and loads all its interfaces from LRM module; each interface is instantiated and attached at the G^2.RSVP-TE instance.

The G².RSVP-TE instance links also a session map to manage a set of LSPs with share a common group of parameters (see Sec. 7.1.2). In fact, when a *createLSP()* is called, the G².RSVP-TE instance checks if a session instance with that *lsp_ident_t* already exists, otherwise, a new session is created and attached at the protocol instance.

```
class G2RSVPTE {
      public:
            G2RSVPTE(void);
            G2RSVPTE(uint32_t defaultRefreshInterval,
                     uint32_t defaultRapidRetransInterval,
                     uint32_t defaultRapidRetryLimit,
                     uint32_t defaultExpoBackoffDelta);
            ~G2RSVPTE(void);

            bool attach(InterfaceKey_t l, Interface * e);
            //bool detach(InterfaceKey_t l);
            bool attach(SessionKey_t k, Session * e);
            bool detach(SessionKey_t k, Session * e);

            // Defines iterator_interfaces
            DEFINE_MAP_ITERATOR(interfaces, Interface);
            // Defines iterator_sessions
            DEFINE_MAP_ITERATOR(sessions,   Session);

            uint32_t nodeId(void);
            void     nodeId(uint32_t id);

            //
            // Session utils
            //
            Session * findSession(SessionKey_t key);

            //
            // Interface utils
            //
            Interface * findInterface(InterfaceKey_t key);
            Interface * findInterface(g2mpls_addr_t  addr,
                                      bool           checkRemote);
            // returns the number of loaded interfaces
            int         loadInterfaces(void);

            //
            // LSP utils
            //
            LSP * findLSP(lsp_ident_t info);
            LSP * createLSP(const lsp_ident_t & ident,
                            Interface *         intf,
```

```
                              Message *           msg);

             LSP * createLSP(const lsp_ident_t &      ident,
                              const std::string &      sessionName,
                              const g2mpls_addr_t &    iTna,
                              const g2mpls_addr_t &    eTna,
                              const sw_cap_t &         swcap,
                              const enc_type_t &       enctype,
                              const gmpls_bwenc_t &    bw,
                              const gpid_t &           gpid,
                              const uint32_t &         setupPrio,
                              const uint32_t &         holdingPrio,
                              const lsp_type_t &       type,
                              const lsp_res_action_t & action,
                              const lsp_rro_mode_t &   rroMode,
                              const uint32_t &         refresh,
                              const bool &             activateAck,
                              const uint32_t &         rapidRetransInterval,
                              const uint32_t &         rapidRetryLimit,
                             const uint32_t &         incrementValueDelta);

             bool  destroyLSP(const lsp_ident_t & id, bool internal = true);

       private:
             std::map<InterfaceKey_t, Interface *> interfaces_;
             std::map<SessionKey_t,   Session *>   sessions_;

             // ...

             uint32_t   nodeId_;                      // router id
             uint32_t   defaultRefreshInterval_;      // refresh interval
             uint32_t   defaultRapidRetransInterval_; // retrans. interval
             uint32_t   defaultRapidRetryLimit_;      // retry limit
             uint32_t   defaultExpoBackoffDelta_;     // incr value delta
};
```

Code 7-1: G2RSVPTE class

7.1.2 Session instance

The *Session* class groups LSPs that share a common:

- Destination Node Id (*nodeId_*)
- Tunnel Id (*tunnelId_*);
- Extended tunnel Id (*extTunId_*).

The relationship with the protocol instance is implemented through the base Ancestor template class.

```
class Session : public Ancestor<true, G2RSVPTE> {
             friend std::ostringstream & operator<< (std::ostringstream & os,
                                                     const Session &      s);
       public:
```

```
		Session(G2RSVPTE * parent);
		Session(G2RSVPTE * parent,
			uint32_t   nId,
			uint32_t   tunId,
			uint32_t   extTunId);
		~Session(void);

		bool     attach(LSPKey_t k, LSP * l);
		bool     detach(LSPKey_t k, LSP * l);

		LSP *    findLSP(LSPKey_t key);

		// return the number of LSPs attached at this session
		uint32_t size(void);
		bool     empty(void);

		// Defines iterator_lsps and methods: begin_lsps/end_lsps
		DEFINE_MAP_ITERATOR(lsps, LSP);

		uint32_t nodeId(void) const;
		void     nodeId(uint32_t id);
		uint32_t tunnelId(void) const;
		void     tunnelId(uint32_t id);
		uint32_t extTunId(void) const;
		void     extTunId(uint32_t id);

	private:
		uint32_t nodeId_;      // Destination Node Id
		uint32_t tunnelId_;    // Tunnel Id
		uint32_t extTunId_;    // Extended Tunnel Id

		std::map<LSPKey_t, LSP *> lsps_;
};
```

Code 7-2: Session class

7.1.3 LSP instance

The *LSP* instance is differentiated from the others by:

- Source Node Id (*nid_*)
- LSP Id (*id_*).

The *LSP* class is the key element of the G²RSVP-TE protocol data model. It has:

- The two LSP identifiers (source node id and LSP id)
- Ingress/Egress termination points info
- A set of flags
- The retransmission and refresh timer values
- The Upstream and Downstream sending message interfaces

- The Path State Block (PSB) and Resv State Block (RSB)
- The LSP FSM instance.

The relationship with the session instance is implemented through the base Ancestor template class.

```
class LSP : public Ancestor<true, Session> {
             friend std::ostringstream & operator<< (std::ostringstream & os,
                                                     const LSP &          l);
      public:
             LSP(Session * parent);
             LSP(Session * parent, uint32_t nId, uint32_t lspId);
             ~LSP(void);

             bool attach(LSPCtrl * ctrl);
             bool attach(UpstreamAckNack * u);
             bool attach(DownstreamAckNack * d);
             bool attach(PSB * psb);
             bool attach(RSB * rsb);

             bool                isEnabled(void) const;
             uint32_t            nid(void) const;
             uint32_t            id(void) const;
             g2mpls_addr_t       iTNA(void) const;
             g2mpls_addr_t       eTNA(void) const;
             PSB *               psb(void);
             RSB *               rsb(void);
             UpstreamAckNack *   usAckNack(void);
             DownstreamAckNack * dsAckNack(void);

             void     iTNA(g2mpls_addr_t addr);
             void     eTNA(g2mpls_addr_t addr);

             std::string sessionName(void) const;

             // Time functions
             uint32_t refreshInterval(void);
             uint32_t rapidRetryLimit(void);
             uint32_t expoBackoffDelta(void);
             uint32_t rapidRetransInterval(void);

             void     refreshInterval(uint32_t time);
             void     rapidRetransInterval(uint32_t time);
             void     rapidRetryLimit(uint32_t time);
             void     expoBackoffDelta(uint32_t time);

             // LSP methods
             bool eroProcess(bool recursive = true);
             bool loopDetect(void);

             // For APIs
             bool signalUpLSP(void);
             bool signalDownLSP(void);
             bool enableLSP(void);
             bool disableLSP(void);
             bool attachEroSubObj(EroSubObject * eroSubObj,
```

```
                                    bool            insertTail = true);
              bool sendPath(bool enqueueEvent = false);
              bool sendResv(bool enqueueEvent = false);
              bool sendDown(bool enqueueEvent = false);
              bool sendTear(bool enqueueEvent = false);
              bool sendConfirm(bool enqueueEvent = false);
              bool xConnCompleted(void);

       private:
              UpstreamAckNack *   us_AckNack_;
              DownstreamAckNack * ds_AckNack_;
              PSB *               psb_;
              RSB *               rsb_;
              uint32_t            nid_;            // source node id
              uint32_t            id_;             // LSP id
              g2mpls_addr_t       ingress_tna_;
              g2mpls_addr_t       egress_tna_;

              // Flags
              LSP_FLAGS           flags_;

              // time intervals
              uint32_t            refreshInterval_;      // refresh interval
              uint32_t            rapidRetransInterval_; // retrans. interval
              uint32_t            rapidRetryLimit_;      // retry limit
              uint32_t            expoBackoffDelta_;     // incr value delta

              // LSP FSM instance
              fsm::G2RSVPTE_LSP_FSM::virtFsm * fsmInst_;
};
```

Code 7-3: LSP class

The LSP flags are:

Flag	short description
G2RSVPTE_FLAG_ENABLED	This LSP is enabled.
G2RSVPTE_FLAG_RECROUTE	The Record Route for this LSP is active (RRO object enable).
G2RSVPTE_FLAG_SIG_ADMIN_DOWN	This LSP is in teardown because of an administrative command.
G2RSVPTE_FLAG_TEAR_DOWN_US	This LSP is in the first signalling tier of teardown.
G2RSVPTE_FLAG_TEAR_DOWN_DS	This LSP is in the second signalling tier of teardown.

Table 7-1: LSP flags.

7.1.3.1 *Upstream/Downstream objects*

The Upstream and Downstream instances inherit from a common interface object that wraps the connection with SCNGW module by means of the SCNSocket class. This class uses the library exposed by SCNGW client to send packets towards others G^2MPLS controllers.

7.1.3.2 *Path/Resv State Block objects*

The Path State Block (PSB) and Resv State Block (RSB) classes inherit directly from the *StateBlock* class according to RFC 2205. The StateBlock class has the following data (in case of PSB, data structures are previous/upstream, whereas in case of RSB, are next/downstream):

- The remote data link, used by next/previous HOP
- The upstream/downstream local data link, used to go to previous/next HOP
- The interface to next/previous HOP
- The next/previous HOP node Id
- The next/previous logical interface handler
- The upstream/downstream label used to transmit to previous/next HOP
- The upstream/downstream label used to receive from previous/next HOP

```
class StateBlock {
       public:
              uint32_t lih(void);
              uint32_t nodeId(void);

              Interface * interface(void);

              g2mpls_addr_t remoteDL(void);
              g2mpls_addr_t localDL(void);

              uint32_t txLabel(void);
              uint32_t rxLabel(void);

              uint32_t refreshTimeout(void);
              uint32_t refreshInterval(void);

       private:
              g2mpls_addr_t remoteDL_;
              g2mpls_addr_t localDL_;
              Interface * intf_;

              uint32_t    nodeId_;
              uint32_t    lih_;

              uint32_t    tx_label_;
              uint32_t    rx_label_;

              uint32_t    refreshTimeout_;  //used Path/Resv refresh timeout
              uint32_t    refreshInterval_; //used Path/Resv refresh interval
```

```
};
Code 7-4: g2rsvpte_dm.h StateBlock class
class PSB : public StateBlock {
       public:
              PSB(void);
              ~PSB(void);

              PathMessage *     pathOut(void);
              PathMessage *     pathIn(void);
              PathErrMessage *  pathErr(void);
              ResvConfMessage * resvConf(void);
              PathMessage *     pathDown(void);
              PathTearMessage * pathTear(void);

       private:
              PathMessage *     pathIn_;   // received Path msg
              PathMessage *     pathOut_;  // transmitted Path msg

              PathErrMessage *  pathErr_;  // TMP rx/tx Path Err msg
              ResvConfMessage * resvConf_; // TMP rx/tx ResvConf msg
              PathMessage *     pathDown_; // TMP rx/tx Path (D=1 R=1) msg
              PathTearMessage * pathTear_; // TMP rx/tx PathTear msg
};

class RSB : public StateBlock {
       public:
              RSB(void);
              ~RSB(void);

              ResvMessage * resvOut(void);
              ResvMessage * resvIn(void);
              ResvMessage * resvDown(void);
              ResvTearMessage * resvTear(void);
              ResvErrMessage * resvErr(void);

       private:
              ResvMessage *     resvIn_;   // received Resv msg
              ResvMessage *     resvOut_;  // transmitted Resv msg

              ResvMessage *     resvDown_; // TMP rx/tx Resv (D=1 R=1) msg
              ResvTearMessage * resvTear_; // TMP rx/tx ResvTear msg
              ResvErrMessage *  resvErr_;  // TMP rx/tx ResvTear msg
};
```

Code 7-5: PSB/RSB classes

7.1.4 Interface instance

The *Interface* class is the data structure that wraps the TE-Link managed by LRM with additional information needed by the G^2.RSVP-TE protocol.

The relationship with the protocol instance is implemented through the base Ancestor template class.

```
class Interface : public Ancestor<true, G2RSVPTE> {
      public:
            Interface(G2RSVPTE * parent);
            Interface(G2RSVPTE *    parent,
                      g2mpls_addr_t localId,
                      g2mpls_addr_t remoteId,
                      uint32_t      nId,
                      opstate_t     op_state,
                      admstate_t    adm_state);
            ~Interface(void);

            g2mpls_addr_t remoteId(void) const;
            void          remoteId(g2mpls_addr_t add);

            g2mpls_addr_t localId(void) const;
            void          localId(g2mpls_addr_t add);

            opstate_t     opState(void) const;
            void          opState(opstate_t state);

            admstate_t    admState(void) const;
            void          admState(admstate_t state);

            uint32_t      neighbourId(void) const;
            void          neighbourId(uint32_t nId);
            void          dump(void) const;

            friend std::ostringstream & operator << (std::ostringstream & os,
                                                    const Interface &    intf);
      private:
            g2mpls_addr_t localId_;
            g2mpls_addr_t remoteId_;
            uint32_t      neighbourId_;
            opstate_t     op_state_;
            admstate_t    adm_state_;
};
```

Code 7-6: Interface classes

7.2 G².RSVP-TE internal API

The internal API is used by the VTY interface and the CORBA G^2.RSVP-TE servants to access the G^2.RSVP-TE data model and functionalities.

The G^2.RSVP-TE internal API is specified in *<sw_root>/g2rsvpted/g2rsvpte_apis.h* and shown below.

```
namespace G2RSVPTE_API {
      RSVP::G2RSVPTE *   g2rsvpteGet(std::string & resp);
      grapiErrorCode_t   g2rsvpteStart(std::string & resp);
      grapiErrorCode_t   g2rsvpteStop(std::string & resp);
};
```

```
namespace G2RSVPTE_GRAPI {
      grapiErrorCode_t  lspCreate(const lsp_ident_t &      key,
                                  const std::string &      sessionName,
                                  const g2mpls_addr_t &    ingressTna,
                                  const g2mpls_addr_t &    egressTna,
                                  const sw_cap_t &         swcap,
                                  const enc_type_t &       enctype,
                                  const gmpls_bwenc_t &    bw,
                                  const gpid_t &           gpid,
                                  const uint32_t &         setupPrio,
                                  const uint32_t &         holdingPrio,
                                  const lsp_type_t &       type,
                                  const lsp_res_action_t & action,
                                  const lsp_rro_mode_t &   rroMode,
                                  const uint32_t &         refresh,
                                  const bool &             activateAck,
                                  const uint32_t &         rapidRetransmInter,
                                  const uint32_t &         rapidRetryLimit,
                                  const uint32_t &         incrementValueDelta,
                                  std::string &            resp);

      grapiErrorCode_t  lspDestroy(const lsp_ident_t &    key,
                                    std::string &          resp);

      grapiErrorCode_t  lspEnable(const lsp_ident_t &     key,
                                   std::string &          resp);

      grapiErrorCode_t  lspDisable(const lsp_ident_t &    key,
                                   std::string &         resp);

      grapiErrorCode_t  lspEroAttach(const lsp_ident_t &              key,
                                      const std::list<lsp_ero_sobj_t>  ero,
                                      std::string &                    resp);

      grapiErrorCode_t  lspEroDetach(const lsp_ident_t &              key,
                                      const std::list<lsp_ero_sobj_t>  ero,
                                      std::string &                    resp);

      grapiErrorCode_t  lspSendPath(const lsp_ident_t &        key,
                                     std::string &              resp);

      grapiErrorCode_t  lspSendResv(const lsp_ident_t &        key,
                                     std::string &              resp);

      grapiErrorCode_t  lspSendConfirm(const lsp_ident_t &     key,
                                        std::string &           resp);

      grapiErrorCode_t  lspSendDown(const lsp_ident_t &        key,
                                     std::string &              resp);

      grapiErrorCode_t  lspSendTear(const lsp_ident_t &        key,
                                     std::string &              resp);

      grapiErrorCode_t  lspForceUp(const lsp_ident_t &         key,
                                    std::string &               resp);

      grapiErrorCode_t  lspForceDown(const lsp_ident_t &       key,
                                      std::string &             resp);
```

```
        grapiErrorCode_t  lspXConnCompleted(const lsp_ident_t &  key,
                                       std::string &        resp);

        grapiErrorCode_t  getLsps(std::list<lsp_ident_t> &       lsps,
                           std::string &                 resp);

        grapiErrorCode_t  lspGetDetails(const lsp_ident_t &      key,
                                  lsp_param_t &            params,
                                    std::string &            resp);
};
```

Code 7-7: Internal API

The internal G².RSVPTE_API functions are:

- *g2rsvpteGet():* allows to get the G².RSVP-TE protocol instance reference.
- *g2rsvpteStart():* allows to create and start a G².RSVP-TE protocol instance.
- *g2rsvpteStop():* allows to stop and delete the G².RSVP-TE protocol instance.

The internal G2RSVPTE_GRAPI functions are:

- *lspCreate()*: allows to create an LSP instance with the specified LSP identity and parameter attribute.
- *lspDestroy()*: allows to destroy the LSP identified by *lsp_ident* if this LSP is disabled.
- *lspEnable()*: allows to enable the specified LSP.
- *lspDisable()*:allows to disable the specified LSP.
- *lspEroAttach()*: allows to attach the list of ERO sub objects at the specified LSP.
- *lspEroDetach()*:allows to detach the list of ERO sub objects from the specified LSP.
- *lspSendPath()*: prepares the G².RSVP-TE Path Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendPath* event to the FSM of the specified LSP.
- *lspSendResv()*: prepares the G².RSVP-TE Resv Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendResv* event to the FSM of the specified LSP.
- *lspSendConfirm()*: prepares the G².RSVP-TE Resv Confirm Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendConfirm* event to the FSM of the specified LSP.
- *lspSendDown()*: allows to start the tear down G².RSVP-TE signalling procedure on the specified LSP if the G²MPLS Controller is the head node of this LSP; otherwise, if the node is the tail of this LSP it checks if the RSB is consistent, prepares the G².RSVP-TE Resv Down Message (Resv Message with the Deletion flag set) to be sent and sends a *G2RSVPTE_LSP_FSM_SendResvDown* event to the FSM of this LSP.
- *lspSendTear()*: prepares the G².RSVP-TE PathTear Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendPathTear* event to the FSM of the specified LSP if the G²MPLS Controller is the head node of this LSP; otherwise, if the node is the tail of this LSP it prepares the G².RSVP-TE Resv Tear Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendResvTear* event to the FSM of this LSP.

- *lspForceUp()*: checks if the LSP is enabled, if the PSB is consistent, prepares the G².RSVP-TE Path Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendPath* event to the FSM of the specified LSP, triggering the G².RSVP-TE signalling procedure for that LSP.
- *lspForceDown()*: if the G²MPLS Controller is the head node of the specified LSP, checks if the PSB is consistent, prepares the G².RSVP-TE Path Down Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendPathDown* event to the FSM of this LSP; on the contrary, if the G²MPLS Controller is the tail node of the specified LSP, checks if the RSB is consistent, prepares the G².RSVP-TE Resv Down Message to be sent and sends a *G2RSVPTE_LSP_FSM_SendResvDown* event to the FSM of this LSP.
- *lspXConnCompleted()*: allows to send a *G2RSVPTE_LSP_FSM_XConnCompleted* event to the FSM of the specified LSP.
- *getLsps()*: returns the list of LSPs.
- *lspGetDetails()*: allows to retrieve all the parameters for the specified LSP.

7.3 G².RSVP-TE external API

The G².RSVP-TE module exposes its interface by means of CORBA servants. Its API for the communication with external modules is specified in the *<sw_root>/idl/g2rsvpte.idl* and shown below.

```
#include "types.idl"
#include "g2mplsTypes.idl"

module g2rsvpte {

      interface NorthBound {
            boolean
            lspCreate(in g2mplsTypes::lspIdent                    lspId,
                      in g2mplsTypes::callIdent                   callId,
                      in g2mplsTypes::lspParams                   lspInfo,
                      in g2mplsTypes::recoveryParams              recoveryInfo,
                      in boolean                                  setup)
                  raises(Types::InternalProblems);

            boolean
            lspAddEroPart(in g2mplsTypes::lspIdent        lspId,
                          in g2mplsTypes::eroSeq          eroItem)
                  raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspDelEroPart(in g2mplsTypes::lspIdent        lspId,
                          in g2mplsTypes::eroSeq          eroItem)
                  raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspEnable(in g2mplsTypes::lspIdent                    lspId)
                  raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspDisable(in g2mplsTypes::lspIdent                   lspId)
```

```
                    raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspDestroy(in g2mplsTypes::lspIdent                     lspId)
                    raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspSetUp(in g2mplsTypes::lspIdent                       lspId)
                    raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            lspSetDown(in g2mplsTypes::lspIdent                     lspId)
                    raises(Types::InternalProblems, Types::CannotFetch);

            typedef sequence<g2mplsTypes::lspIdent>        lspIdentSeq;
            lspIdentSeq getLsps()
                    raises(Types::InternalProblems);

            boolean
            lspGetDetails(in g2mplsTypes::lspIdent         lspId,
                          out g2mplsTypes::callIdent              callId,
                          out g2mplsTypes::lspParams              lspInfo,
                          out g2mplsTypes::recoveryParams recoveryInfo,
                          out g2mplsTypes::statesBundle   states)
                    raises(Types::InternalProblems, Types::CannotFetch);
    };

    interface TnrControl {
            void
            actionResponse(in Types::uint32           cookie,
                           in g2mplsTypes::tnrcResult result,
                           in long                    responseCtxt)
                    raises(Types::InternalProblems);

            void
            actionNotify(in Types::uint32             cookie,
                         in g2mplsTypes::tnrcEvent    event,
                         in long                      notifyCtxt)
                    raises(Types::InternalProblems);
    };
};
```

Code 7-8: G².RSVP-TE external API IDL.

The g2rsvpted exposes the G².RSVP-TE internal API to g2pcerad, nccd and rcd daemons through the *NorthBound* interface, and exposes callback-like interfaces to tnrcd through the *TnrControl* interface.

The *NorthBound* methods are mapped 1:1 with the G^2.RSVP-TE internal API as shown in Figure 7-1.

External API	Internal API
NorthBound::lspCreate()	G2RSVPTE_GRAPI::lspCreate()
NorthBound::lspAddEroPart()	G2RSVPTE_GRAPI::lspEroAttach()

NorthBound::lspDelEroPart()	G2RSVPTE_GRAPI::lspEroDetach()
NorthBound::lspEnable()	G2RSVPTE_GRAPI::lspEnable()
NorthBound::lspDisable()	G2RSVPTE_GRAPI::lspDisable()
NorthBound::lspDestroy()	G2RSVPTE_GRAPI::lspDestroy()
NorthBound::lspSetUp()	G2RSVPTE_GRAPI::lspForceUp()
NorthBound::lspSetDown()	G2RSVPTE_GRAPI::lspForceDown()
NorthBound::getLsps()	G2RSVPTE_GRAPI::getLsps()
NorthBound::lspGetDetails()	G2RSVPTE_GRAPI::lspGetDetails()

Table 7-1: Mapping between internal and external G².RSVP-TE API

The *TnrControl* interface methods are like asynchronous callbacks with the following behaviour:

- *actionResponse()*: allows the TNRC to deliver the result of the operation (identified by the *cookie*) previously requested by the G^2.RSVP-TE.
- *actionNotify()*: allows the TNRC to deliver an asynchronous notification about the specified operation to the G^2.RSVP-TE.

7.4 G².RSVP-TE LSP FSM

The main element of the Phosphorus G².RSVP-TE is the LSP, which is controlled across the signalling phases of the protocol with a specific finite state machine. The LSP FSM tracks the creation and installation phase of an LSP on a G².RSVP-TE instance. The LSP is the result of a 2(3)-signalling tiers, i.e. Path-Resv (Path-Resv-ResvConf). The FSM states and root events are explained in Table 7-2 and Table 7-3, while the overall FSM picture with the transition events between states are shown in.Figure 7-2.

```
#
#  G2RSVP-TE LSP FSM definition
#

{ FSM }

name = G2RSVPTE_LSP_FSM
definition-file = g2rsvpte_lsp.def
# If graphviz-file is defined the graphviz file will be create
graphviz-file = g2rsvpte_lsp.dot
include-name = g2rsvpte_lsp.h
```

```
#start-state = Down [optional]

#
# Events
#
#
# rootEvent = derivedEvent1, derivedEvent2, ...
#

{ Events }       ################### REMOVED Net postfix from rootEvents

RecvPath         =  evRecvPathOk,       evRecvPathKo
RecvPathDown     =  evRecvPathDownOk,  evRecvPathDownKo
RecvResvDown     =  evRecvResvDownOk,  evRecvResvDownKo
RecvResv         =  evRecvResvOk,       evRecvResvKo,      evRecvResvVeryKo
RecvConfirm      =  evRecvConfirmOk,   evRecvConfirmKo
RecvPathTear     =  evRecvPathTearOk,  evRecvPathTearKo
RecvResvTear     =  evRecvResvTearOk,  evRecvResvTearKo
RecvPathErr      =  evRecvPathErrOk,   evRecvPathAlarm,   evRecvPathErrCrankback,
evRecvActivateErr, evRecvPathErrKo
RecvResvErr      =  evRecvResvErrOk,   evRecvResvErrKo
RecvNotify       =  evRecvNotifyOk,    evRecvNotifyDown, evRecvNotifyKo
RecvActivate     =  evRecvActivateOk,  evRecvActivateKo
RecvPathTimer    =  evRecvPathTimer
RecvResvTimer    =  evRecvResvTimer
RecvPathTimeout  =  evRecvPathTimeout
RecvResvTimeout  =  evRecvResvTimeout
SendPath         =  evSendPath
SendResv         =  evSendResv
SendConfirm      =  evSendConfirm
SendPathDown     =  evSendPathDown
SendResvDown     =  evSendResvDown
SendPathTear     =  evSendPathTear
SendResvTear     =  evSendResvTear
#SendActivate     =  evSendActivate
XConnCompleted   =  evXConnCompleted
XConnErr         =  evXConnErr
XConnDown        =  evXConnDown
XConnPreempt     =  evXConnPreempt

#
# States
#
# state = state1   [The first state is the start one if start-state is not set]
#       eventX -> dstState
#
# state = state2
#       eventY -> dstState
#

{ States }

#
State = Down
      evRecvPathOk -> PathReceived
      evRecvPathKo -> Down
      evSendPath   -> PathReceived
```

```
#
State = PathReceived      ########### WAS PathProcessed
      evRecvPathOk            -> PathReceived
      evRecvPathKo            -> PathReceived
      evRecvResvOk            -> WaitEqptConf
      evRecvResvKo            -> PathReceived
#     evRecvResvVeryKo         -> Down
      evRecvPathTearOk        -> Down
      evRecvPathTearKo        -> PathReceived
      evRecvPathErrOk         -> Down
      evRecvPathErrKo         -> PathReceived
#     evRecvPathAlarm          -> Down
      evRecvPathErrCrankback -> PathReceived
      evSendResv              -> WaitEqptConf
      evSendPathTear          -> Down
      evSendResvTear          -> Down
      evRecvPathTimer         -> PathReceived
      evRecvPathTimeout       -> Down
      evXConnErr              -> Down
      evXConnCompleted        -> WaitResv

#
State = WaitEqptConf ########### WAS            ResourceWait
      evRecvPathOk            -> WaitEqptConf
      evRecvPathKo            -> WaitEqptConf
      evRecvResvOk            -> WaitEqptConf
      evRecvResvKo            -> WaitEqptConf
      evRecvResvVeryKo        -> Down
      evRecvPathTearOk        -> Down
      evRecvPathTearKo        -> WaitEqptConf
      evRecvResvTearOk        -> Down
      evRecvResvTearKo        -> WaitEqptConf
      evRecvPathErrOk         -> Down
      evRecvPathErrKo         -> WaitEqptConf
#     evRecvPathAlarm          -> Down
#     evRecvPathErrCrankback   -> WaitEqptConf
      evRecvResvErrOk         -> WaitEqptConf
      evRecvResvErrKo         -> WaitEqptConf
      evSendPathTear          -> Down
      evSendResvTear          -> Down
      evRecvPathTimer         -> WaitEqptConf
      evRecvPathTimeout       -> Down
      evRecvResvTimer         -> WaitEqptConf
      evRecvResvTimeout       -> Down
      evXConnErr              -> PathReceived
      evXConnCompleted        -> WaitResvConf

#
State = WaitResv
      evRecvPathOk            -> WaitResv
      evRecvPathKo            -> WaitResv
      evRecvResvOk            -> WaitResvConf
      evRecvResvKo            -> WaitResv
      evRecvResvVeryKo        -> Down
      evRecvPathTearOk        -> Down
      evRecvPathTearKo        -> WaitResv
      evRecvResvTearOk        -> Down
      evRecvResvTearKo        -> WaitResv
```

```
        evRecvPathErrOk         -> Down
        evRecvPathErrKo         -> WaitResv
        evRecvPathAlarm         -> Down
        evRecvPathErrCrankback  -> PathReceived
#       evRecvResvErrOk          -> WaitResv
#       evRecvResvErrKo          -> WaitResv
        evSendResv              -> WaitResvConf
        evSendPathTear          -> Down
        evSendResvTear          -> Down
        evRecvPathTimer         -> WaitResv
        evRecvPathTimeout       -> Down
        evXConnPreempt          -> WaitResv

#
State = WaitResvConf               ################## WAS    WaitConf
        evRecvPathOk            -> WaitResvConf
        evRecvPathKo            -> WaitResvConf
        evRecvPathDownOk        -> TearDown
        evRecvPathDownKo        -> WaitResvConf
        evRecvResvOk            -> WaitResvConf
        evRecvResvKo            -> WaitResvConf
        evRecvConfirmOk         -> Installed
        evRecvConfirmKo         -> WaitResvConf
        evRecvPathTearOk        -> Down
        evRecvPathTearKo        -> WaitResvConf
        evRecvResvTearOk        -> Down
        evRecvResvTearKo        -> WaitResvConf
        evRecvPathErrOk         -> Down
        evRecvPathErrKo         -> WaitResvConf
        evRecvPathAlarm         -> Down
        evRecvResvErrOk         -> WaitResvConf
        evRecvResvErrKo         -> WaitResvConf
        evSendConfirm           -> Installed
        evSendPathTear          -> Down
        evSendResvTear          -> Down
        evRecvPathTimer         -> WaitResv
        evRecvResvTimer         -> WaitResv
        evXConnPreempt          -> WaitResvConf

#
State = Installed                ###########  WAS Active
        evRecvPathOk            -> Installed
        evRecvPathKo            -> Installed
        evRecvPathDownOk        -> TearDown
        evRecvPathDownKo        -> Installed
        evRecvResvOk            -> Installed
        evRecvResvKo            -> Installed
        evRecvResvVeryKo        -> Down
        evRecvResvDownOk        -> TearDown
        evRecvResvDownKo        -> Installed
        evRecvConfirmOk         -> Installed
        evRecvConfirmKo         -> Installed
        evRecvNotifyOk          -> Installed
        evRecvNotifyDown        -> Down
        evRecvNotifyKo          -> Installed
        evRecvPathTearOk        -> Down
        evRecvPathTearKo        -> Installed
        evRecvResvTearOk        -> Down
```

```
      evRecvResvTearKo       -> Installed
      evRecvPathErrOk        -> Down
      evRecvPathErrKo        -> Installed
      evRecvPathAlarm        -> Installed
      evRecvResvErrOk        -> Installed
      evRecvResvErrKo        -> Installed
      evRecvActivateOk       -> Installed
      evRecvActivateKo       -> Installed
      evRecvActivateErr      -> Installed
      evSendPathDown         -> TearDown
      evSendResvDown         -> TearDown
#      evSendActivate         -> Installed
      evRecvPathTimer        -> Installed
      evRecvResvTimer        -> Installed
      evXConnErr             -> Installed
      evXConnCompleted       -> Installed
      evXConnDown            -> Installed
      evXConnPreempt         -> Installed

#
State = TearDown                ############ WAS WaitTear
      evRecvPathDownOk       -> TearDown
      evRecvPathDownKo       -> TearDown
      evRecvResvDownOk       -> TearDown
      evRecvResvDownKo       -> TearDown
      evRecvPathTearOk       -> Down
      evRecvPathTearKo       -> TearDown
      evRecvResvTearOk       -> Down
      evRecvResvTearKo       -> TearDown
      evRecvPathErrOk        -> Down
      evRecvPathErrKo        -> TearDown
      evRecvPathAlarm        -> TearDown
      evRecvResvErrOk        -> TearDown
      evRecvResvErrKo        -> TearDown
      evSendPathDown         -> TearDown
      evSendResvDown         -> TearDown
      evSendPathTear         -> Down
      evSendResvTear         -> Down
      evRecvPathTimeout      -> Down
      evRecvResvTimeout      -> Down
```

Code 7-9: G².RSVP-TE LSP FSM.

State	short description
Down	The LSP instance is created but no action or message has been received yet, or the LSP has been torn down and it is going to be completely deleted in the protocol instance.
PathReceived	The first or refresh Path has been received (downstream node) or sent (upstream node) during the early phases of the signalling.
WaitEqptConf	The Resv has been received (upstream node) or sent (downstream node) but the equipment is still working on the implementation of the requested configuration.
WaitResv	The equipment implemented the requested configuration and the protocol is waiting a Resv for this LSP.

WaitResvConf	The protocol is waiting a ResvConf for this LSP (3-tiers signalling).
Installed	The 2(3)-signalling tiers have been completed successfully and the reservation session is installed. Traffic is ok.
TearDown	A Tear Down message has been received/sent (Path or Resv with ADMIN_STATUS) and the LSP is waiting for the completion of the deletion signalling flow.

Table 7-2: G².RSVP-TE LSP FSM: states

Root event	short description
RecvPath	A G².RSVP-TE Path Message has been received.
RecvPathDown	A G².RSVP-TE Path Message with ADMIN_STATUS has been received.
RecvResvDown	A G².RSVP-TE Resv Message with ADMIN_STATUS has been received.
RecvResv	A G².RSVP-TE Resv Message has been received.
RecvConfirm	A G².RSVP-TE Resv Confirm Message has been received.
RecvPathTear	A G².RSVP-TE PathTear Message has been received.
RecvResvTear	A G².RSVP-TE Resv Tear Message has been received.
RecvPathErr	A G².RSVP-TE PathErr Message has been received.
RecvResvErr	A G².RSVP-TE ResvErr Message has been received.
RecvNotify	A G².RSVP-TE Notify Message has been received.
RecvPathTimeout	A G².RSVP-TE Path Timeout has occurred.
RecvResvTimeout	A G².RSVP-TE Resv Timeout has occurred.
SendPath	A G².RSVP-TE Path Message must be sent.
SendResv	A G².RSVP-TE Resv Message must be sent.
SendConfirm	A G².RSVP-TE Resv Confirm Message must be sent.
SendPathDown	A G².RSVP-TE Path Message with ADMIN_STATUS must be sent.
SendResvDown	A G².RSVP-TE Resv Message with ADMIN_STATUS must be sent.
SendPathTear	A G².RSVP-TE PathTear Message must be sent.

SendResvTear	A G².RSVP-TE Resv Tear Message must be sent.
XConnCompleted	The coss connection has been completed.
XConnErr	The requested coss connection has been failed.
XConnDown	The cross connection has gone down.
XConnPreempt	The cross connection has been preempted.

Table 7-3: G2RSVP-TE LSP FSM: root events

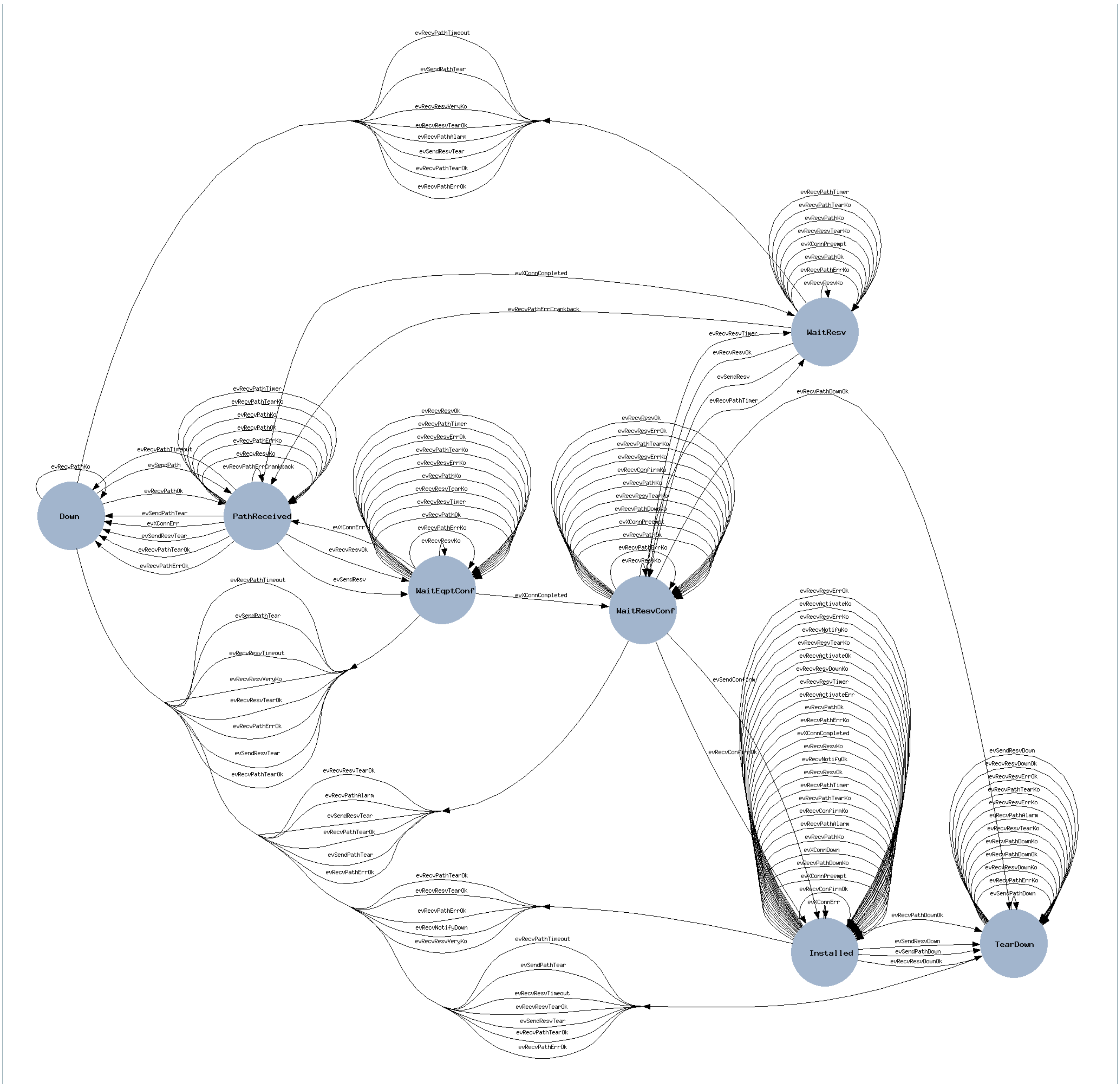

Figure 7-2: G²MPLS LSP finite state machine

7.4.1 Example transitions

The Figure 7-3 shows an example of LSP signal up. The highlighted line represents events and transitions for the Ingress node, whereas the dotted line represents events and transitions for the Egress node.

In this picture the TNRC notify (*evXConnCompleted* event) has been received after the *evRecvResv / evSendResv* event.

Independently from the current state, in case of an error event, the FSM comes back to *Down* state.

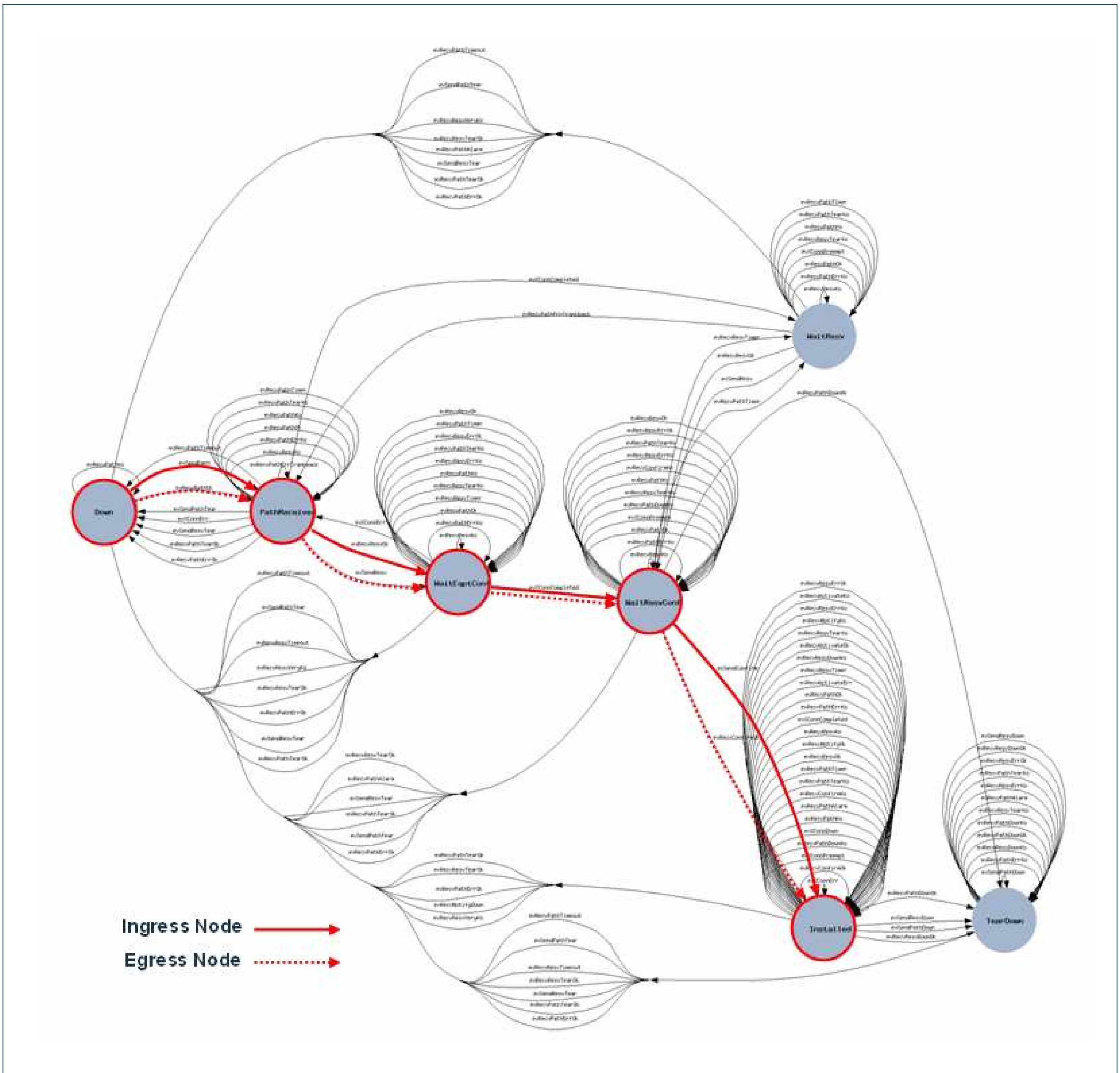

Figure 7-3: Example of G^{2}MPLS LSP signalling setup

7.5 G².RSVP-TE parsing and formatting

The G².RSVP-TE parsing and formatting is based on the serialization and de-serialization of the internal message and object classes by means of the stream operators as shown in Figure 7-4.

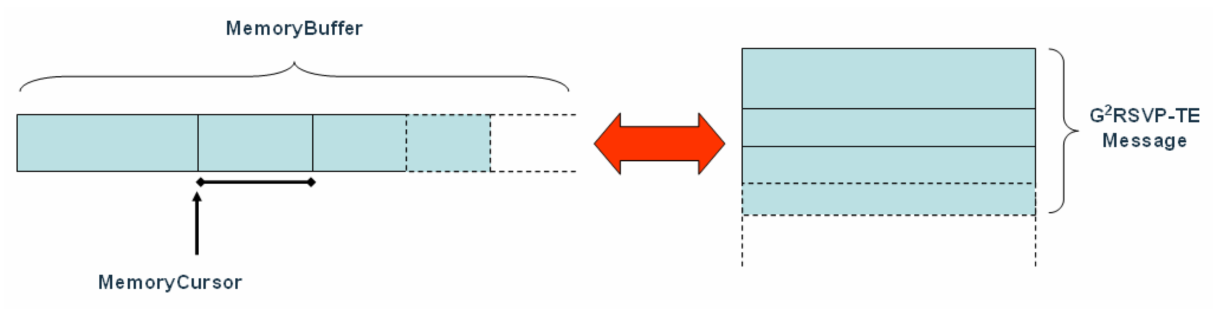

Figure 7-4: parsing and formatting sketch.

The *MemoryBuffer* class is the main data structure both for parsing and formatting functions. The *MemoryStream* class, that has one instance of the *MemoryBuffer*, is used to convert a buffer into a G².RSVP-TE Message.

The parsing phase is described by the following steps:

- When the buffer is received from the SCNGW module, a MemoryBuffer is created.
- A MemoryStream object is instantiated from the MemoryBuffer.
- The stream operator of the MemoryStream is used to create a G².RSVP-TE Message.

The formatting phase is described by the following steps:

- When the G².RSVP-TE Message is ready to be sent a MemoryStream object is created from the message by means of stream operator.
- The MemoryStream object has a MemoryBuffer instance created from the data of the G².RSVP-TE Message.
- The raw data into MemoryBuffer are sent to SCNGW module.

The *MemoryCursor* class is an helper object to make easier the serialisation and de-serialisation from MemoryBuffer to G².RSVP-TE Message and vice-versa. It has all the functions and utilities to get/set data from/to MemoryBuffer object as shown in Code 7-10, Code 7-11 and Code 7-12.

```
extern MemoryCursor & operator << (MemoryCursor &, const Message &);
extern MemoryCursor & operator >> (MemoryCursor &, Message * &);

class MemoryCursor {
       friend class MemoryBuffer;
       friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                              ipv4_t &       data);
```

```
        friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                           ipv6_t &       data);
        friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                           uint8_t &      data);
        friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                           uint16_t &     data);
        friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                           uint32_t &     data);
        friend MemoryCursor & operator >> (MemoryCursor & cursor,
                                           uint64_t &     data);
        friend MemoryCursor & operator >> (MemoryCursor & mc,
                                           Message * &    msg);

        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const ipv4_t   data);
        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const ipv6_t   data);
        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const uint8_t  data);
        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const uint16_t data);
        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const uint32_t data);
        friend MemoryCursor & operator << (MemoryCursor & cursor,
                                           const uint64_t data);
        friend MemoryCursor & operator << (MemoryCursor &  cursor,
                                           const Message & msg);

        friend std::ostream & operator << (std::ostream &       os,
                                           const MemoryCursor & mb);
public:
        MemoryCursor(MemoryBuffer * buffer,
                      size_t         start,
                     size_t         stop);
        ~MemoryCursor(void);

        MemoryCursor & resize(size_t start, size_t stop);
        MemoryCursor neighbor(size_t len);
        size_t remainingSize(void);

        size_t start(void);
        size_t stop(void);

private:
        // pointer to main buffer for this family of cursors
        MemoryBuffer * buffer_;

        // buffer index (range [0, size -1])
        size_t start_;
        size_t stop_;

        // Cursor index: range [0, stop_ - start_ + 1]
        // if (current == stop_ - start_ + 1)  =>  buffer is full
        size_t current_;
};
```

Code 7-10: MemoryCursor class

```
class MemoryBuffer {
        friend std::ostream & operator << (std::ostream &       os,
                                           const MemoryBuffer & mb);
public:
        MemoryBuffer(uint8_t * buffer, size_t size);
        MemoryBuffer(size_t size);
        ~MemoryBuffer(void);

        MemoryCursor & cursor(void);
        void print(std::ostream & os, size_t start, size_t stop) const;
        size_t size(void) const;
        const uint8_t * getData(void);

        // Checksum utils
        uint16_t calculateChecksum(void) const;
        void     writeChecksum(void);
        bool     isChecksumOk(void);

        // offset MUST have a range [0, size -1]
        WOP(w8,  uint8_t,  HTONC);  // void  w8(size_t off, uint8_t d);
        WOP(w16, uint16_t, htons);
        WOP(w32, uint32_t, htonl);
        WOP(w64, uint64_t, htonll);
        WOP(w32_addr, ipv4_t, HTONC);

        ROP(r8,  uint8_t,  NTOHC);  // uint8_t r8(size_t off);
        ROP(r16, uint16_t, ntohs);
        ROP(r32, uint32_t, ntohl);
        ROP(r64, uint64_t, ntohll);
        ROP(r32_addr, ipv4_t, NTOHC);

private:
        uint8_t * buffer_;
        size_t    size_;
};
```

Code 7-11: MemoryBuffer class

```
class MemoryStream {
        friend MemoryStream & operator >> (MemoryStream & ms,
                                           Message * &    msg);
        friend MemoryStream & operator << (MemoryStream &  ms,
                                           const Message & msg);
        friend std::ostream & operator << (std::ostream &       os,
                                           const MemoryStream & ms);
public:
        MemoryStream(void);
        MemoryStream(uint8_t * buffer, size_t size);
        ~MemoryStream(void);

        void flushBuffer(void);
        const uint8_t * getBufferData(void);
        size_t          getBufferDataSize(void) const;

private:
        MemoryBuffer * buffer_;
};
```

Code 7-12: MemoryStream class

All the G².RSVP-TE protocol Messages, Objects and SubObjects have their own functions and the following mandatory interfaces:

- stream operator
- set/get to set/get protocol data
- isConsistent method to check the consistency of the packets according to the standard.

8 Call Controllers

8.1 CC shared objects and functions (xCC)

The xCC shim software implements a set of common objects (Python classes) and methods that are used (as is) or extended/replaced by the G².NCC and G².CCC protocols.

The xCC is implemented in Python 2.5 (code in <sw_root>/pyg2mpls/xcc/), and works in a real multi-threaded environment (as compared to “fake” Quagga threads).

The xCC is based on a set of legacy Python modules, plus a number of modules purposely developed for the Phosphorus-WP2 G²MPLS project. These modules are listed in the following:

- legacy ones (see docs about each module at http://docs.python.org/lib/module-<*module-name*>.html, unless specified differently):
    - os
    - signal
    - sys
    - time
    - re
    - thread
    - threading
    - traceback
    - socket
    - *xml* (for Python ≥ 2.5) or *elementtree* (for Python < 2.5) (http://docs.python.org/lib/module-xml.etree.ElementTree.html)
    - *omniORB* and *omniorbpy* (http://omniorb.sourceforge.net/)
- developed for the Phosphorus-WP2 G²MPLS project (see section 14.4 for details):
    - baseobj
    - bits
    - corbahelper

- o fsm
- o logger
- o netutils
- o protocol
- o timer
- o udpcomm
- o version
- o xmlmsg
- o g2types

The xCC modules are composed of:

- *ccdm.py*: the base xCC data model
- *ccsrv.py*: xCC CORBA servant, for both the G².NCC and the G².CCC (the deviations in behaviour are introduced by the specific classes)
- *ccsigif.py*: xCC signalling interface wrapper and XML implementation

8.1.1 xCC data model

The xCC data model is depicted in Figure 8-1.

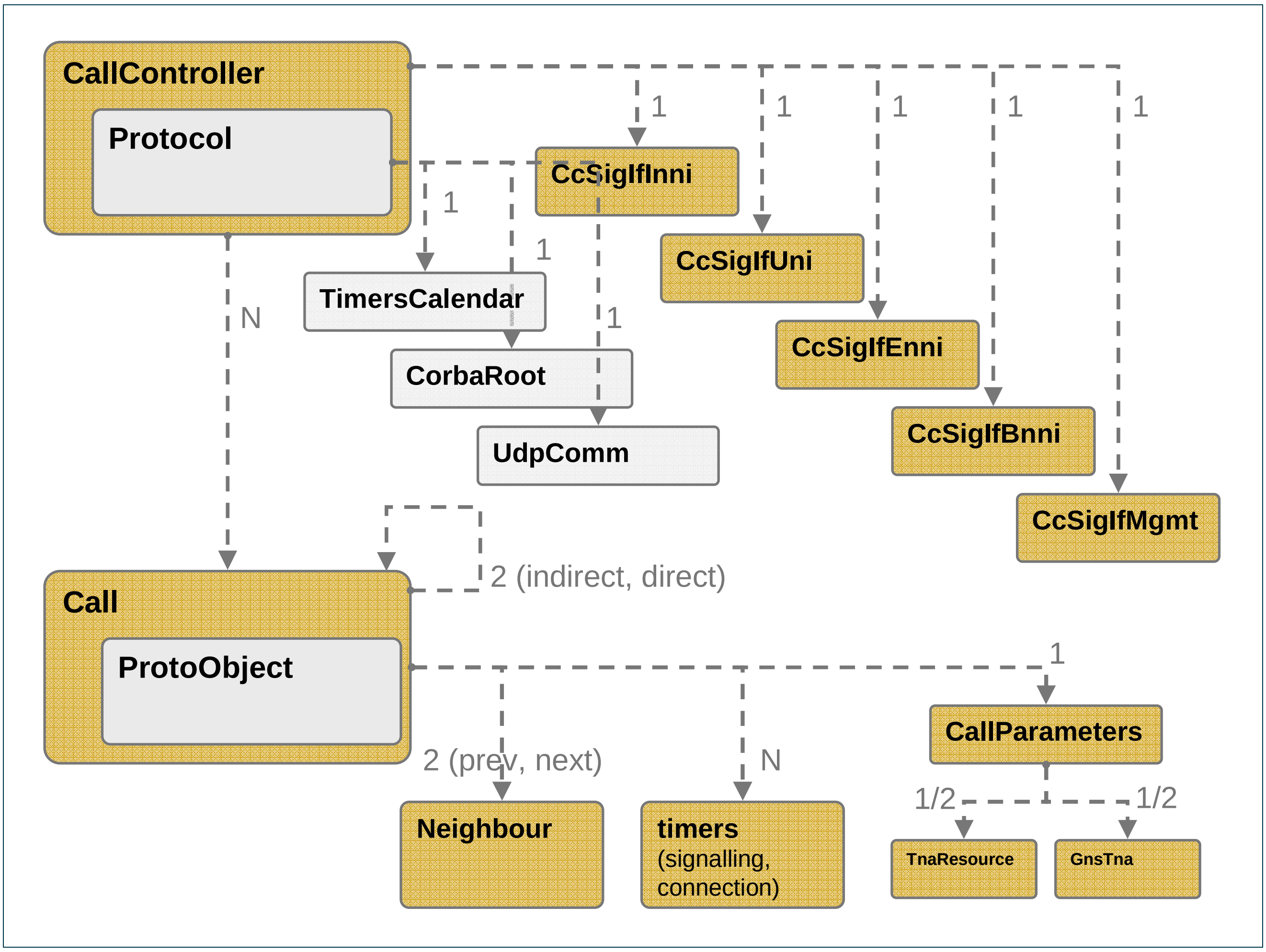

Figure 8-1: The base xCC data model

The main class is the CallController, which inherits directly from the Protocol class in the module protocol. This class has a number of direct descendants (the CcSigIf<i/f> classes) and indirect descendants (inherited from Protocol): the TimersCalendar, the CorbaRoot (with CORBA client and servants under it) and the UdpComm classes.

The Call class is the core item for implementing the call data and behaviour, and links to:

- a couple of *Neighbour* classes: the previous (aka upstream) and next (aka downstream) Call Controller (either CCC or NCC) with respect to the direction of call setup (from the initiator to the receiver)
- a number of timers, for both signalling (expiration timers on call setup, in order to clean states if the call setup doesn't converge in a period of time) and connection (aka LSP) setup (this is for NCC only)

- a number of sub-parameters. Worth to be highlighted, the *CallParameters* class, which links to the call endpoints (either a legacy *Tna* or a G² *GnsTna*).

8.1.2 xCC (CCC/NCC) External API

The API for both the CCC and NCC is specified in *<sw_root>/idl /CallController.idl*, and reported in Code 8-1. The API has two CORBA interfaces: *Mgmt* and *SouthBound.*

The *Mgmt* interface allows to perform management-like operations on the CCC or NCC. In particular, the foreseen usage scenarios for this interface are:

- Dynamic call creation and setup by the grid MW. In this case, the *Mgmt* methods at the CCC-a are invoked by the G.UNI GW, that maps grid job requests from the MW into G² Calls.
- SPC Calls. In this case, the *Mgmt* methods at the NCC-1 are invoked by some NMS.
- Command-Line Interface. The *Mgmt* methods are invoked by the implementation of the CCC or NCC CLI (VTY, see section 14)

The *SouthBound* interface is used for the interactions between the Call Controller and the underlying Recovery Controller, in the upward direction. Its main function is to allow the Recovery Controller to notify the Call Controller about events regarding the recovery bundles (each attached to a Call in the Call Controller domain).

```
#include "types.idl"
#include "g2mplsTypes.idl"


module CallController {

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

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

boolean
callDisable(in g2mplsTypes::callIdent               id,
          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
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);

            boolean
            callGetGnsTna(in g2mplsTypes::callIdent             id,
                        in  g2mplsTypes::resourcePosition       pos,
                        out g2mplsTypes::gridParams             gnsTna)
                  raises(Types::InternalProblems, Types::CannotFetch);

            boolean
            getRecoBundleDetails(in g2mplsTypes::recoBundleIdent id,
                             out g2mplsTypes::recoveryParams    info,
                             out g2mplsTypes::statesBundle      states,
                             out lspIdentSeq                    lsps)
                  raises(Types::InternalProblems, Types::CannotFetch);
      };

      interface SouthBound {
            enum callEvent {
                  CALLEVENT_CONN_READY,
                  CALLEVENT_CONN_FAILED_UP,
                  CALLEVENT_CONN_FAILED_DOWN,
                  CALLEVENT_CONN_DELETED
            };

            boolean getNotification(in g2mplsTypes::callIdent   id,
                                  in callEvent                  event)
                  raises(Types::InternalProblems);
      };
};
```

Code 8-1: CallController.idl: CCC and NCC API

The methods for the *Mgmt* interface are:

- *callCreate()*: allows to create a new call at the CCC-a or NCC-1; in case of NCC-1, this is the door for setting up an SPC Call.
- *callSetTna()*: allows to specify a legacy TNA resource (TNA, + Data Link, + Label) (ingress or egress) for the newly created Call (it has to be still "Idle").
- *callSetGnsTna()*:allows to specify a GNS TNA (ingress or egress) for the newly created Call (it has to be still "Idle").
- *callAddEroPart()*: allows to add a piece of Explicit Route to the newly created Call (it has to be still "Idle"). The Call ERO allows to specify the sequence of domains (i.e. NCCs) to be traversed by the

Call; each Call ERO element is a standard RSVP ERO, and the NCCs along the path are identified by, either:

  - their node id
  - their ingress TE Link ids (w.r.t. the direction of the path)

- *callEnable()* and *callDisable()*: allow to set the administrative status of the Call to “enabled” and “disabled”, respectively. This is for future use, e.g. to temporarily make a call unavailable for usage, without tearing it down.

- *callDestroy():* allows to remove a newly created Call (it has to be still “Idle”). In that status, no signalling has occurred yet, and the call cannot disappear as a consequence of a teardown. An explicit command is needed.

- *callSetUp()* and *callSetDown()*: the access points for setting up and tearing down the Call, respectively. When *callSetUp()* is invoked, a number of checks will occur on consistency and completeness of the information made available ([GNS] TNAs, ERO, etc.).

- *getCalls()*: allows to retrieve the list of the IDs of the Calls currently present at the NCC or CCC.

- *callGetDetails()*: allows to retrieve part of the details of a specific Call (call parameters, LSP parameters, recovery information, states, IDs of the recovery bundles attached to this call). Further information is retrieved by:

- *callGetTna()*: allows to retrieve the details on the legacy TNA resource at the ingress or egress position.

- *callGetGnsTna()*: allows to retrieve the details on the GNS TNA at the ingress or egress position.

The methods for the *SouthBound* interface are:

- *getNotification*(): allows the Call Controller to receive notifications from the Recovery Controller about its recovery bundles (aka “connections” in G.7713/Y.1704 terminology), attached to a Call. The main events are:

  - a new recovery bundle is ready
  - a new recovery bundle has been torn down
  - the setup of a recovery bundle failed

- o the teardown of a recovery bundle failed

8.1.3 xCC Signalling Interfaces

The *ccsigif* module implements a generic wrapper for all the signalling interfaces that the CCC or NCC have to cross with their transactions. These are:

- *G.I-NNI (CcSigIfInni)*. No signalling protocol is specifically mandated by ASON for the NCC-to-NCC communication across the I-NNI (unless piggybacked on G.RSVP-TE signalling for connection setup). IETF CCAMP introduces the usage of the G.RSVP-TE Notify message for I-NNI call signalling purposes (RFC 4974, see D2.1 and D2.2), but with a number of unclear and incomplete points. Due to these incompleteness and to the needed GNS enhancements, a dedicated and proprietary signalling based on XML has been defined and implemented.

- *G.UNI (CcSigIfUni)*. To be based on OIF UNI 2.0 (see D2.1, D2.2, D2.7 for references)

- *G.E-NNI (CcSigIfEnni)*. To be based on OIF E-NNI 2.0 signalling (see D2.1, D2.2, D2.7 for references)

- *B-NNI (CcSigIfBnni)*. This is the *Border Node-to-Node Interface*, which implements the part of signalling between UNI-N NCCs needed to support the concept of Indirect Call introduced in D2.1. This is based on a proprietary signalling based on XML.

- *Mgmt (CcSigIfMgmt)*. Not a real signalling interface. It is currently a pure stub, and might be used in a future engineering of the stack as the source point for SNMP traps (e.g. to let the NMS know when an SPC Call is ready).

Each of these interfaces is instantiated and attached at the *CallController* level, and provides a gateway to the underlying signalling functions (send and receive), e.g. through G.UNI RSVP or G.ENNI RSVP for G.UNI and G.E-NNI, respectively, or a full implementation of the XML signalling specified.

8.1.3.1 *G.I-NNI and B-NNI XML signalling*

The specified signalling protocol is based on the ASON message (G.7713/Y.1704) types and includes all the relevant information needed to setup the Call.

The supported messages are:

- For Call setup:
    - o SetupRequest

- o SetupIndication
- o SetupConfirm

- For Call teardown:
  - o ReleaseRequest
  - o ReleaseConfirm

The basic message structure is as follows.

```
<!ELEMENT ccsigmsg (header, body)>

      <!ELEMENT header (type, seqnum, sender)>
            <!ELEMENT type (#PCDATA[6])>
            <!ELEMENT seqnum (#PCDATA)>
            <!ELEMENT sender (#PCDATA)>


      <!ELEMENT body (name, client-name?, call-id?, indirect?, rel-ind-call-id?,
      emulated-if?, call-parms?, lsp-parms?, ero?, reason?, errored-seqnum?)>

            <!ELEMENT name (#PCDATA)>
            <!ELEMENT client-name (#PCDATA)>

            <!ELEMENT call-id (type, srcId, localId, segments?)>
               <!ELEMENT type (#PCDATA)>
               <!ELEMENT srcId (#PCDATA)>
               <!ELEMENT localId (#PCDATA)>

            <!ELEMENT indirect (#PCDATA)>
            <!ELEMENT rel-ind-call-id (type, srcId, localId, segments?)>
            <!ELEMENT emulated-if (#PCDATA)>
            <!ELEMENT reason (#CDATA)>
            <!ELEMENT errored-seqnum (#PCDATA)>

            <!ELEMENT ero (eroelem +)>
               <!ELEMENT eroelem (nodeId, teLink, upDataLink, upLabel,
               downDataLink, downLabel, loose)>
                  <!ELEMENT nodeId (#PCDATA)>
                  <!ELEMENT teLink (#PCDATA)>
                  <!ELEMENT upDataLink (#PCDATA)>
                  <!ELEMENT upLabel (#PCDATA)>
                  <!ELEMENT downDataLink (#PCDATA)>
                  <!ELEMENT downLabel (#PCDATA)>
                  <!ELEMENT loose (#PCDATA)>
```

[6] A string indicating one of the message types reported above.

```
<!ELEMENT call-parms (originator, jobProject, jobName, gnstnas,
disjointness, recoveryType, startTime, endTime, tnares)>
   <!ELEMENT originator (#PCDATA)>
   <!ELEMENT jobProject (#CDATA)>
   <!ELEMENT jobName (#CDATA)>
   <!ELEMENT disjointness (#PCDATA)>
   <!ELEMENT startTime (#PCDATA)>
   <!ELEMENT recoveryType (#PCDATA)>
   <!ELEMENT endTime (#PCDATA)>
   <!ELEMENT tnares (ingress, egress)>
     <!ELEMENT ingress (dataLink, label, tna)>
     <!ELEMENT egress (dataLink, label, tna)>
          <!ELEMENT dataLink (#PCDATA)>
          <!ELEMENT label (#PCDATA)>
          <!ELEMENT tna (#PCDATA)>
   <!ELEMENT gnstnas (ANY[7])>


<!ELEMENT lsp-parms (lspRole, lspType, swCap, encType, gpid,
bandwidth, tnResAction, rroMode, setupPrio, holdingPrio, linkProtMask,
includeAll, includeAny, excludeAny, useAcks, rapidRetryLimit,
rapidRetransIntval, incrementValueDelta, refreshInterval,
crankbackScope, maxCbackRetrSrc, maxCbackRetrIntmd)>
   <!ELEMENT lspRole (#PCDATA)>
   <!ELEMENT lspType (#PCDATA)>
   <!ELEMENT swCap (#PCDATA)>
   <!ELEMENT encType (#PCDATA)>
   <!ELEMENT gpid (#PCDATA)>
   <!ELEMENT bandwidth (#PCDATA)>
   <!ELEMENT tnResAction (#PCDATA)>
   <!ELEMENT rroMode (#PCDATA)>
   <!ELEMENT setupPrio (#PCDATA)>
   <!ELEMENT holdingPrio (#PCDATA)>
   <!ELEMENT linkProtMask (#PCDATA)>
   <!ELEMENT includeAll (#PCDATA)>
   <!ELEMENT includeAny (#PCDATA)>
   <!ELEMENT excludeAny (#PCDATA)>
   <!ELEMENT useAcks (#PCDATA)>
   <!ELEMENT rapidRetryLimit (#PCDATA)>
   <!ELEMENT rapidRetransIntval (#PCDATA)>
   <!ELEMENT incrementValueDelta (#PCDATA)>
   <!ELEMENT refreshInterval (#PCDATA)>
   <!ELEMENT crankbackScope (#PCDATA)>
   <!ELEMENT maxCbackRetrSrc (#PCDATA)>
   <!ELEMENT maxCbackRetrIntmd (#PCDATA)>
```

An example of SetupRequest is reported in the following, already parsed:

[7] This is actually a structured element, as well, but its structure it is too complex to be reported here. Basically, its tag names and structure are organized according to the basic GNS IDL types. See Appendix A for further details.

```
header:
  type: 'SetupRequest'
  seqnum: 'i:1'
  sender: '192.168.40.1'
body:
  call-id:
    localId: '0x0000000000000001'
    segments:
    srcId: 'ipv4#192.168.40.1'
    type: 'CALLIDTYPE_OPSPEC'
  name: 'CALLIDTYPE_OPSPEC#(ipv4#192.168.40.1):0x1'
  indirect: 'b:0'
  emulated-if: 'I-NNI'
  call-parms:
    originator: 'ISSUERTYPE_UNI_IF'
    jobProject: 'progetto'
    jobName: 'myjob'
    gnstnas:
    disjointness: 'DISJOINTNESS_NONE'
    startTime: 'i:0'
    recoveryType: 'RECOVERYTYPE_UNPROTECTED'
    endTime: 'i:100'
    tnares:
      ingress:
        dataLink: 'ipv4#0.0.0.0'
        label:
        tna: 'ipv4#10.10.1.101'
      egress:
        dataLink: 'ipv4#0.0.0.0'
        label:
        tna: 'ipv4#10.30.2.120'
  lsp-parms:
    maxCbackRetrIntmd: 'i:0'
    rapidRetransIntval: 'i:0'
    rroMode: 'LSPRROMODE_TEL_DETAIL'
    rapidRetryLimit: 'i:0'
    gpid: 'GPID_LAMBDA'
    incrementValueDelta: 'i:0'
    holdingPrio: 'i:0'
    setupPrio: 'i:0'
    crankbackScope: 'CRANCKBACKSCOPE_E2E'
    linkProtMask: 'PROTTYPE_UNPROTECTED'
    excludeAny: 'i:0'
    useAcks: 'i:0'
    swCap: 'SWITCHINGCAP_LSC'
    lspRole: 'LSPROLE_UNDEFINED'
    includeAny: 'i:0'
    lspType: 'LSPTYPE_SPC'
    bandwidth: 'i:1000000'
    maxCbackRetrSrc: 'i:3'
    refreshInterval: 'i:0'
    encType: 'ENCODINGTYPE_LAMBDA'
    tnResAction: 'LSPRESOURCEACTION_XCONNECT'
    includeAll: 'i:0'
  ero:
    listelem-001:
      elem:
        downDataLink:
```

```
        downLabel:
        loose: 'b:0'
        nodeId:
        teLink: 'ipv4#192.168.2.50'
        upDataLink:
        upLabel:
```

Code 8-2: Example of parsed SetupRequest.

8.2 G².NCC – The Grid-GMPLS Network Call Controller

8.2.1 G².NCC basics

The G².NCC is the core component for the Grid-GMPLS end-to-end Service Plane. It implements the concept of G² Call, which extends that of ASON/GMPLS Call. The [G²] Call[8] is the bridging element between the G²MPLS Network Control Plane and the Service Plane functionalities. As such, it supports two important features:

- It incorporate information about the “service end-points”, be them legacy TNAs or non-network (grid) resources (defined as “GNS TNAs” in software)
- It offers gateway functions to the AuthN/AuthZ Infrastructure (developed in WP4), thus augmenting the G.UNI and G.E-NNI with inter-carrier capabilities

The G².NCC is implemented in Python 2.5 (code in <sw_root>/pyg2mpls/nccd/).

It shares a common shim software with the G².CCC (G² Client Call Controller), located in <sw_root>/pyg2mpls/xcc/. The shared software between G².NCC and G².CCC implements a set of common objects and functions, which are then inherited by the specialized objects and functions in G².NCC and G².CCC.

The description of the shared “xCC” software can be found in section 14.4.

8.2.2 G².NCC software overview

The G².NCC composing files are:

- *config.py*: protocol-specific configuration file

[8] From now on, the G2 Call is simply referred to as “Call”.

- *main.py*: start-up file, for launching the NCC
- *nccdm.py*: the NCC data model, implementing the *NetworkCallController* and *NetworkCall* classes
- *ncall_fsm.py*: the implementation of the transitions of the NCC Call FSM
- *ncall_fsm_desc.py*: the description of the NCC Call FSM, automatically generated from *<sw_root>/tools/FSM/tools/ncc_call.conf*.

The G^2.NCC is implemented as a single process, and a number o threads (Figure 8-2):

- The main G^2.NCC thread (1), which starts up all the protocol components and enters the ominORB run() cycle.
- The FSM engine (1), which (in its configured usage) waits for FSM events to pop up in the FSM events queue, and execute the related transitions
- The Timers manager (1), which waits for the next timer delta to expire in the timers calendar queue, and executes the related callback function
- The UDP socket manager (1), which waits for UDP packets to appear in the UDP socket, receive them and execute the related callback functions at protocol level
- A number of ORB threads (N), for the execution of servant methods and client invocations.

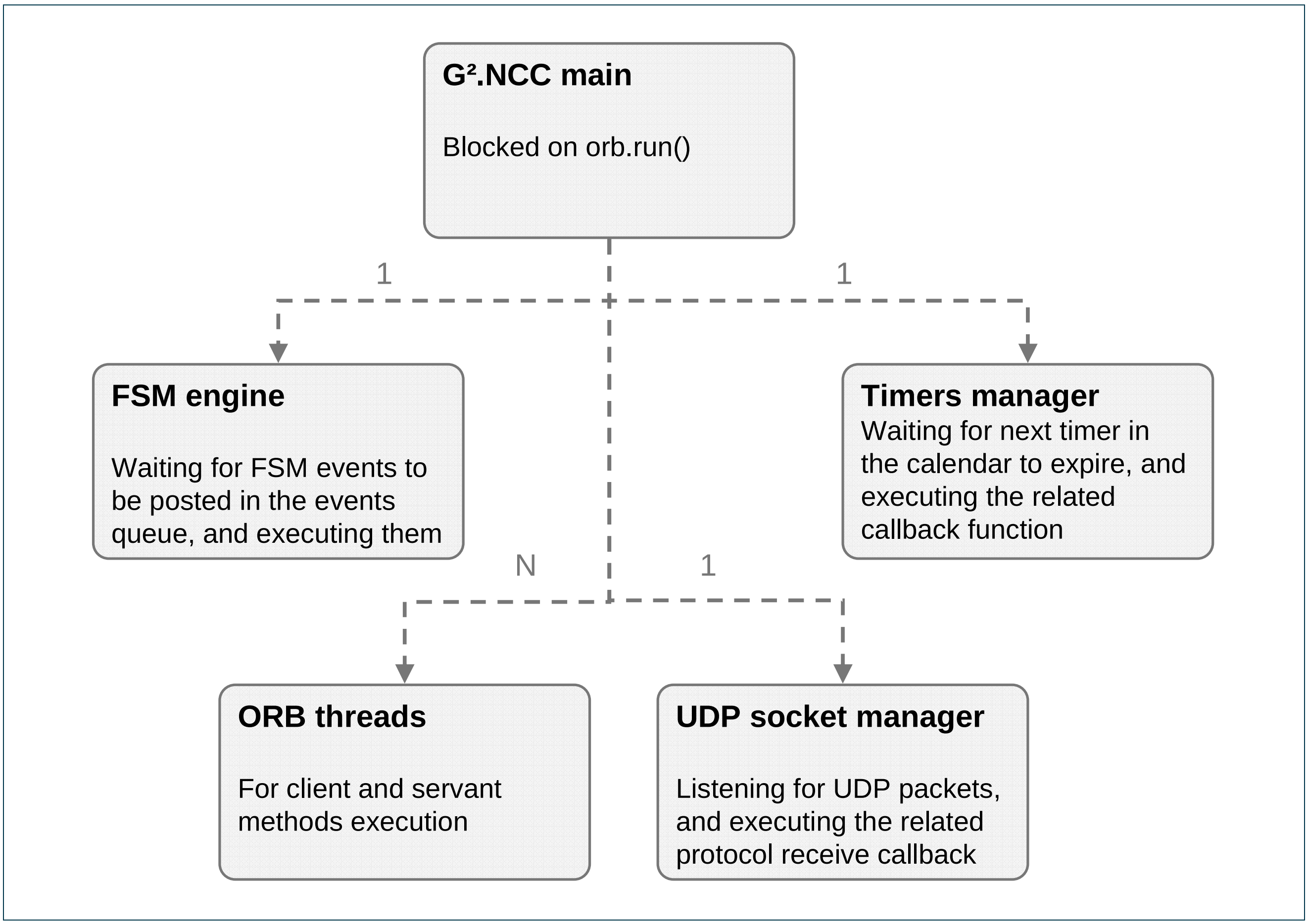

Figure 8-2: G².NCC threads structure

8.2.3 G².NCC data model

Figure 8-3 depicts the NCC Call data model. The main class is the *NetworCallController*, which inherits directly from the *CallController* class in ccdm.py, with its signalling interfaces.

The *NetworkCall* class inherits from the *Call* class in ccdm.py, and, with respect to it, add links to some objects:

- one instance of the *NetworkCallFsm* class, whose methods collect all the in/out transitions of the NCC Call FSM;
- a mirror image of the underlying Recovery Bundle handled by the Recovery Controller (see section 8.3);
- an Call ERO, as a list of *EroItem*(s).

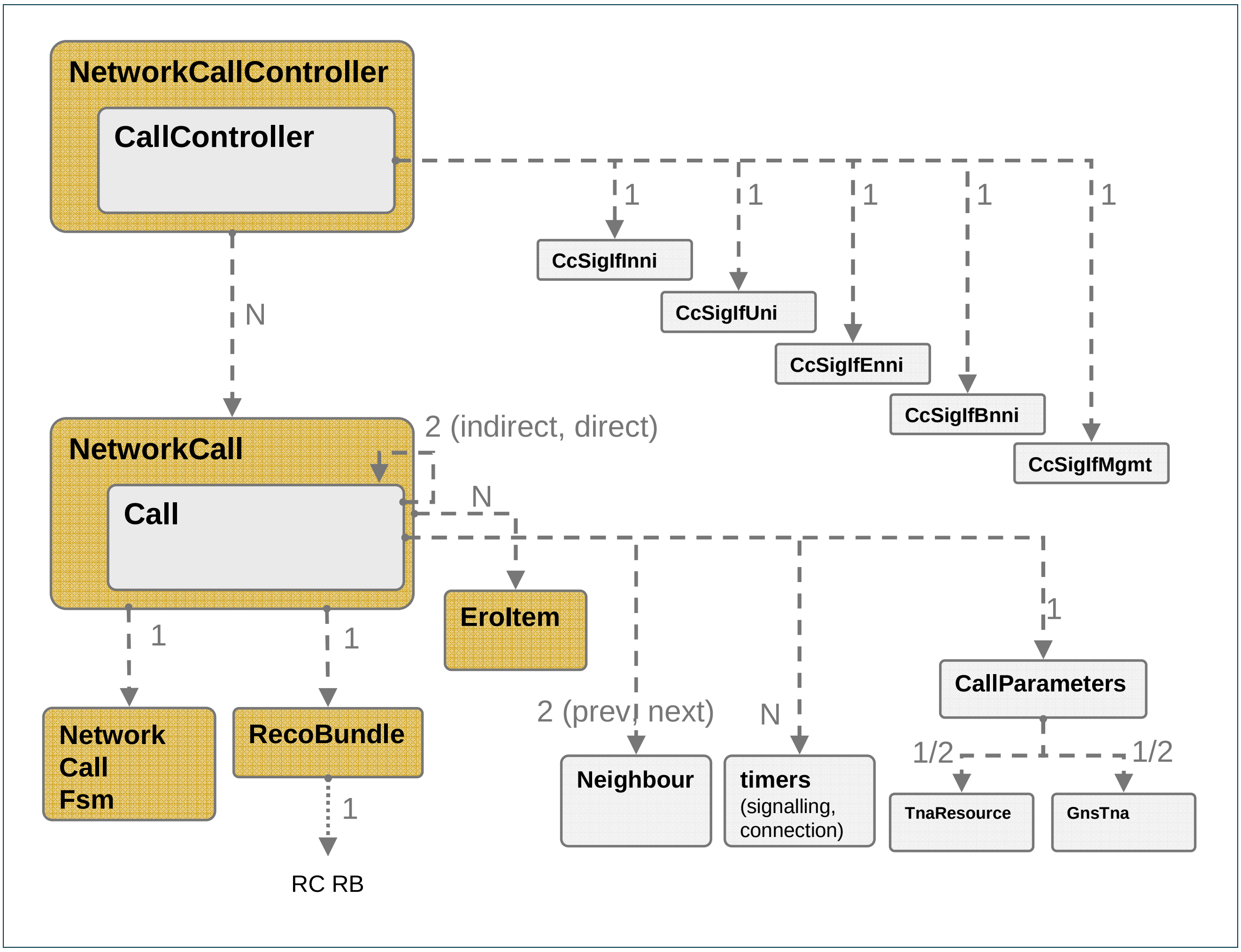

Figure 8-3: G².NCC data model

8.2.3.1 *TNA rules*

When setting up a new call at the NCC-1 via management (*callSetUp*) or when the call is initiated by UNI signalling, a check on the consistency of the provided TNAs (either legacy or GNS) is performed. The legacy TNA is expressed in the form of a *TnaResource,* i.e. a TNA, plus a Data Link, plus a Label. Not all of this info has to be non-null. From now on, the detailed TNA information (aka *TnaResource*) is indifferently referred to as TNA.

The check algorithm is described in the following.

- Both TNAs (ingress and egress) should be present (either in the form of legacy resource or GNS)

- If the ingress TNA is a legacy resource,
  - If the Data Link in the TNA resource is non-null, it should belong to the specified TNA, and both of them should belong to the checking NCC. – If true, the check is over, with a positive result
  - If a null Data Link is present, the check is limited to the TNA: it should belong to the checking NCC. – If true, the check is over, with a positive result
  - If we get here, the TNA does not belong to the checking node $\Rightarrow$ this is an Indirect Call. The checking NCC will ask the PCE which NCC owns the specified TNA, and set that node as the "next" neighbour. – If found, the check is over, with a positive result.
- If the ingress TNA is not a legacy resource, it's for sure a GNS TNA. In this case, the call is always Indirect: Direct Calls always need to specify a network TNA as the ingress point.

8.2.4 G².NCC Call FSM

The FSM of the G².NCC Call is "inspired" by ITU-T Rec. G.7713/Y.1704 (rev. 05/2006) and RFC 4974 (with a 3-tier Call signalling, instead of a simple two-tier); see D2.1 and D2.2 for references. According to the view of the design team of the G².NCC, both recommendations have to be considered as informational suggestions rather than real implementation guidelines. The principle followed is the IETF CCAMP one: the Call has to be completely set up before any network connection (aka LSP) is initiated. Honouring this useful principle forced the adoption of a 3-tier signalling, instead of a simple 2-tier as suggested by RFC 4974 (a minimum of 3-tier is needed when every NCC along the path has to know when the Call is completely ready).

The core skeleton of the FSM is derived from G.7713/Y.1704 (rev. 05/2006), although a number of modifications had to be introduced to make it a usable and working FSM.

The FSM specification is in *<sw_root>/tools/FSM/tools/ncc_call.conf,* and is reported in the following:

```
#
#  NCC CALL FSM definition
#

{ FSM }

name = NCC_CALL_FSM
definition-file = ncc_call.def
# If graphviz-file is defined the graphviz file will be create
graphviz-file = ncc_call.dot
#include-name = ncc_call.h
start-state = Idle #[optional]

#
# Events
```

```
#
#
# rootEvent = derivedEvent1, derivedEvent2, ...
#

{ Events }

inSetupRequest              = inSetupRequestOk, inSetupRequestKo
inSetupIndication           = inSetupIndicationOk, inSetupIndicationKo
inSetupConfirm              = inSetupConfirmOk, inSetupConfirmKo,
inSetupConfirmSkipConn
inReleaseRequest            = inReleaseRequestOk, inReleaseRequestKo
inReleaseIndication         = inReleaseIndicationOk, inReleaseIndicationKo
SetupVerification           = SetupVerificationOk, SetupVerificationKo
ReleaseVerification         = ReleaseVerificationOk, ReleaseVerificationKo,
ReleaseVerificationSkipConn
inCallSigError              = inCallSigError
ConnectionReady             = ConnectionReady
ConnectionFailed            = ConnectionFailed
ConnSetupTimeout            = ConnSetupTimeout
ConnectionVerified          = ConnectionVerifiedOk, ConnectionVerifiedKo
ScnErrorOn                  = ScnErrorOn
ScnErrorOff                 = ScnErrorOff
ConnectionReleased          = ConnectionReleased
ConnRelFailed               = ConnRelFailed
ConnRelTimeout              = ConnRelTimeout



#
# States
#
# state = state1   [The first state is the start one if start-state is not set]
#       eventX -> dstState
#
# state = state2
#       eventY -> dstState
#

{ States }
# see ITU-T Rec. G.7713/Y.1704 (05/2006) and RFC 4974 (with a 3-tier Call signalling)

#
state = Idle                                                # stable
       inSetupRequestOk              -> VerifyCallSetupRequest # aka 'SetReq';
either from mgmt (e.g. setupCall), I-NNI (i.e. Notify msg), UNI, E-NNI (Path)
       inSetupRequestKo              -> .                      #

#
state = VerifyCallSetupRequest
       SetupVerificationOk           -> CallSetupRequestInitiated     # aka
'SetVer'; verify ok should be automatic on downstream NCC
       SetupVerificationKo           -> Idle                   # aka 'SetNVer'
       inReleaseRequestOk            -> Idle                   # aka 'RelReq';
either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown,
ResvDown, PathErr)
       inReleaseRequestKo            -> .                      #
```

```
#
state = CallSetupRequestInitiated                                # setup
       inSetupIndicationOk            -> CallSetupResponded            # either from
I-NNI (i.e. Notify msg), UNI, E-NNI (Resv)
       inSetupIndicationKo            -> Idle                   #
       inCallSigError                 -> Idle                   # either from I-NNI
(i.e. Notify msg), UNI, E-NNI (PathErr)
       inReleaseRequestOk             -> Idle                   # aka 'RelReq';
either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown,
ResvDown, PathErr)
       inReleaseRequestKo             -> .                      # aka 'RelReq';
either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown,
ResvDown, PathErr)

#
state = CallSetupResponded       # setup
       inSetupConfirmOk               -> SetupConnection        # either from I-NNI
(i.e. Notify msg), UNI, E-NNI (ResvConf)
       inSetupConfirmSkipConn         -> Active                 # either from I-NNI
(i.e. Notify msg), UNI, E-NNI (ResvConf)
       inSetupConfirmKo               -> Idle                   #
       inCallSigError                 -> Idle                   # either from I-NNI
(i.e. lack of ack to Notify), UNI, E-NNI (ResvErr)
       inReleaseRequestOk       -> Idle                  # aka 'RelReq'; either from
mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown, ResvDown,
PathErr)
       inReleaseRequestOk             -> Idle                   #

#
state = SetupConnection                # setup (connections are being set up)
       ConnectionReady                -> VerifyCall             # aka 'SetCon'; the
Recovery Bundle is up (Resv/ResvConf on last LSP in the RC for upstream/downstream
NCC)
       ConnectionFailed               -> ReleaseConnection            # aka
'SetNCon'; the Recovery Bundle failed (ResvErr/PathErr on last LSP in the RC for
upstream/downstream NCC)
       ConnSetupTimeout               -> ReleaseConnection            # aka
'SetExp'; the Recovery Bundle setup timed out
       inReleaseRequestOk             -> VerifyCallReleaseRequest     # aka
'RelReq'; either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI
(PathDown, ResvDown, PathErr)
       inReleaseRequestKo             -> .                      # aka 'RelReq';
either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown,
ResvDown, PathErr)

#
state = VerifyCall                     # setup
       ConnectionVerifiedOk           -> Active                 # aka 'SetCallVer'
       # nop, so far
       ConnectionVerifiedKo           -> ReleaseConnection            # aka
'SetCallNVer' # nop, so far
       inReleaseRequestOk             -> VerifyCallReleaseRequest     # aka
'RelReq'; either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI
(PathDown, ResvDown, PathErr)
       inReleaseRequestKo             -> .                      #

#
state = Active                         # stable
```

```
       ScnErrorOn                                    -> SigError              # aka
'SigErr'
       inReleaseRequestOk             -> VerifyCallReleaseRequest      # aka
'RelReq'; either from mgmt (e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI
(PathDown, ResvDown, PathErr)
       inReleaseRequestKo             -> .                          #

#
state = SigError                  # stable; not used, so far
       ScnErrorOff                    -> Active                     # aka 'SigNErr'

#
state = VerifyCallReleaseRequest
       ReleaseVerificationOk          -> ReleaseConnection                # aka
'RelVer'; verify ok should be automatic on downstream NCC
       ReleaseVerificationSkipConn    -> CallReleaseRequestInitiated   # aka
'RelVer'; verify ok should be automatic on downstream NCC
       ReleaseVerificationKo          -> Idle                       # aka 'RelNVer'

#
state = ReleaseConnection         # release (connections are being released)
       ConnectionReleased             -> CallReleaseRequestInitiated   # aka
'RelCon'
       ConnRelFailed                  -> CallReleaseRequestInitiated   # aka
'RelNCon'
       ConnRelTimeout                 -> CallReleaseRequestInitiated   # aka
'RelExp'

#
state = CallReleaseRequestInitiated                         # release
       inReleaseIndicationOk          -> Idle                       #
       inReleaseIndicationKo          -> Idle                       #
       inCallSigError                 -> Idle                       #
```

Code 8-3: G².NCC Call FSM.

The G².NCC Call states are reported in the following table. The steady ones have their names in italic.

State	short description
Idle	The Call has been created, but no signalling has occurred on it yet.
VerifyCallSetupRequest	The call setup signalling has been initiated (either a *SetupRequest* was received from the network, or a management command has been issued), and policy verification has started (i.e. an AuthZ request has been sent to the AAI). Waiting for a reply to the policy verification. Depending on the policy configuration, this state can be skipped at some NCCs (e.g. it can be valid only for the ingress ones, downstream of UNI or E-NNIs).
CallSetupRequestInitiated	The policy verification concluded successfully (or it was simply skipped), and the *SetupRequest* message has been propagated downstream. Waiting for an answer to it (*SetupIndication*).
CallSetupResponded	A *SetupIndication* has been received from the downstream NCC (or CCC if the downstream NI is a UNI). Waiting for the Call to be fully completed (i.e. the NCC has to see a *SetupConfirm* concerning this Call).

SetupConnection	The *SetupConfirm* has been received (or sent, if the Call FSM is at NCC-1), and the Call setup signalling has successfully completed. The setup of the network connections has started (i.e. the creation and setup of Recovery Bundles at the Recovery Controller have been commanded). Waiting for this process to successfully complete.
VerifyCall	The Call is now equipped with network connections (i.e. Recovery Bundles and LSPs). This state can be optionally used at some NCCs (e.g. upstream ones) to verify the Call connectivity across the domain. If this is not foreseen, the Call jumps to the *Active* state.
Active	The Call has now reached is up steady state: it has been authorized, signalled, equipped with network connections and (optionally) verified at Data Plane level.
SigError	An alternate steady state w.r.t. the *Active* one: some signalling error has occurred on the Call after its setup.
VerifyCallReleaseRequest	The call teardown signalling has been initiated (either a *ReleaseRequest* was received from the network, or a management command has been issued), and policy verification has started (i.e. an AuthZ request has been sent to the AAI). Waiting for a reply to the policy verification. Depending on the policy configuration, this state can be skipped at some NCCs (e.g. it can be valid only for the ingress ones, downstream of UNI or E-NNIs).
ReleaseConnection	The policy verification concluded successfully (or it was simply skipped); now the teardown has been authorized. The teardown of network connections has started (i.e. proper teardown commands have been issued to the Recovery Controller concerning the Recovery Bundle associated to this Call). Waiting for the network connections to be torn down.
CallReleaseRequestInitiated	All the network connections associated to this Call have been torn down (i.e. no more RBs at RC, and LSPs at G².RSVP-TE), and the *ReleaseRequest* message has been propagated upstream or downstream. Waiting for an answer to it (*ReleaseIndication*); when it will come, the Call will jump back to its *Idle* state and be deleted.

Table 8-1: G².NCC Call FSM: states

The following table reports the root events that feed the FSM. When a root event might result in different detailed events, this is discussed case by case.

Root event	short description
inSetupRequest	A *SetupRequest* has been received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI or Mgmt. In the latter case, actually it is a command from the management (i.e. via CORBA) which reached the NCC Call.
inSetupIndication	A *SetupIndication* has been received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI.
inSetupConfirm	A *SetupConfirm* has been received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI.

inReleaseRequest	A *ReleaseRequest* has been received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI or Mgmt. In the latter case, actually it is a command from the management (i.e. via CORBA) which reached the NCC Call.
inReleaseIndication	A *ReleaseIndication* has been received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI.
SetupVerification	The Call setup policy verification concluded, either positively or negatively (different derived events).
ReleaseVerification	The Call teardown policy verification concluded, either positively or negatively (different derived events).
inCallSigError	Some call signalling error was received through one of the NCC signalling interfaces: G.I-NNI, G.UNI, G.E-NNI, B-NNI.
ConnectionReady	The setup of the network connections (aka RB at the RC) concluded successfully.
ConnectionFailed	The setup of the network connections (aka RB at the RC) failed.
ConnSetupTimeout	The setup of the network connections (aka RB at the RC) did not conclude within the configured timeframe.
ConnectionVerified	The Data Plane verification of the network connections (aka RB at the RC) has been carried out successfully.
ScnErrorOn	Some error in the SCN occurred.
ScnErrorOff	The pending errors in the SCN have been cleared.
ConnectionReleased	The teardown of the network connections (aka RB at the RC) concluded successfully.
ConnRelFailed	The teardown of the network connections (aka RB at the RC) failed.
ConnRelTimeout	The teardown of the network connections (aka RB at the RC) did not conclude within the configured timeframe.

Table 8-2: G^2.NCC Call FSM: root events

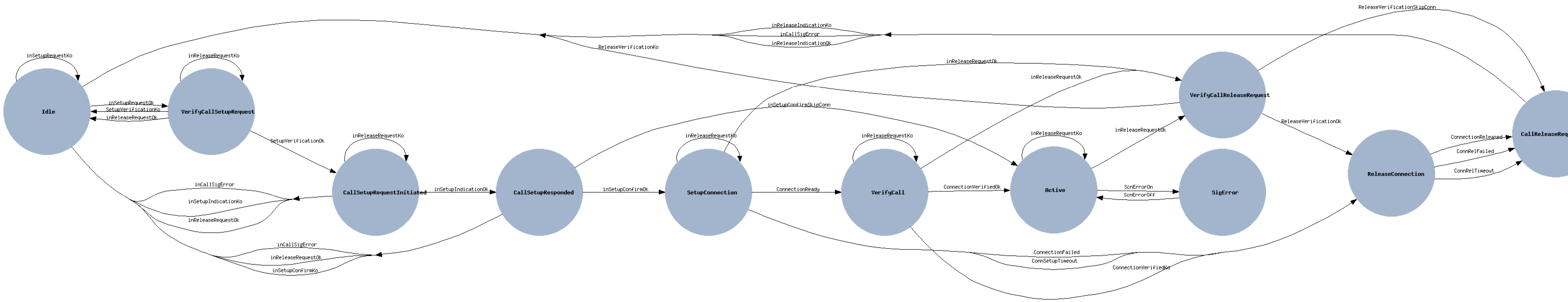

Figure 8-4: G².NCC Call FSM.

8.3 G².CCC – The Grid-GMPLS Client Call Controller

8.3.1 G².CCC basics

The G².CCC provides a prototypal implementation of the client-end of the G² Call. The client-side of the Call is the access point for the creation of G² services, and their request as GNS through the G.UNI. The G² Call at the CCC can be controlled in two ways:

- Automatically from job requests coming from the grid middleware, translated into Calls by the G.UNI Gateway (see section 11).
- Via management, using the CORBA interface to the CCC.

The G².CCC is implemented in Python 2.5 (code in <sw_root>/pyg2mpls/cccd/). It shares a common shim software with the G².NCC (G² Network Call Controller), located in <sw_root>/pyg2mpls/xcc/, as discussed before for the NCC, and detailed in section 14.4.

8.3.2 G².CCC software overview

The G².CCC composing files are:

- *config.py*: protocol-specific configuration file
- *main.py*: start-up file, for launching the CCC
- *cccdm.py*: the CCC data model, implementing the *ClientCallController* and *ClientCall* classes
- *ccall_fsm.py*: the implementation of the transitions of the CCC Call FSM
- *ccall_fsm_desc.py*: the description of the CCC Call FSM, automatically generated from *<sw_root>/tools/FSM/tools/ccc _call.conf*.

The G^2.CCC is implemented as a single process, and a number o threads (Figure 8-5):

- The main G^2.CCC thread (1), which starts up all the protocol components and enters the ominORB run() cycle.

- The FSM engine (1), which (in its configured usage) waits for FSM events to pop up in the FSM events queue, and execute the related transitions
- The Timers manager (1), which waits for the next timer delta to expire in the timers calendar queue, and executes the related callback function
- The UDP socket manager (1), which waits for UDP packets to appear in the UDP socket, receive them and execute the related callback functions at protocol level
- A number of ORB threads (N), for the execution of servant methods and client invocations.

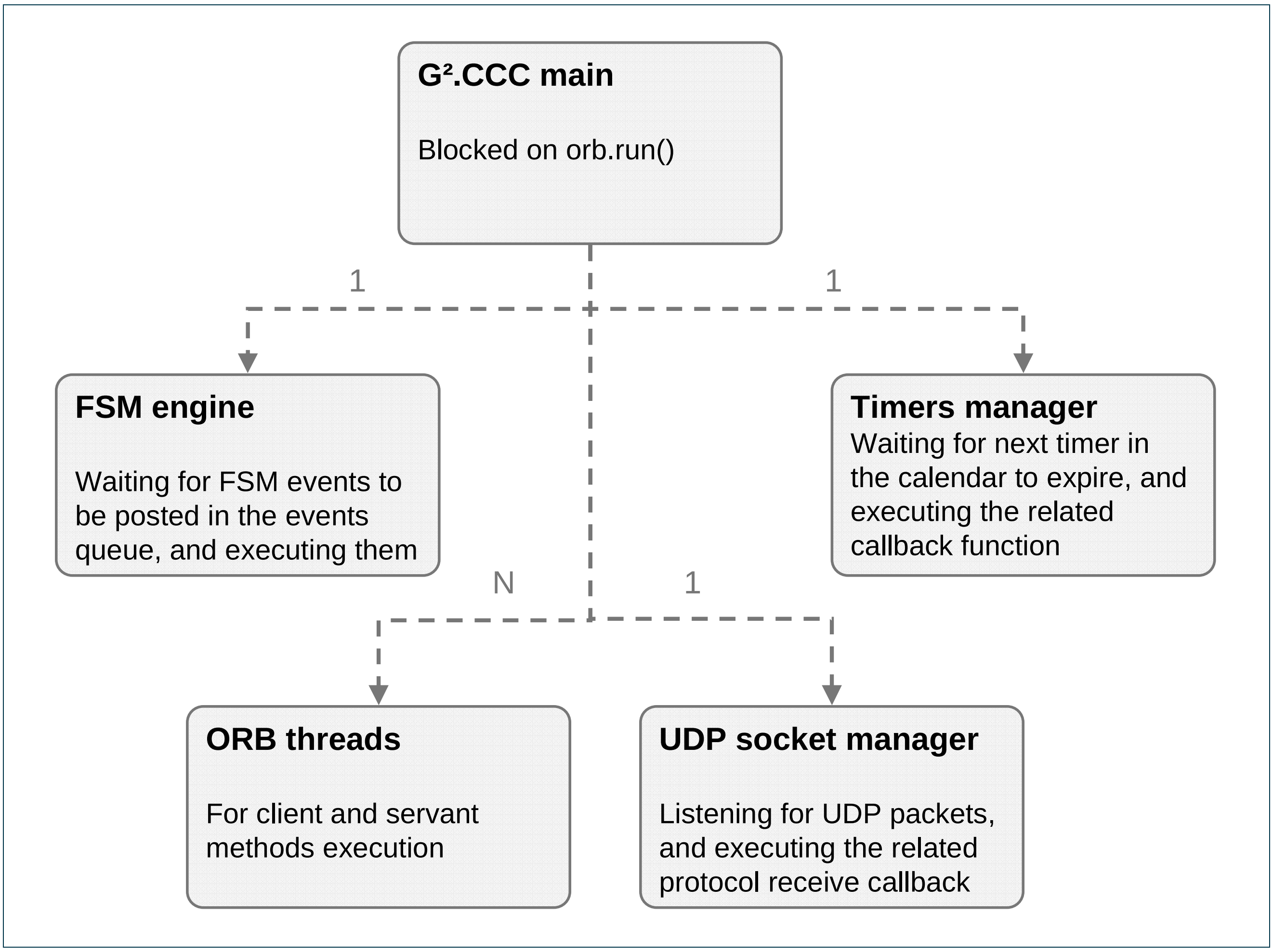

Figure 8-5: G².CCC threads structure

8.3.3 G².CCC data model

Figure 8-6 depicts the CCC Call data model. The main class is the *ClientCallController*, which inherits directly from the *CallController* class in ccdm.py, with its signalling interfaces.

The *ClientCall* class inherits from the *Call* class in ccdm.py, and it is a simplified version of the NCC Call. It points to one instance of the *ClientCallFsm* class, whose methods collect all the in/out transitions of the CCC Call FSM.

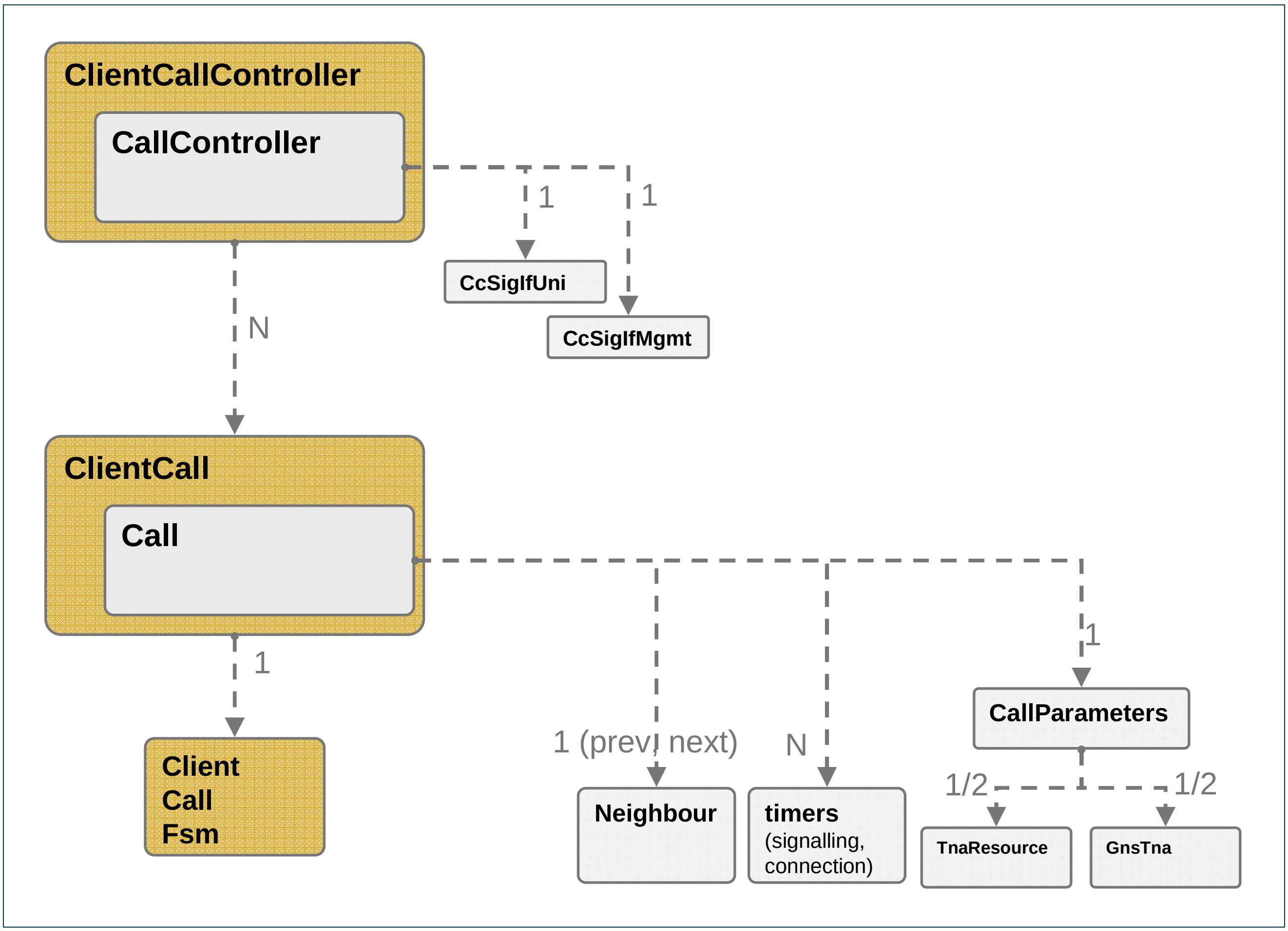

Figure 8-6: G².CCC data model

8.3.4 G².CCC Call FSM

As in the case of the the NCC Call FSM, the FSM of the G².CCC Call is “inspired” by ITU-T Rec. G.7713/Y.1704 (rev. 05/2006) and RFC 4974 (with a 3-tier Call signalling, instead of a simple two-tier); see D2.1 and D2.2 for references.

With respect to the NCC Call FSM, the CCC Call FSM is simpler (less states and less events), mostly due to the fact that the CCC has not to deal with network connections; i.e. it implements just the Service Plane part of the Call.

The FSM specification is in *<sw_root>/tools/FSM/tools/ccc _call.conf,* and is reported in the following:

```
#
#  CCC CALL FSM definition
#

{ FSM }

name = CCC_CALL_FSM
definition-file = ccc_call.def
# If graphviz-file is defined the graphviz file will be create
graphviz-file = ccc_call.dot
#include-name = ccc_call.h
start-state = Idle #[optional]

#
# Events
#
#
# rootEvent = derivedEvent1, derivedEvent2, ...
#

{ Events }

inSetupRequest             = inSetupRequestOk, inSetupRequestKo
inSetupIndication          = inSetupIndicationOk, inSetupIndicationKo
inSetupConfirm             = inSetupConfirmOk, inSetupConfirmKo
inReleaseRequest           = inReleaseRequestOk, inReleaseRequestKo
inReleaseIndication        = inReleaseIndicationOk, inReleaseIndicationKo
SetupVerification          = SetupVerificationOk, SetupVerificationKo
ReleaseVerification        = ReleaseVerificationOk, ReleaseVerificationKo
inCallSigError             = inCallSigError
ScnErrorOn                 = ScnErrorOn
ScnErrorOff                = ScnErrorOff



#
# States
#
# state = state1   [The first state is the start one if start-state is not set]
#       eventX -> dstState
#
```

```
# state = state2
#       eventY -> dstState
#

{ States }
# see ITU-T Rec. G.7713/Y.1704 (05/2006) and RFC 4974 (with a 3-tier Call signalling)

#
state = Idle
       # stable
       inSetupRequestOk          -> VerifyCallSetupRequest       # aka 'SetReq';
either from mgmt (e.g. setupCall), I-NNI (i.e. Notify msg), UNI, E-NNI (Path)
       inSetupRequestKo          -> .                                         #

#
state = VerifyCallSetupRequest
       SetupVerificationOk       -> CallSetupRequestInitiated    # aka 'SetVer';
verify ok should be automatic on downstream NCC
       SetupVerificationKo       -> Idle                         # aka 'SetNVer'
       inReleaseRequestOk        -> Idle                         # aka 'RelReq';
either from mgmt (e.g. teardownCall) or UNI
       inReleaseRequestKo        -> .                            #

#
state = CallSetupRequestInitiated                                # setup
       inSetupIndicationOk       -> CallSetupResponded           #
       inSetupIndicationKo       -> Idle                         #
       inCallSigError                  -> Idle
       # from UNI
       inReleaseRequestOk        -> Idle                         # aka 'RelReq';
either from mgmt (e.g. teardownCall) or UNI
       inReleaseRequestKo        -> .                            # aka 'RelReq';
either from mgmt (e.g. teardownCall) or UNI

#
state = CallSetupResponded                   # setup; aka "Call Setup Accepted"
       inSetupConfirmOk          -> Active     # from UNI
       inSetupConfirmKo          -> Idle                   #
       inCallSigError                  -> Idle             # from UNI
       inReleaseRequestOk        -> Idle             # aka 'RelReq'; either from mgmt
(e.g. teardownCall), I-NNI (i.e. Notify msg), UNI, E-NNI (PathDown, ResvDown, PathErr)
       inReleaseRequestOk        -> Idle             #

#
state = Active
       # stable
       ScnErrorOn                -> SigError               # aka 'SigErr'
       inReleaseRequestOk        -> VerifyCallReleaseRequest     # aka 'RelReq';
either from mgmt (e.g. teardownCall) or UNI
       inReleaseRequestKo        -> .                            #

#
state = SigError
       # stable; not used, so far
       ScnErrorOff               -> Active                       # aka 'SigNErr'

#
state = VerifyCallReleaseRequest
```

```
       ReleaseVerificationOk     -> CallReleaseRequestInitiated   # aka 'RelVer';
verify ok should be automatic on downstream CCC
       ReleaseVerificationKo     -> Idle                        # aka 'RelNVer'

#
state = CallReleaseRequestInitiated                             # release
       inReleaseIndicationOk     -> Idle                                            #
       inReleaseIndicationKo     -> Idle                                            #
       inCallSigError            -> Idle                                            #
```

Code 8-4: G².CCC Call FSM.

The G².CCC Call states are reported in the following table. The steady ones have their names in italic.

State	short description
Idle	The Call has been created, but no signalling has occurred on it yet.
VerifyCallSetupRequest	The call setup signalling has been initiated (either a *SetupRequest* was received from the network, or a management/G.UNI GW command has been issued), and policy verification has started (i.e. an AuthZ request has been sent to the AAI). Waiting for a reply to the policy verification. Depending on the policy configuration, this state can be skipped at some CCCs, e.g. it can be valid only for the CCC-z, in order to allow or disallow access to grid resources to the caller.
CallSetupRequestInitiated	The policy verification concluded successfully (or it was simply skipped), and the *SetupRequest* message has been propagated downstream. Waiting for an answer to it (*SetupIndication*).
CallSetupResponded	A *SetupIndication* has been received from the downstream CCC (or CCC if the downstream NI is a UNI). Waiting for the Call to be fully completed (i.e. the CCC has to see a *SetupConfirm* concerning this Call).
Active	The *SetupConfirm* has been received (CCC-z) or sent (CCC-a). The Call has now reached is up steady state: it has been authorized and signalled.
SigError	An alternate steady state w.r.t. the *Active* one: some signalling error has occurred on the Call after its setup.
VerifyCallReleaseRequest	The call teardown signalling has been initiated (either a *ReleaseRequest* was received from the network, or a management command has been issued), and policy verification has started (i.e. an AuthZ request has been sent to the AAI). Waiting for a reply to the policy verification. Depending on the policy configuration, this state can be skipped at some or all CCCs.
CallReleaseRequestInitiated	The release request has been authorized (or just skipped), and the *ReleaseRequest* message has been propagated upstream or downstream. Waiting for an answer to it (*ReleaseIndication*); when it will come, the Call will jump back to its *Idle* state and be deleted.

Table 8-3: G².CCC Call FSM: states

The following table reports the root events that feed the FSM. When a root event might result in different detailed events, this is discussed case by case.

Root event	short description
inSetupRequest	A *SetupRequest* has been received through one of the CCC signalling interfaces: G.UNI or Mgmt. In the latter case, actually it is a command from the management or middleware via the G.UNI GW (i.e. via CORBA) which reached the CCC Call.
inSetupIndication	A *SetupIndication* has been received through the CCC G.UNI signalling interface.
inSetupConfirm	A *SetupConfirm* has been received through the CCC G.UNI signalling interface.
inReleaseRequest	A *ReleaseRequest* has been received through one of the CCC signalling interfaces: G.UNI or Mgmt. In the latter case, actually it is a command from the management or middleware via the G.UNI GW (i.e. via CORBA) which reached the CCC Call.
inReleaseIndication	A *ReleaseIndication* has been received through the CCC G.UNI signalling interface.
SetupVerification	The Call setup policy verification concluded, either positively or negatively (different derived events).
ReleaseVerification	The Call teardown policy verification concluded, either positively or negatively (different derived events).
inCallSigError	Some call signalling error was received through the CCC G.UNI signalling interface.
ScnErrorOn	Some error in the SCN occurred.
ScnErrorOff	The pending errors in the SCN have been cleared.

Table 8-4: G².CCC Call FSM: root events

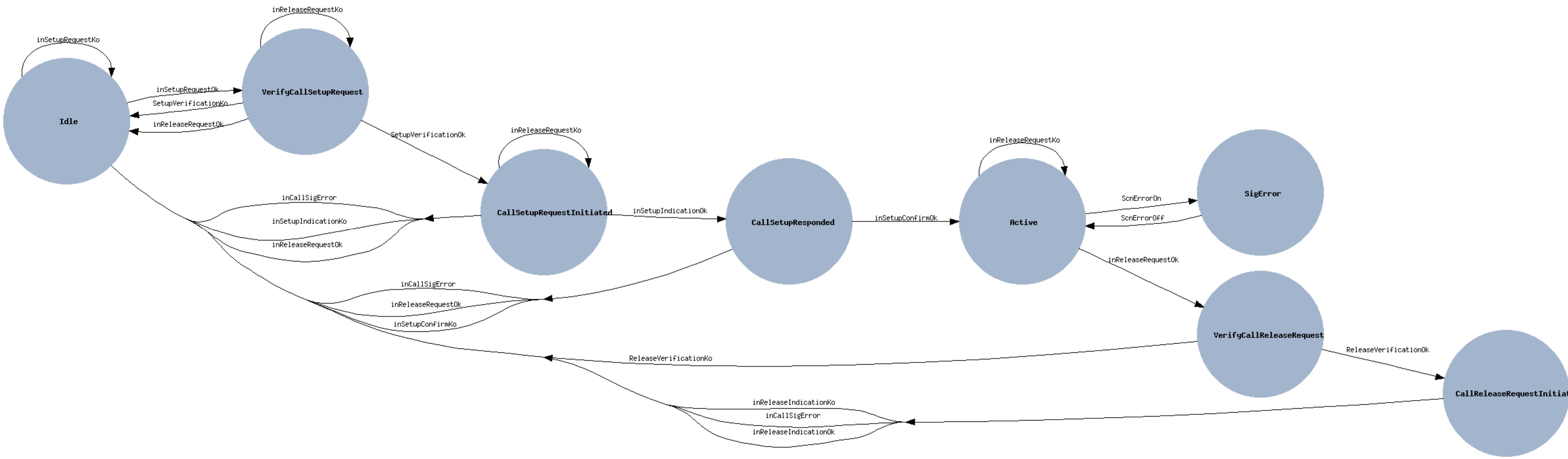

Figure 8-7: G².CCC Call FSM.

9 Recovery Controller (RC)

9.1 Recovery Controller basics

The Recovery Controller (RC) is the key module for creating and handling the recovery (i.e. both protection and restoration) functionalities. The Recovery Controller is not actually mandated in any of the ASON functional modules or PCs, but it is implied by the concept of a Call Segment transport with resiliency properties. The Recovery Controller interfaces to the G².RSVP-TE directly, and commands the creation, setup, teardown and deletion of G².RSVP-TE LSPs. The G².RSVP-TE, in its turn, keeps the RC informed about the status of the requested LSPs, via a set of notifications (see section 9.5).

The RC implements the recovery of LSP introducing the concept of "**Recovery Bundle**" (RB, or RecoBundle). A Recovery Bundle introduces a new functional layer between two ASON objects: the Call and the Connection. In practical terms, the Call Controller responsible for setting up the transport network resources across the administrative domain (i.e. the upstream NCC) will not create the LSPs directly, but will ask the underlying RC to create a Recovery Bundle, with specific recovery features. The RC, in its turn, will equip the Recovery Bundle with as many LSPs as needed by the specified recovery level. This might mean 1 (e.g. for unprotected, or reoruting aka "on-the-fly" restoration) or 2 LSPs (e.g. for a 1+1 protection). Also, the RB will be set with a specific behaviour, depending on the selected recovery (e.g. an RB with just 1 LSP in it will behave differently on failures, depending if the selected behaviour is "unprotected" or "rerouting").

The current implementation of the RC deals with intra-domain recovery only. Inter-domain recovery is affected by pending architectural and protocol-specific issues (e.g. availability of inter-domain OAM) that go beyond the scope of WP2 in Phosphorus.

The specified recovery types for G²MPLS are defined in <sw_root>/idl/g2mplsTypes.idl (a more detailed discussion can be found in D2.1):

- Unprotected (*RECOVERYTYPE_UNPROTECTED*): no protection for this RB; just like having an LSP directly attached to the overlay Call.

- 1+1 Protection (*RECOVERYTYPE_PROTECTION*): a typical 1+1 protection, which is a native feature in SONET/SDH transport networks (SNCP), but a challenge for WSONs (LSC switching capability) or Transport Ethernet networks.

- Pre-planned Protection (*RECOVERYTYPE_PREPLANNED*): protection path calculated before any failure occurred, and "activated" when the failure occurs on the worker LSP.

- Rerouting restoration, aka On-the-fly (*RECOVERYTYPE_OTF*): no path are pre-calculated; everything is performed (rerouting and signalling) when the worker failure occurs. Future releases will allow to differentiate between "soft" (i.e. make-before-break) or "hard" (i.e. break-before-make) rerouting (according to the IETF terminology, *not* the G.7713 one here). The RB FSM already support these two different styles.

- Revertive rerouting (*RECOVERYTYPE_OTF_REVERTIVE*): same as the classic rerouting, but the ability to revert back to the original worker LSP, if its failure heals.

Currently, for fast prototyping reasons, the implemented recovery types are unprotected and hard rerouting. More will be added in the future, according to the actual needs of the NRENs experimenting or deploying the G²MPLS Control Plane.

9.2 Recovery Controller software overview

The RC is implemented in Python 2.5 (code in <sw_root>/pyg2mpls/rcd/). The composing files are:

- *config.py*: protocol-specific configuration file
- *main.py*: start-up file, for launching the RC
- *rcdm.py*: the RC data model, implementing the RecoveryController and RecoveryBundle classes
- *rcsrv.py*: the Recovery Controller CORBA servants
- *recobundle_fsm.py*: the implementation of the transitions of the Recovery Bundle FSM
- *recobundle_fsm_desc.py*: the description of the Recovery Bundle FSM, automatically generated from *<sw_root>/tools/FSM/tools/rc_recobundle.conf*.

The RC is implemented as a single process, and a number o threads (Figure 9-1):

- The main RC thread (1), which starts up all the protocol components and enters the ominORB run() cycle.

- The FSM engine (1), which (in its configured usage) waits for FSM events to pop up in the FSM events queue, and execute the related transitions
- The Timers manager (1), which waits for the next timer delta to expire in the timers calendar queue, and executes the related callback function
- A number of ORB threads (N), for the execution of servant methods and client invocations.

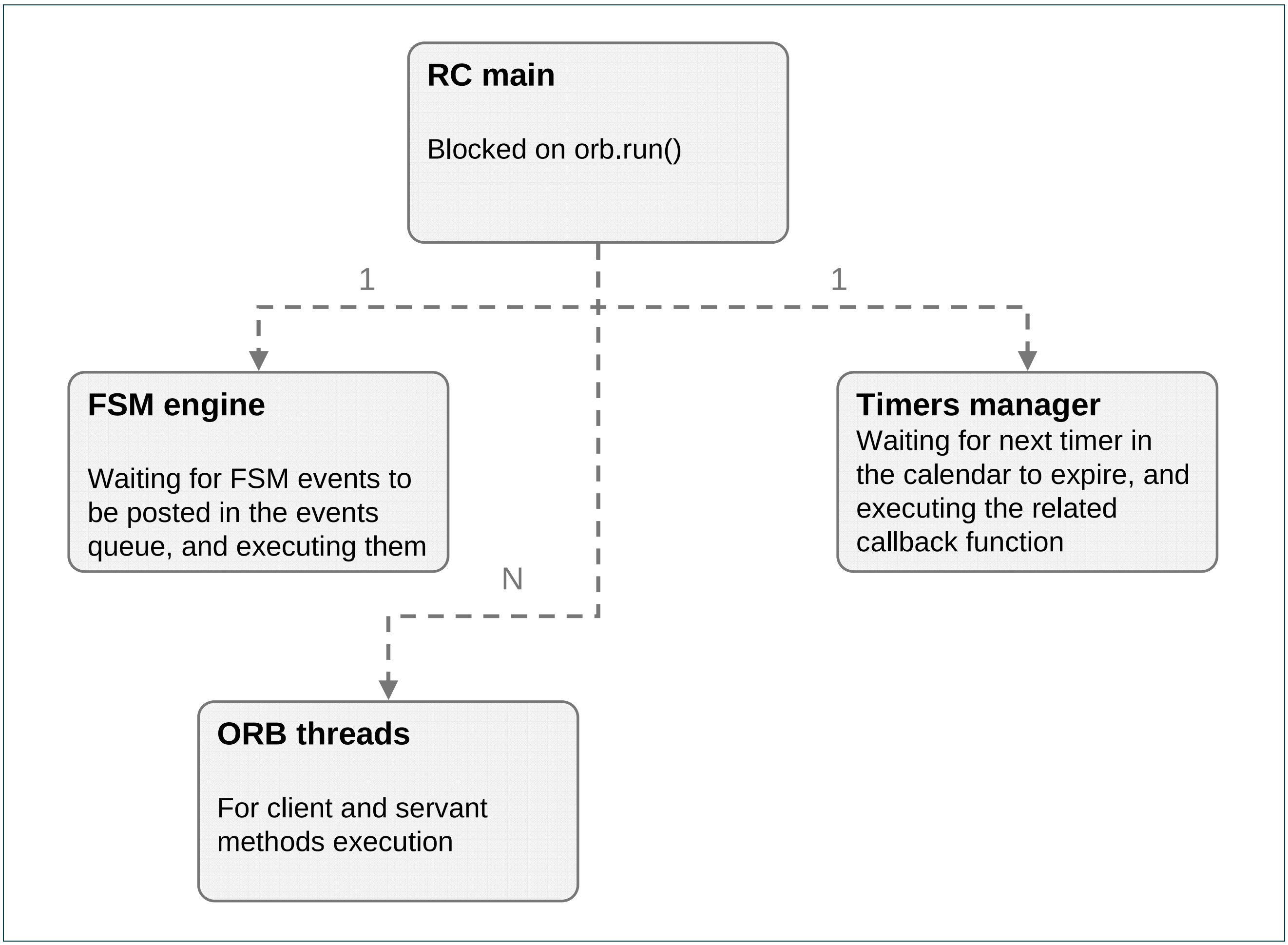

Figure 9-1: RC threads structure

9.3 Recovery Controller data model

Figure 9-2 depicts the RC data model. The main class is the *RecoveryController*, which inherits directly from the *Protocol* class in the module protocol. This class has a number of indirect descendants (inherited from *Protocol*): the *TimersCalendar* and the *CorbaRoot* (with CORBA client and servants under it).

The *RecoveryBundle* class is the core item for implementing the recovery behaviour, and links to:

- One instance of the *RecoveryBundleFsm* class, whose methods collect all the in/out transitions of the RB FSM.
- The *Lsp* class, a mirror image of the corresponding LSP at the G^2.RSVP-TE level: it is needed to store some basic data about the LSP; e.g. whether it exists or not, whether is up or not, some of its parameters, etc.
- A number of timers for managing timeouts during the recovery procedures.
- The pointer (*CallId*) to the owning Call at the NCC level, plus a copy of its parameters (*CallParameters,* mainly for the parameters related to the recovery properties of the Call).

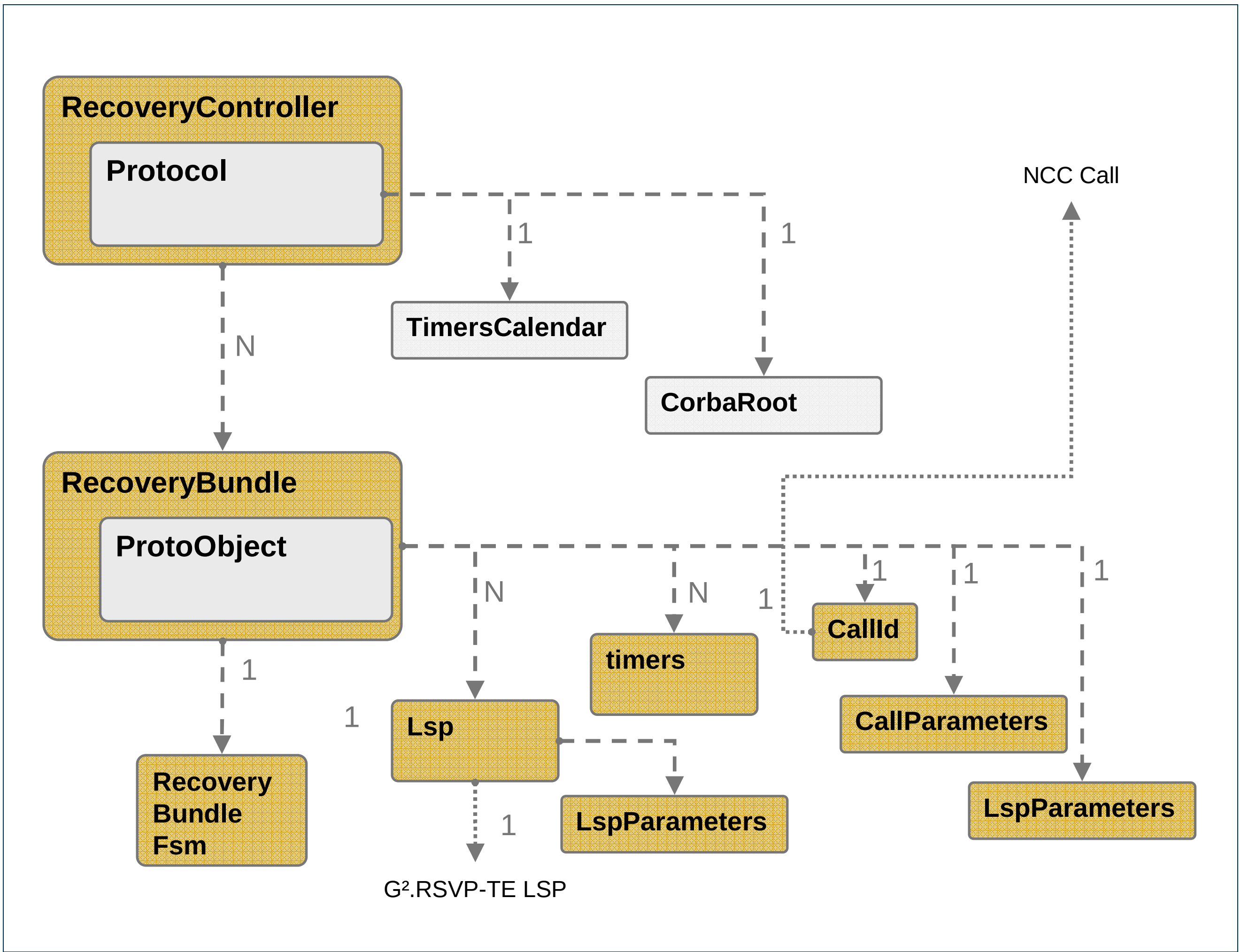

Figure 9-2: RC data model

9.4 RC Recovery Bundle FSM

The FSM of the Recovery Bundle is designed in such a way to allow for many possible recovery procedures. This choice makes the FSM intrinsically more complex than a set of separate FSMs, one for each kind of RB (according to the recovery type), but achieve a higher flexibility when it comes to dynamically change the recovery type of an RB, with no service disruption.

The FSM specification is in *<sw_root>/tools/FSM/tools/rc_recobundle.conf,* and is reported in the following:

```
#
#  Recovery Controller (RC) - Recovery Bundle FSM definition
#

{ FSM }

name = RC_RECOBUNDLE_FSM
definition-file = rc_recobundle.def
# If graphviz-file is defined the graphviz file will be create
graphviz-file = rc_recobundle.dot
#include-name = rc_recobundle.h
start-state = Down #[optional]

#
# Events
#
#
# rootEvent = derivedEvent1, derivedEvent2, ...
#

{ Events }
             WorkerInstalled           = evWorkerInstalled
             ProtectionInstalled       = evProtectionInstalled
             WorkerSigErr              = evWorkerSigErr
             ProtectionSigErr          = evProtectionSigErr
             WorkerDeleted             = evWorkerDeleted
             ProtectionDeleted         = evProtectionDeleted
             WorkerFailed              = evWorkerFailedUseSR, evWorkerFailedUseHR,
evWorkerFailedMngErr, evWorkerFailedNoAction
             ProtectionFailed          = evProtectionFailedNoAction#,
evProtectionFailedNotUseSR
             WorkerHealed              = evWorkerHealed
             ProtectionHealed          = evProtectionHealed
             SwappingRoles             = evSwappingRoles
             RetryTimer                = evRetryTimer, evRetryTimeout
             ActivateLsp               = evActivateLspXConnSet,
evActivateLspXConnUnset, evActivateLspNone, evActivateLspErr
             SRLspRevert               = evSRLspRevertReq, evSRLspRevertAck,
evSRLspRevertNack, evSRLspRevertErr
             RecoveryManualTrigger     = evRecoveryManualTrigger
             RetryRecovery             = evRetryRecoveryOk, evRetryRecoverySROk,
evRetryRecoveryKo
             ProtectionRedo            = evProtectionRedoOk, evProtectionRedoErr
```

```
              ProtectionDismiss         = evProtectionDismissOk,
evProtectionDismissErr


#
# States
#
# state = state1   [The first state is the start one if start-state is not set]
#         eventX -> dstState
#
# state = state2
#         eventY -> dstState
#

{ States }

#
state = Down
              evWorkerInstalled         -> OneConnection
              evProtectionInstalled           -> OneConnection    # if 1+1 and slow
ResvConf on worker
              evWorkerSigErr                  -> .
              evProtectionSigErr        -> .

#
state = OneConnection
              evWorkerInstalled         -> LspBackupInstalled
              evProtectionInstalled           -> LspBackupInstalled
              evWorkerFailedUseSR       -> RestoringSoft
              evWorkerFailedUseHR       -> RestoringHard
              evWorkerFailedMngErr            -> .
              evWorkerFailedNoAction          -> .
              evWorkerHealed                  -> .
              #evWorkerSigErr                 -> .
              evProtectionSigErr        -> .
              evWorkerDeleted                 -> Down
              evProtectionDeleted       -> .
              evRecoveryManualTrigger         -> RestoringSoft
              evRetryRecoveryOk         -> .
              evRetryRecoverySROk       -> RestoringSoft
              evRetryRecoveryKo         -> .
              evProtectionRedoOk        -> LspBackupInstalled
              evProtectionRedoErr       -> .
              evSRLspRevertReq          -> .
              evSRLspRevertErr          -> .
              evSRLspRevertAck          -> .
              evSRLspRevertNack         -> .

#
state = RestoringHard
              evWorkerHealed                        -> .
              evWorkerDeleted                       -> RestoredHard
              evWorkerSigErr                        -> RestoredHard
              evRetryTimer                    -> RestoredHard
              evRetryTimeout                        -> OneConnection

#
state = RestoredHard
```

```
            evWorkerInstalled                -> OneConnection
            evWorkerSigErr                         -> .
            evRetryTimer                     -> .
            evRetryTimeout                         -> Down

#
state = RestoringSoft
            evWorkerHealed                   -> .
            evProtectionInstalled            -> .
            evProtectionDeleted        -> OneConnection
            evProtectionSigErr         -> OneConnection
            evSwappingRoles                  -> RestoredSoft

#
state = RestoredSoft
            evWorkerHealed                         -> .
            evProtectionHealed               -> .
            evProtectionDeleted              -> OneConnection
            evProtectionSigErr               -> .
            evRetryTimer                     -> .
            evRetryTimeout                         -> .

#
state = LspBackupInstalled
            #evWorkerFailedNotUseSR          -> Recovering
            evWorkerFailedNoAction           -> .
            #evProtectionFailedNotUseSR      -> OneConnection
            evProtectionFailedNoAction -> .
            evProtectionHealed         -> .
            evWorkerHealed                   -> .
            evWorkerDeleted                  -> OneConnection
            evProtectionDeleted        -> OneConnection
            evActivateLspXConnSet            -> Recovering
            evActivateLspNone          -> .
            evActivateLspErr           -> .
            evSRLspRevertReq           -> .
            evSRLspRevertErr           -> .
            evSRLspRevertAck           -> .
            evSRLspRevertNack          -> .
            evRetryRecoveryOk          -> OneConnection
            evRetryRecoveryKo          -> .
            evProtectionDismissOk            -> OneConnection
            evProtectionDismissErr           -> .

#
state = Recovering
            evWorkerHealed                   -> LspBackupInstalled
            evProtectionHealed         -> .
            #evProtectionFailedNotUseSR      -> Reprotecting
            evProtectionFailedNoAction -> .
            evActivateLspXConnUnset          -> LspBackupInstalled
            evActivateLspNone          -> .
            evActivateLspErr           -> .
            evProtectionDeleted        -> OneConnection
            evRetryRecoveryOk          -> Reprotecting
            evRetryRecoveryKo          -> .
            evProtectionDismissOk            -> Reprotecting
            evProtectionDismissErr           -> .
```

```
#
state = Reprotecting
             evWorkerHealed                  -> OneConnection
             evProtectionInstalled           -> Recovering
             evProtectionSigErr        -> .
             evWorkerDeleted                 -> Down
             evProtectionDeleted       -> .
             evRetryRecoveryOk         -> .
             evRetryRecoveryKo         -> .
             evProtectionRedoOk        -> Recovering
             evProtectionRedoErr       -> .
```

Code 9-1: RC Recovery Bundle FSM.

The RB states are reported in the following table. The steady ones (depending on the recovery type) have their names in italic.

State	short description
Down	The RB has been created, but has either no LSPs, or signalling on its LSP hasn't occurred yet. Steady state.
OneConnection	The RB has one of its LSPs installed (i.e. up). Arrival state for some recovery types (e.g. unprotected or rerouting), or transient state for others (which still need the backup LSP to be installed).
RestoringHard	A hard rerouting (i.e. a break-before-make on-the-fly restoration) has begun, but not yet finished: here waiting for the worker LSP to be torn down.
RestoredHard	Still in hard rerouting. The former (and failed) worker deletion has been carried out, and the setup of a new worker LSP is now initiated. When the new worker LSP will be installed successfully, the RB will go back to its *OneConnection* steady state.
RestoringSoft	A soft rerouting (i.e. a make-before-break on-the-fly restoration) has begun, but not yet finished: here waiting for the backup LSP to be ready (i.e. installed).
RestoredSoft	Still in soft rerouting. The new backup LSP setup has been carried out, and the deletion of the former (and failed) worker LSP is now initiated. When the former worker LSP will be deleted successfully, the RB will go back to its *OneConnection* steady state.
LspBackupInstalled	The RB has now 2 LSPs: steady state for any recovery (i.e. protection) scheme based on 2 LSPs.
Recovering	State where the activation of a pre-planned backup LSP is in progress, caused by a failure in the worker LSP. Steady state until the worker LSP heals, then back to *LspBackupInstalled*.
Reprotecting	Substituting the backup LSP (be it pre-planned or not).

Table 9-1: RC Recovery Bundle FSM: states

The following table reports the root events that feed the FSM. When a root event might result in different detailed events, this is discussed case by case.

Root event	short description
WorkerInstalled	The worker LSP signalling (by G^2.RSVP-TE) has successfully completed, and the LSP is now up and running.
ProtectionInstalled	The backup LSP signalling (by G^2.RSVP-TE) has successfully completed, and the LSP is now up and running.
WorkerSigErr	Some error(s) occurred during the signalling (by G^2.RSVP-TE) of the worker LSP, and its setup failed.
ProtectionSigErr	Some error(s) occurred during the signalling (by G^2.RSVP-TE) of the backup LSP, and its setup failed.
WorkerDeleted	The teardown (by G^2.RSVP-TE) of the worker LSP has successfully completed; no instance of that LSP exists anymore at G^2.RSVP-TE.
ProtectionDeleted	The teardown (by G^2.RSVP-TE) of the backup LSP has successfully completed; no instance of that LSP exists anymore at G^2.RSVP-TE.
WorkerFailed	A Data (aka Transport) Plane failure (i.e. alarm) raised somewhere along the worker LSP; G^2.RSVP-TE might or might not have more detailed information of what happened, and where (i.e. at which node/link). Depending on the properties of this RB, this root event will result in a detailed event that brings to some next restoration state (e.g. soft or hard rerouting).
ProtectionFailed	A Data (aka Transport) Plane failure (i.e. alarm) raised somewhere along the backup LSP; G^2.RSVP-TE might or might not have more detailed information of what happened, and where (i.e. at which node/link).
WorkerHealed	The failure (aka alarm) at the worker LSP has disappeared “spontaneously”, i.e. without the intervention of any recovery procedure by the RC.
ProtectionHealed	The failure (aka alarm) at the backup LSP has disappeared “spontaneously”, i.e. without the intervention of any recovery procedure by the RC.
SwappingRoles	During a soft rerouting (aka make-before-break on-the-fly) restoration, the roles of the backup LSP and of the worker LSP have “swapped”, i.e. the configuration of transport network resources at the two ends of the LSP (e.g. the SNCP in SDH) have changed into a condition where the former worker LSP is not the backup one, and vice versa. For some TN technology, waiting the swap is necessary in order to be able to tear down the former worker LSP (now backup).
RetryTimer	See *RetryRecovery* below; this root event only applies during the hard rerouting restoration (when a failure in recovering is very dangerous: it might leave the RB – and the Call – with no LSPs under it).
ActivateLsp	This event allows to “activate” a pre-planned backup LSP, i.e. change it from a planned one (with either some forms of pre-signalling or not) into a real LSP. Depending on the RB properties and the RB setup/teardown phase, this root event might result in making TN cross-connections, undoing them, or just no action.
SRLspRevert	If the RB supports a soft rerouting restoration with reversion (i.e. a revertive on-the-fly restoration), it will keep the former (and alarmed) worker LSP. This event indicates that it is time to revert back to the former LSP (e.g. since the alarm on it has cleared).
RecoveryManualTrigger	This event emulates the occurrence of a alarm on the worker LSP: the RB recovery behaviour is triggered via management procedures.

RetryRecovery	Something has failed during a recovery attempt, and a timer has been set to try to recover in the future, or, at least, to clean up the situation and revert back to a steady state. This event might result in either an actual new attempt, or in stopping any future attempts (e.g. the number of maximum retry times have been reached).
ProtectionRedo	Try again adding a backup LSP to this RB (it previously failed due to some signalling reasons, probably).
ProtectionDismiss	Stop trying adding a backup LSP to this RB; the RB might end in a steady state that is not the one foreseen by its recovery type.

Table 9-2: RC Recovery Bundle FSM: root events

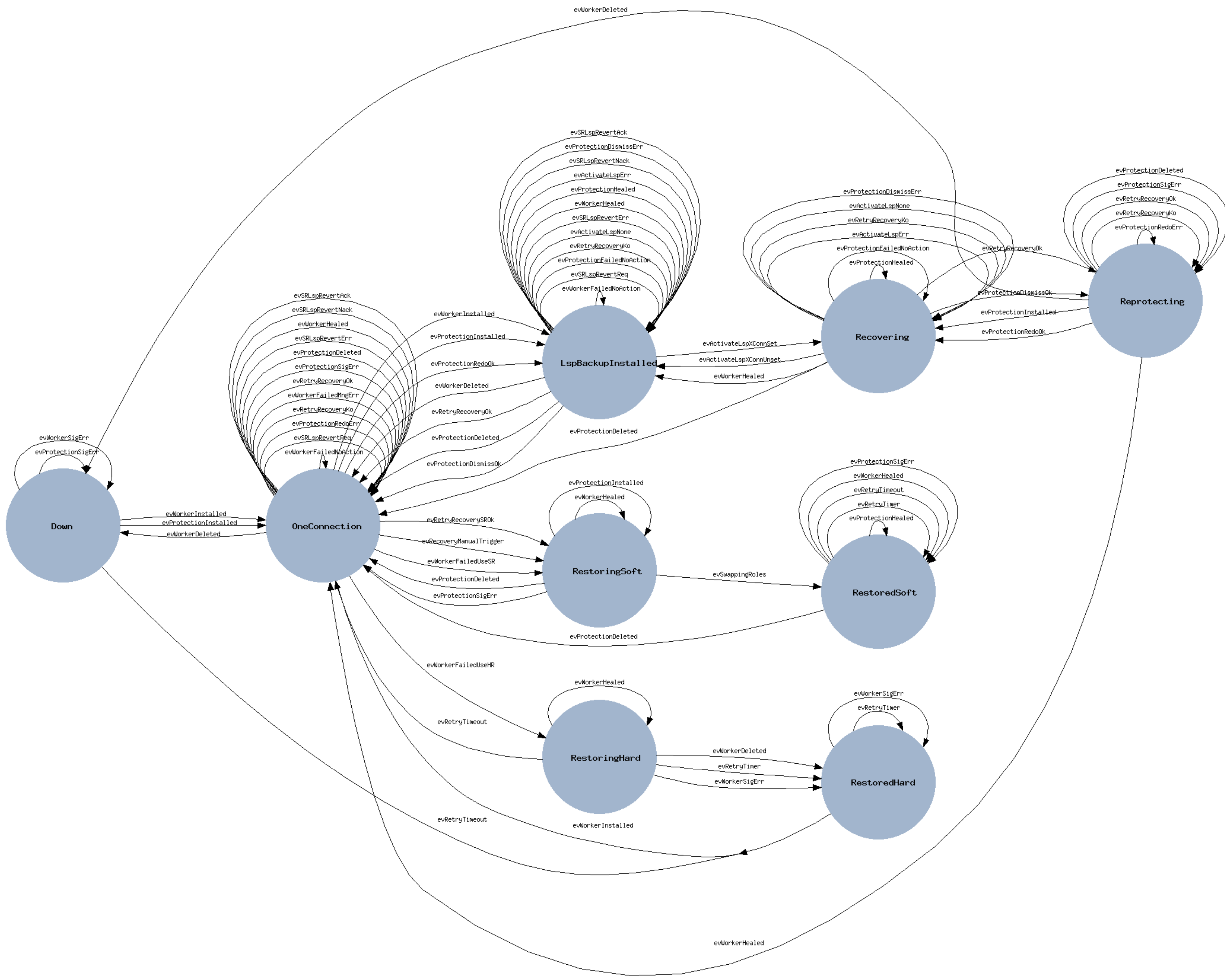

Figure 9-3: Recovery Bundle FSM

9.5 Recovery Controller External APIs

The API for the Recovery Controller is specified in *<sw_root>/idl /RecoveryController.idl*, and reported in Code 9-2. The API has two CORBA interfaces in the *RecoveryController* module: *NorthBound* and *SouthBound.*

The *NorthBound* interface implements the communication between the Network Call Controller and the Recovery Controller, in the southbound direction (i.e. commands from the NCC to the RC).

The *SouthBound* interface is used by the Recovery Controller to receive notifications from the G².RSVP-TE about the handled LSPs.

```
#include "types.idl"
#include "g2mplsTypes.idl"

module RecoveryController {
       interface NorthBound {

             typedef sequence<g2mplsTypes::recoBundleIdent>             rbIdentSeq;

             boolean
             rbCreate(in g2mplsTypes::recoBundleIdent      id,
                          in g2mplsTypes::callIdent              callId,
                          in g2mplsTypes::callParams                   callInfo,
                          in g2mplsTypes::recoveryParams         recoveryInfo,
                          in g2mplsTypes::lspParams              lspInfo,
                          in boolean                                   setup)
                   raises(Types::InternalProblems);

             boolean
             rbAddEroPart(in g2mplsTypes::recoBundleIdent  id,
                                in g2mplsTypes::eroSeq     eroItem)
                   raises(Types::InternalProblems, Types::CannotFetch);

             boolean
             rbEnable(in g2mplsTypes::recoBundleIdent      id)
                   raises(Types::InternalProblems, Types::CannotFetch);

             boolean
             rbDisable(in g2mplsTypes::recoBundleIdent     id)
                   raises(Types::InternalProblems, Types::CannotFetch);

             boolean
             rbDestroy(in g2mplsTypes::recoBundleIdent     id)
                   raises(Types::InternalProblems, Types::CannotFetch);

             boolean
             rbSetUp(in g2mplsTypes::recoBundleIdent id)
                   raises(Types::InternalProblems, Types::CannotFetch);
```

```
		boolean
		rbSetDown(in g2mplsTypes::recoBundleIdent     id)
			raises(Types::InternalProblems, Types::CannotFetch);

		rbIdentSeq getRecoBundles()
			raises(Types::InternalProblems);

		boolean
		rbGetDetails(in  g2mplsTypes::recoBundleIdent id,
				       out g2mplsTypes::callIdent            callId,
				       out g2mplsTypes::recoveryParams       recoveryInfo,
				       out g2mplsTypes::lspParams            lspInfo,
				       out g2mplsTypes::statesBundle         states)
			raises(Types::InternalProblems, Types::CannotFetch);

	};

	interface SouthBound {
		enum lspDetailedEvent {
			LSPDETAILEDEVENT_PATH,
			LSPDETAILEDEVENT_RESV,
			LSPDETAILEDEVENT_CONFIRM,
			LSPDETAILEDEVENT_NOTIFY,
			LSPDETAILEDEVENT_DOWN,
			LSPDETAILEDEVENT_ERR
		};

		enum lspEvent {
			LSPEVENT_READY,
			LSPEVENT_SIGERROR,
			LSPEVENT_FAILURE,
			LSPEVENT_HEALING,
			LSPEVENT_GOINGDOWN
		};

		struct eventInfo {
			lspEvent	event;
		};

		enum sigPhase {
			SIGPHASE_SETUP,
			SIGPHASE_TEARDOWN,
			SIGPHASE_RECOVERY
		};

		enum tnResourceAction {
			TNRESOURCEACTION_XCONN,
			TNRESOURCEACTION_PROTECT,
			TNRESOURCEACTION_JOIN      /* for MRN */
		};

		boolean notifyLspNew(in g2mplsTypes::lspIdent       lspident,
				       in g2mplsTypes::callIdent       callId,
				       in g2mplsTypes::lspParams       info);

		boolean notifyLspDeleted(in g2mplsTypes::lspIdent         ident);

		boolean notifyLspEvent(in g2mplsTypes::lspIdent      lspident,
```

```
                                        in eventInfo       evinfo);

            boolean tellTNResourceAction(in g2mplsTypes::lspIdent      lspident,
                                        in sigPhase                phase,
                                        out tnResourceAction       action);
      };
};
```

Code 9-2: Recovery Controller external APIs IDL.

The methods for the *NorthBound* interface are:

- *rbCreate()*: allows the NCC to create (and start setting up, if the flag is set) a new RB at the RC. The Call ID is passed down to the RB and stored, to create an bi-directional association between the Call and the RB, and to allow the RB to later pass the Call ID to the G².RSVP-TE for LSP signalling purposes. Same applies to the Call parameters.

- *rbAddEroPart()*: allows the NCC to add a piece of Explicit Route to the newly created RB (it has to be still "Down"). This might be useful in some contexts, e.g. if the NCC would need to set an RB scope (i.e. destination node) narrower than the whole domain.

- *rbEnable()* and *rbDisable()*: allow to set the administrative status of the RB to "enabled" and "disabled", respectively. This is for future use, e.g. to temporarily make an RB unavailable for usage, without tearing it down.

- *rbDestroy():* allows to remove a newly created RB (it has to be still "Down"). In that status, no evolution has occurred yet, and the RB cannot disappear as a consequence of a teardown. An explicit command is needed.

- *rbSetUp()* and *rbSetDown()*: the access points for setting up and tearing down the RB, respectively. When *rbSetUp()* is invoked, the Recovery Controller will start adding the needed LSPs to the RB, and telling the G².RSVP-TE to set them up. Vice versa for the tear down procedure.

- *getRecoBundles()*: allows to retrieve the list of the IDs of the RBs currently present at the RC.

- *rbGetDetails()*: allows to retrieve the details of a specific RB (associated Call ID, recovery parameters, LSP parameters,states). Further information is retrieved by:

The methods for the *SouthBound* interface are:

- *notifyLspNew():* the signalling of a new LSP has reached this RC (usually located at the egress Border Controller of the domain, since RB signalling starts from the ingress, as a practical rule). The new LSP can be associated to a specific Call thanks to the Call ID transported, and thus to the specific RB owned by that Call. As a result of this notification, the triplet <Call, RB, LSP> is bundled.

- *notifyLspDeleted():* the teardown of an LSP has completed; G².RSVP-TE will destroy this LSP instance soon exiting this method, and the RB has to align with that and evolve its FSM accordingly.

- *notifyLspEvent():* invoked by G².RSVP-TE to notify some specif events on the LSP that might be of interest for the RB, i.e.:

    - *LSPEVENT_READY*: the LSP setup signalling has successfully completed: the LSP is up and running, and *installed* from the RB's viewpoint.

    - *LSPEVENT_SIGERROR*: some signalling errors have occurred on this LSP, either during the setup or teardown phases.

    - *LSPEVENT_FAILURE*: failure (aka alarming) of some transport network resources (i.e. node or link) along the path of this LSP. G².RSVP-TE might or might not know more about this failure.

    - *LSPEVENT_HEALING*: the previously mentioned failure has disappeared; the LSP is working again now.

    - *LSPEVENT_GOINGDOWN*: the teardown signalling of this LSP has begun, and not as a result of a previous *rbSetDown()* from this RC (i.e. probably the other-end RB has started a teardown of the LSP).

- *tellTNResourceAction()*: invoked by G².RSVP-TE at LSP end nodes (either ingress or egress) to know what exactly it should do when installing transport network resources. This is a critical action, where only the RB knows exactly what to do, since the action depends much on information beyond the single LSP treated by the G².RSVP-TE: the role of the LSPs (i.e. worker or backup), its relationship with other LSPs in the RB, etc. Depeding on the signalling phase, the basic actions could be:

    - *TNRESOURCEACTION_XCONN*: ask the TNRC to create a simple cross-connection.

    - *TNRESOURCEACTION_PROTECT*: ask the TNRC to add a protection to a previously existing cross-connection, with this protecting label.

    - *TNRESOURCEACTION_JOIN*: ask the TNRC to stitch resources (i.e. labels) belonging to different ISCs (i.e. Interface Switching Capabilities). Needed for the future support of Multi-Region Network / Multi-Layer Network (MRN/MLN) features.

10 G^{2}MPLS Path Computation Engine Routing Algorithm (G^2.PCE-RA)

10.1 G^2.PCE-RA basics

In the Phosphorus-g2mpls stack the roles of the G^2.PCE-RA are to:

- store the global view of the network topology (multi-domain including also grid sites with their own resources)
- provide an interface for the other modules to request routes and other routing queries (e.g. TNA resolution) across the overall topology.

For these purposes, G^2.PCE-RA interacts with:

- the OSPF process, which exports LSDB contents in terms of G^2.PCE-RA data structures;
- the Network Call Controller process , which is the main requester for call routes, topology queries, etc.
- the G.RSVP-TE process, which requests G^2.PCE-RA for ERO computation/completion in case of sparse EROs during LSP signalling or crankback;
- G^2.PCE-RA VTY interface, which is mainly used for printing topology/module information and testing the path computation module by means of dummy requests[9].

The G^2.PCE-RA component is broken down into sub-components, each responsible fro specific tasks.

The G^2.PCE-RA Thread Master manages and schedules the activities of the QUAGGA pseudo-threads of the G^2.PCE-RA process, thus coordinating the incoming/outgoing messages from the two external interfaces, the IPC middleware stratum and the VTY.

[9] The dummy route computations does not imply the signaling of the produced ERO(s), but they have impact on the bandwidth estimation mechanism of the G^2.PCE-RA; thus, subsequent call route requests from NCC or LSP route from G.RSVPTE could not be fulfilled, if they are done within the expiration time of the estimation and no topology update has occurred in the meanwhile.

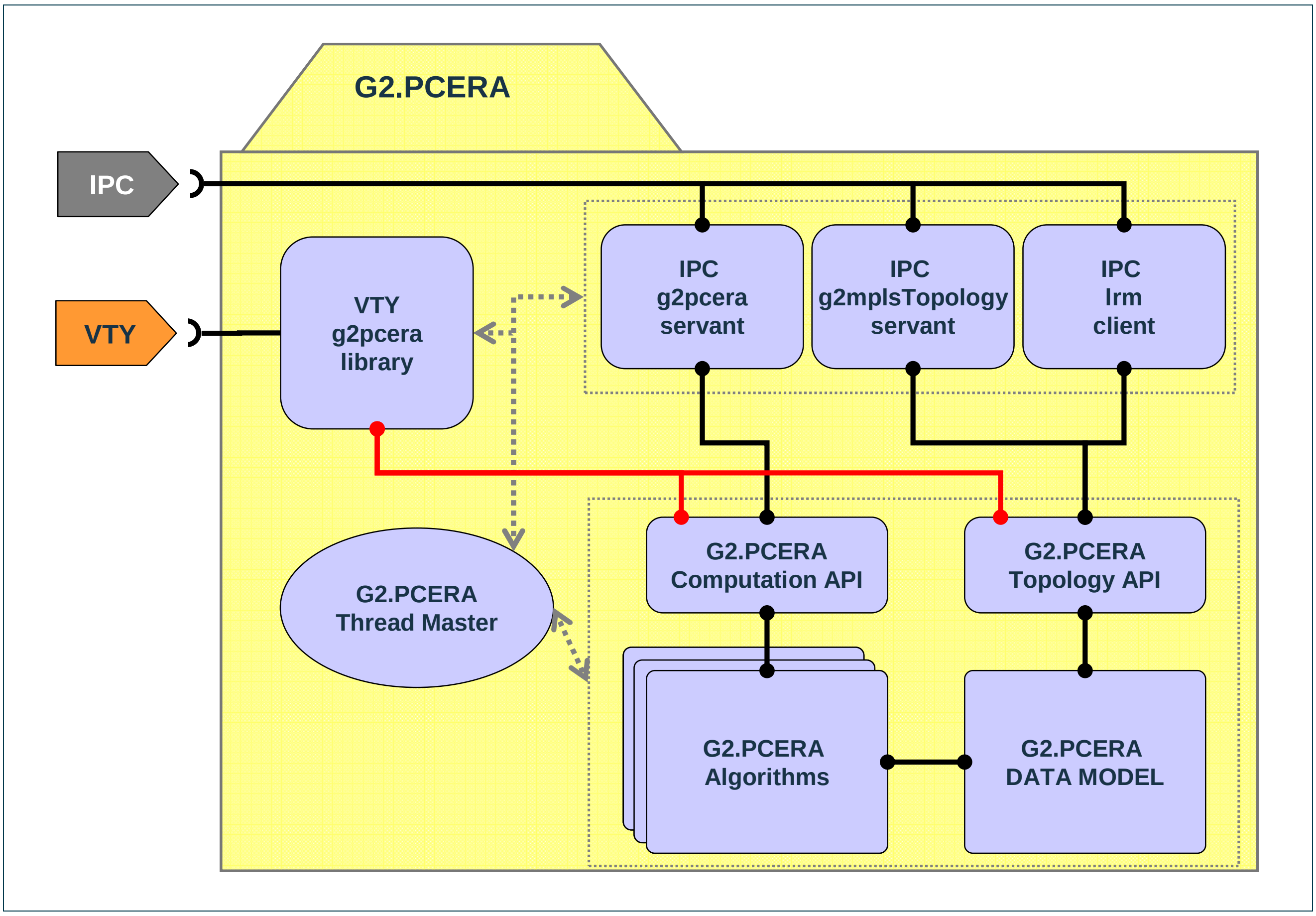

Figure 10-1: The G^2.PCE-RA component break-down.

The IPC G^2.PCE-RA servants implement the CORBA sever side for the topology updates and the route computations, while the IPC LRM client is used by the G^2.PCE-RA to retrieve the routerID of the hosting G^{2}MPLS controller, which will act in the topology as root node.

The VTY G^2.PCE-RA library implements the specific G^2.PCE-RA VTY commands (parsing and processing) for printing topology/module information and testing the path computation module by means of dummy requests.

G^2.PCE-RA data model and algorithms sub-components represent the core engine of the overall process and are detailed in the following sections for sake of clarity.

The CORBA client and servants and the VTY library are interfaced to the core G^2.PCE-RA processing engine through an internal common API, which is split in two namespaces: one for topology, the other for computations.

The first API namespace, i.e. topology, is used to directly access (create, update, destroy) data model structures, in which the Grid and network topology is stored.

On the contrary, the computations API directly access the G^2.PCE-RA algorithms, which run in a nearly read-only mode on the topology, get the topology view as it is at the time of request execution and can just update Path Computation related elements of the data model.

10.2 Topology view in G^2.PCE-RA

The G^2.PCE-RA topological view of the network is always node-centric, because each G^2MPLS controller in the domain builds up its own topology map depending on the information managed by the IGP (OSPF-TE). According to the G^2MPLS specifications [PH-WP2-D2.1] and [PH-WP2-D2.2], the G^2.PCE-RA on a NE holds a complete TE detail of the Area/Domain[10] it belongs to. In case of multi-domain operation across the E-NNI, the G^2.PCE-RA holds a summarized view of the other domains, in terms of inter-domain links between domains and – optionally – intra-domain links within the domains. This hierarchical routing model provides a more scalable approach to the inter-domain problem.

An example topology view that can be built into G^2.PCE-RA is provided in the following Figure 10-2 and Figure 10-3.

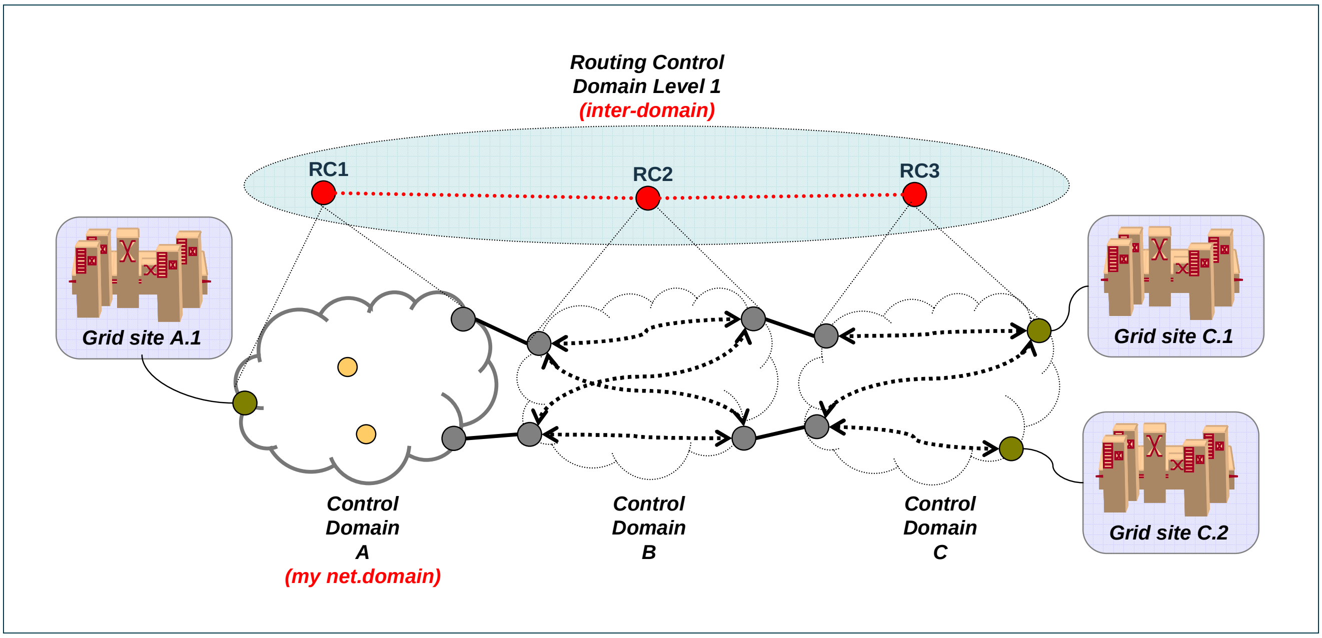

Figure 10-2: Mixed topology with three domains, inter and intra-domain te-links and Grid sites.

[10] In this document Area and Domain are synonymous, since no support for multi-area routing within a single control domain is needed.

Control Domain A is the root domain, it connects a Grid site with resources (GN1), it is attached to Control Domain B (transit), can reach Control Domain B (transit) that connects other Grid sites with resources (GS-C.1 and GS-C.2).

The resulting topology view is shown in Figure 10-3. G^2.PCE-RA contains in its topology the three Grid nodes (GS-A.1, GS-C.1 and GS-C.2) and a number of network nodes, some of them belonging to the domain core (NN2, NN6), others operating on the domain edge (NN1), others acting as domain border nodes (NN3, NN5) and others learns as domains (RC2, RC3).

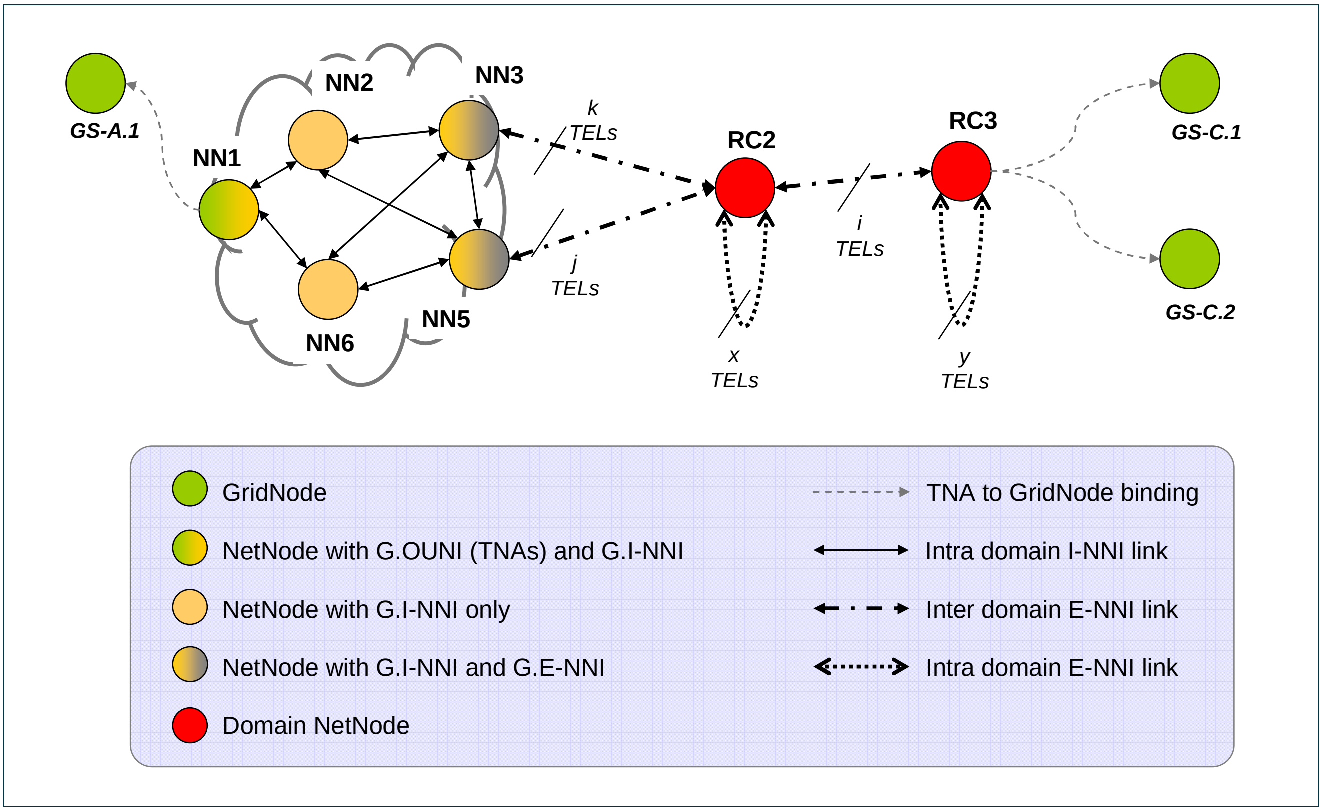

Figure 10-3: G^2.PCE-RA representation of the previous mixed topology.

Bidirectional connectivity between network nodes (domain or not) is obtained though TE-Links (intra-domain I-NNI, inter-domain E-NNI or intra-domain E-NNI).

Association between Grid nodes and their Provider Edge routers is maintained through the TNA.

The topology concept is implemented by the G^2.PCE-RA data model described in the following section.

10.3 G².PCE-RA data model

The G².PCE-RA data model is sketched in Figure 10-4, and follows a hierarchical structure in which an ancestor element includes a set of child sub-elements, reported in the following sub-sections with their contents hghlighted.

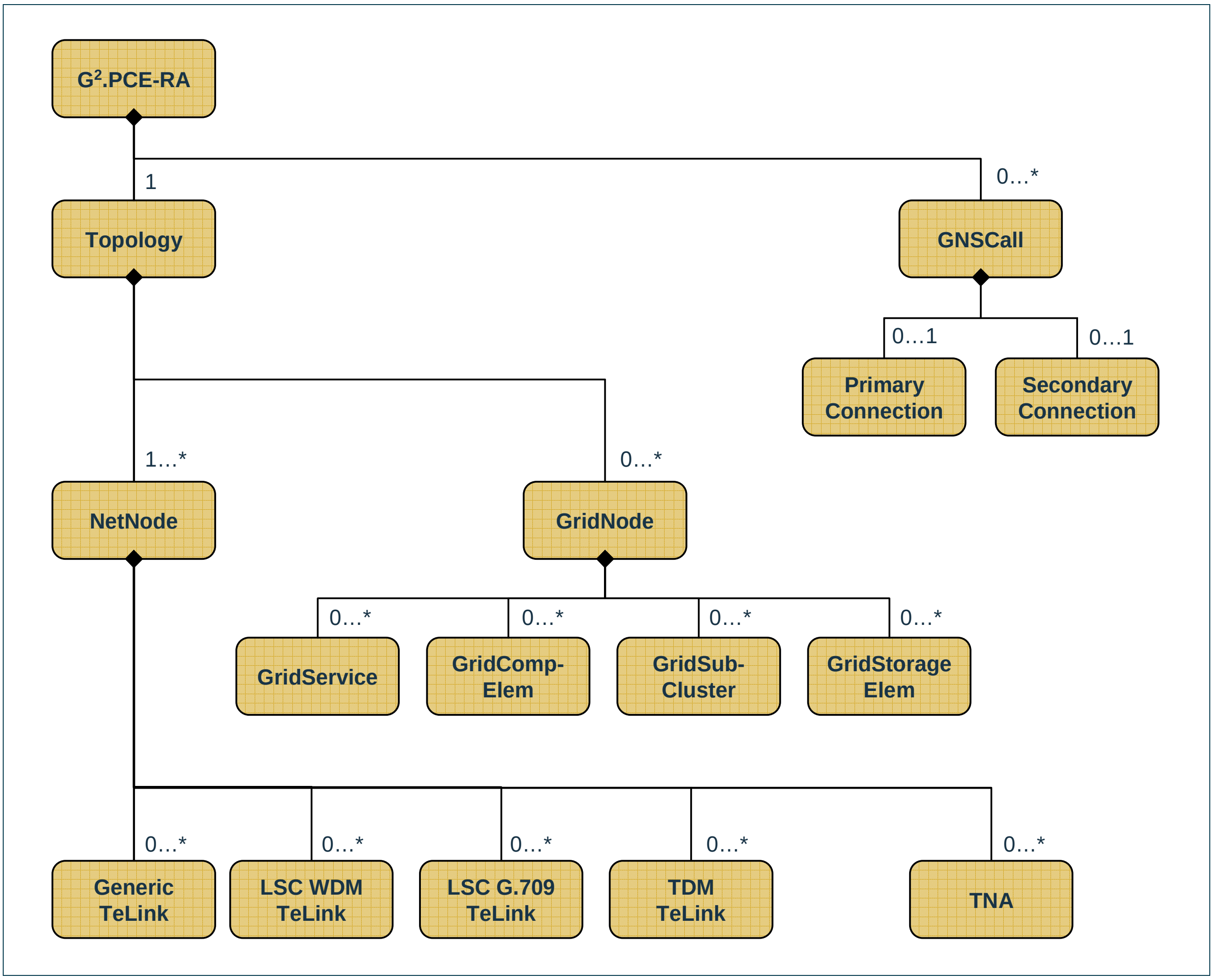

Figure 10-4: The base G².PCE-RA data model.

The hierarchy mechanism has been generalized though templates as shown in the following:

```
        template <bool REP, class P> class Ancestor {
        public:
```

```
		Ancestor(P * p) {
			//assert(p);

			parent_      = p;
			reparenting_ = REP;
		}
		~Ancestor(void) {
			// mindless
			parent_ = 0;
		}

		// returns the parent
		P * parent(void) {
			return parent_;
		}

		// assign the new parent and returns the previous one
		P * reparent(P * p) {
			//assert(p);

			P * tmp;

			if (!reparenting_) {
				G2PCERA_ERROR("Reparenting not allowed!");
				return 0;
			}

			tmp = parent_;
			parent_ = p;

			return tmp;
		}

	private:
		P *	parent_;
		bool	reparenting_;
	};
```

Code 10-1: Ancestor template.

Most of the types used on the G^2.PCE-RA data model elements can be found in *<sw_root>/lib/g2mpls_types.h* and *<sw_root>/g2pcerad/g2pcera_common.hh.*

10.3.1 G^2.PCE-RA instance

```
	class G2PCERA {
	public:
		G2PCERA(void);
		G2PCERA(uint32_t	rootNodeId,
			spfType_t	spfSelector = spfDijkstra);

		~G2PCERA(void);
```

```
        uint32_t            rootNodeId(void);
        spfType_t           spfSelector(void);

        bool                loadTopology(void);
        const Topology *    getTopology(void);

        bool                attachGnsCall(GnsCall *         ptr);
        bool                detachGnsCall(GnsCall *         ptr);
        GnsCall *           getGnsCall(const call_ident_t & id);
        std::list<GnsCall *>      getGnsCalls(void);
        size_t              getGnsCallsCount(void);

private:
        gnsCallIdent_t            callIdMangle(const call_ident_t & id);

        spfType_t                               spfSelector_;
        uint32_t                                rootNodeId_;
        Topology *                              topology_;
        std::map<gnsCallIdent_t, GnsCall *>     gnsCalls_;
};
```

Code 10-2: G².PCE-RA instance.

10.3.2 GNS calls

```
// map keys
typedef std::string gnsCallIdent_t;  // key

class GnsCall: public Ancestor<true, G2PCERA> {
public:
        GnsCall(G2PCERA *                 parent,
                const call_ident_t &            ident,
                const ero_hop_t &         srcHop,
                const ero_hop_t &         dstHop,
                const call_info_t &       callInfo,
                const recovery_info_t &         recInfo,
                const lsp_info_t &        lspInfo);
        ~GnsCall(void);

        call_ident_t        ident(void);

        void                dump(std::string &  prefix,
                                 bool           recursive);


        bool                attachConnection(lsp_role_t      role,
                                         Connection *ptr);
        bool                detachConnection(lsp_role_t      role,
                                         Connection *ptr);
        Connection *        getConnection(lsp_role_t  role);

        disjointness_level_t      getDisjoinnessLevel(void);

        bool                isConfirmed(void);
        void                isConfirmed(bool    flag);
```

```
    private:
            void                checkDisjoinnessLevel(void);

            call_ident_t                ident_;
            ero_hop_t                   srcHop_;
            ero_hop_t                   dstHop_;
            call_info_t                 callInfo_;
            recovery_info_t             recInfo_;
            lsp_info_t                  lspInfo_;

            Connection *                primary_;
            Connection *                secondary_;

            disjointness_level_t        disjoinnessLevel_;

            bool                        isConfirmed_;
    };
```

Code 10-3: GNS calls.

10.3.3 Connections

```
    class Connection: public Ancestor<true, GnsCall> {
    public:
            Connection(GnsCall *        parent,
                      lsp_ident_t       ident);
            ~Connection(void);

            void                dump(std::string &  prefix,
                                     bool           recursive);

            bool                addEroHop(bool            onTop,
                                    const ero_hop_t & hop);
            bool                delEroHop(const ero_hop_t &      hop);

            std::list<ero_hop_t>        getEro(void);

    private:
            lsp_ident_t                 ident_;
            std::list<ero_hop_t>        ero_;
    };
```

Code 10-4: Connections.

10.3.4 Topology

```
    // map keys
    typedef uint32_t    nodeKey_t;
```

```
class Topology: public Ancestor<true, G2PCERA> {
public:
      Topology(G2PCERA *  parent);
      ~Topology(void);

      bool                 attachNode(Node *  ptr);
      bool                 detachNode(Node *  ptr);
      Node *               getNode(node_ident_t      ident);
      GridNode *           getGridNode(uint32_t       peRouterId);

      std::list<Node *>    getNodes(void);
      bool                 modTotNodesCount(uint32_t  howMany,
                                          bool         add);
      bool                 modTotLinksCount(uint32_t  howMany,
                                          bool         add);
      bool                 modTotTnasCount(uint32_t   howMany,
                                         bool         add);

      void                 dump(std::string &  prefix,
                                bool          recursive);
      bool          getData(topology_summary_data_t &      data);

private:
      uint32_t                          totNodes_;
      uint32_t                          totLinks_;
      uint32_t                          totTnas_;

      // Maximum link     cost in the topology
      uint32_t                          maxLinkCost_;
      // Global SPF revision number
      uint32_t                          spfRevision_;

      std::map<nodeKey_t, Node *>       nodes_;
};
```

Code 10-5: Topology.

10.3.5 Nodes

```
class Node: public Ancestor<true, Topology>  {
public:
      Node(topo_node_type_t      type,
           uint32_t       id);
      ~Node(void);

      topo_node_type_t    type(void);
      node_ident_t        ident(void);

      void                dump(std::string &  prefix);

private:
      topo_node_type_t    type_;
      uint32_t            id_;
};
```

Code 10-6: Node.

10.3.5.1 *NetNode*

```
class NetNode: public Node {
public:
      NetNode(uint32_t          id,
            net_node_data_t &   data);
      ~NetNode(void);

      void               dump(std::string &  prefix,
                              bool           recursive);

      bool               setData(const net_node_data_t &  data);
      bool               getData(net_node_data_t & data);

      bool               attachOutLink(TeLink *    ptr);
      bool               detachOutLink(TeLink *    ptr);
      std::list<TeLink *> getOutLinks(void);
      size_t             getOutLinksCount(void);
      TeLink *           getOutLink(const telink_ident_t & id);

      bool               attachTna(Tna *     ptr);
      bool               detachTna(Tna *     ptr);
      std::list<Tna *>   getTnas(void);
      size_t             getTnasCount(void);
      Tna *              getTna(const g2mpls_addr_t & id);

      bool               attachCandElems(Node *    ptr);
      bool               detachCandElems(Node *    ptr);
      std::list<Node *>  getCandElems(void);

      uint32_t           rootCost(void);
      void               rootCost(uint32_t   newCost);

      uint32_t           spfRevision(void);
      void               spfRevision(uint32_t      newRev);

      uint8_t                  nodeFlags(void);    // bitmask
      void               nodeFlags(uint8_t   newMask);

      bool               attachFwdLink(TeLink *  ptr);
      bool               detachFwdLink(TeLink *  ptr);
      std::list<TeLink *> getFwdLinks(void);

      bool               fitInConstraints(uint32_t colors,
                                          uint16_t    area);

private:
      bool                     is_domain_;
      opstate_t                op_state_;
      admstate_t               adm_state_;
      uint32_t                 te_colors_;
      std::list<uint16_t>      areas_;

      std::list<Tna *>         tnas_;
```

```
            std::list<TeLink *>         links_; // outgoing links

            // Path Computation related members
            std::list<Node *>           candidateElems_;
            uint32_t                    rootCost_;
            uint32_t                    spfRevision_;
            uint8_t                             nodeFlags_;  // bitmask
            std::list<TeLink *>         fwdLinks_;    // cand. links

    };
```

Code 10-7: Net Node.

10.3.5.2 *GridNode*

```
    class GridNode: public Node {
    public:
            GridNode(uint32_t          id,
                    grid_site_data_t & data);
            ~GridNode(void);

            void                dump(std::string &  prefix,
                                     bool           recursive);

            bool                setData(const grid_site_data_t & data);
            bool                getData(grid_site_data_t &data);
            uint32_t            getPeRouterId(void);

            bool                attachGridServices(GridService *   ptr);
            bool                detachGridServices(GridService *   ptr);
            GridService *       getGridService(void /* policy*/);
            GridService *       getGridService(uint32_t  id);

            bool                attachGridCE(GridCompElem * ptr);
            bool                detachGridCE(GridCompElem * ptr);
            GridCompElem *              getGridCompElem(void /* policy*/);
            GridCompElem *              getGridCompElem(uint32_t  id);


            bool        attachGridSubClusters(GridSubCluster * ptr);
            bool        detachGridSubClusters(GridSubCluster * ptr);
            GridSubCluster *    getGridSubCluster(void /* policy*/);
            GridSubCluster *    getGridSubCluster(uint32_t  id);

            bool                attachGridSE(GridStorageElem * ptr);
            bool                detachGridSE(GridStorageElem * ptr);
            GridStorageElem *   getGridStorageElem(void /* policy*/);
            GridStorageElem *   getGridStorageElem(uint32_t  id);

    private:
            std::string *                       name_;
            geo_coords_t                        location_;
            uint32_t                            peRouterId_;

            std::map<uint32_t, GridService *>       gridServices_;
```

```
		std::map<uint32_t, GridCompElem *>	gridCompElems_;
		std::map<uint32_t, GridSubCluster *>	gridSubClusters_;
		std::map<uint32_t, GridStorageElem *>	gridStorageElems_;
	};
```

Code 10-8: Grid Node.

10.3.5.3 *Grid Subnodes*

```
	template <class DATA>
	class GridSubNode : public Ancestor<true, GridNode> {
	public:
		GridSubNode(grid_subnode_ident_t ident,
			    DATA &		    data) :
			Ancestor<true, GridNode> (0) {
			//assert(p);

			ident_ = ident;
			data_  = data;
		}

		~GridSubNode(void) {
			// mindless
		}

		grid_subnode_ident_t	ident(void) {
			return ident_;
		}

		bool			setData(const DATA & data) {
			data_ = data;
			return true;
		}

		bool			getData(DATA & data) {
			data = data_;
			return true;
		}

		void			dump(std::string &  prefix) {
			G2PCERA_DEBUG("%s"DUMP_BAR1, prefix.c_str());

			prefix += DUMP_TAB;

			logDump(prefix, ident());

			DATA data;
			if (!getData(data)) {
				G2PCERA_ERROR("Cannot get data "
					      "from object in %s",
					      __PRETTY_FUNCTION__);
				return;
			}

			logDump(prefix, data);
```

```
                G2PCERA_DEBUG("%s"DUMP_BAR2, prefix.c_str());
        }


    private:
        grid_subnode_ident_t        ident_;
        DATA                        data_;
    };
```

Code 10-9: Grid Subnodes.

```
    class GridService: public GridSubNode<grid_service_data_t> {
    public:
        GridService(grid_subnode_ident_t subNodeId,
                grid_service_data_t & data) :
            GridSubNode<grid_service_data_t>(subNodeId, data) {};
        ~GridService(void) {};
    };

typedef struct grid_service_data_mask {
    uint32_t                        data:1;
    uint32_t                        state:1;
    uint32_t                        endpoint_addr:1;
} grid_service_data_mask_t;

typedef struct grid_service_data {
    grid_service_data_mask_t        mask_;
    grid_service_info_t             data;
    grid_service_state_t            state;
    g2mpls_addr_t                   endpoint_addr;
} grid_service_data_t;
```

Code 10-10: Grid Service subnode.

```
    class GridCompElem: public GridSubNode<grid_ce_data_t>  {
    public:
        GridCompElem(grid_subnode_ident_t      subNodeId,
                 grid_ce_data_t &     data) :
            GridSubNode<grid_ce_data_t>(subNodeId, data) {};
        ~GridCompElem(void) {};
    };

typedef struct grid_ce_data_mask {
    uint32_t                        lrms_info:1;
    uint32_t                        host_addr:1;
    uint32_t                        gatekeeper_port:1;
    uint32_t                        job_manager:1;
    uint32_t                        data_dir:1;
    uint32_t                        default_storage_elem_id:1;
    uint32_t                        jobs_state:1;
    uint32_t                        jobs_stats:1;
    uint32_t                        jobs_timeperf:1;
    uint32_t                        jobs_timepolicy:1;
    uint32_t                        jobs_loadpolicy:1;
```

```
        uint32_t                        free_job_slots_calendar:1;
} grid_ce_data_mask_t;

typedef struct grid_ce_data {
        grid_ce_data_mask_t             mask_;
        grid_lrms_info_t                lrms_info;
        g2mpls_addr_t                   host_addr;
        uint32_t                        gatekeeper_port;
        std::string *                   job_manager;
        std::string *                   data_dir;
        uint32_t                        default_storage_elem_id;
        grid_jobs_state_t               jobs_state;
        grid_jobs_stats_t               jobs_stats;
        grid_jobs_time_perf_t           jobs_timeperf;
        grid_jobs_time_policy_t         jobs_timepolicy;
        grid_jobs_load_policy_t         jobs_loadpolicy;
        std::map<uint32_t, uint16_t>    free_job_slots_calendar;
} grid_ce_data_t;
```

Code 10-11: Grid Computational Element subnode.

```
        class GridSubCluster: public GridSubNode<grid_subcluster_data_t>  {
        public:
                GridSubCluster(grid_subnode_ident_t    subNodeId,
                               grid_subcluster_data_t &  data) :
                        GridSubNode<grid_subcluster_data_t>(subNodeId, data){};
                ~GridSubCluster(void) {};
        };

typedef struct grid_subcluster_data_mask {
        uint32_t                        cpu:1;
        uint32_t                        os:1;
        uint32_t                        memory:1;
        uint32_t                        software:1;
        uint32_t                        software_env_setup:1;
        uint32_t                        subcluster_calendar:1;
} grid_subcluster_data_mask_t;

typedef struct grid_subcluster_data {
        grid_subcluster_data_mask_t     mask_;
        grid_cpu_info_t                 cpu;
        grid_os_info_t                  os;
        grid_memory_info_t              memory;
        grid_application_t              software;
        std::string *                   software_env_setup;
        std::map<uint32_t, grid_cpu_count_t>    subcluster_calendar;
} grid_subcluster_data_t;
```

Code 10-12: Grid Subcluster subnode.

```
        class GridStorageElem: public GridSubNode<grid_se_data_t>  {
        public:
                GridStorageElem(grid_subnode_ident_t   subNodeId,
                               grid_se_data_t &    data) :
                        GridSubNode<grid_se_data_t>(subNodeId, data) {};
```

```
            ~GridStorageElem(void) {};
      };

typedef struct grid_se_data_mask {
      uint32_t                        storage_info:1;
      uint32_t                        online_size:1;
      uint32_t                        nearline_size:1;
      uint32_t                        storage_area_name:1;
      uint32_t                        storage_area_path:1;
      uint32_t                        storage_area_info:1;
      uint32_t                        se_calendar:1;
} grid_se_data_mask_t;

typedef struct grid_se_data {
      grid_se_data_mask_t                   mask_;
      grid_storage_info_t                   storage_info;
      grid_storage_size_t                   online_size;
      grid_storage_size_t                   nearline_size;
      std::string *                         storage_area_name;
      std::string *                         storage_area_path;
      grid_storage_area_info_t              storage_area_info;
      std::map<uint32_t, grid_storage_count_t>      se_calendar;
} grid_se_data_t;
```

Code 10-13: Grid Storage Element subnode

10.3.6 TNAs

```
      class Tna: public Ancestor<true, NetNode> {
      public:
            Tna(const g2mpls_addr_t & id);
            ~Tna(void);

            g2mpls_addr_tident(void);
            bool        setRemNode(GridNode *           ptr);
            bool        getRemNodeIdent(node_ident_t &   ident);
            void        dump(std::string &        prefix);

      private:
            g2mpls_addr_t                   ident_;
            // used for algorithm purposes
            GridNode *                      remNode_;
      };
```

Code 10-14: TNAs

10.3.7 TE Links

```
      class TeLink: public Ancestor<true, NetNode> {
      public:
            TeLink(telink_ident_t     id);
```

```
            ~TeLink(void);

            topo_link_type_t    type(void);
            telink_ident_t            ident(void);

            void         dump(std::string &  prefix);

            bool         setData(const telink_com_data_t &      data);
            bool         getData(telink_com_data_t &            data);

            bool         setStates(const opstate_t &            op,
                                 const admstate_t &             adm);
            bool         getStates(opstate_t &                  op,
                                 admstate_t &                   adm);

            bool         appendIscs(const std::list<isc_t> &    iscs);
            bool         removeIscs(std::list<isc_t> &    iscs);
            bool         getIscs(std::list<isc_t> &       iscs);

            bool         setGenAvailBw(const avail_bw_per_prio_t & bw);
            bool         getGenAvailBw(avail_bw_per_prio_t &      bw);

            bool         appendSrlgs(const std::list<uint32_t> & srlgs);
            bool         removeSrlgs(std::list<uint32_t> &      srlgs);
            bool         getSrlgs(std::list<uint32_t> &         srlgs);
            bool         appendCalEvents(const
std::map<uint32_t,avail_bw_per_prio_t> &      cal);
            bool         removeCalEvents(std::map<uint32_t,avail_bw_per_prio_t> &
      cal);
            bool         getCalEvents(std::map<uint32_t,avail_bw_per_prio_t> &
      cal);

            bool         fitInConstraints(const cspf_constr_t & data);

            uint64_t     linkCost(void);
            void         linkCost(uint64_t   newCost);

      private:
            telink_ident_t                  ident_;

            topo_link_mode_t                mode_;
            // adminMetric_ is the base OSPF link metric
            uint32_t                        adminMetric_;

            uint32_t                        teMetric_;
            uint32_t                        teColorMask_;
            uint8_t                         teProtectionTypeMask_;
            uint32_t                        teMaxBw_;
            uint32_t                        teMaxResvBw_;

            opstate_t                       opState_;
            admstate_t                      admState_;

            std::list<isc_t>                teSwCaps_;
            avail_bw_per_prio_t             teAvailBw_;
            std::list<uint32_t>             teSrlgs_;
            std::map<uint32_t, avail_bw_per_prio_t>       teLinkcalendar_;
```

```
            // used for algorithm purposes
            Node *                          remNode_;
            TeLink *                        reverseLink_;
            int64_t                         linkCost_;
    };
```

Code 10-15: TE Links

10.3.7.1 *SDH/SONET TE Links*

```
    class TeSdhSonetLink: public TeLink {
    public:
            TeSdhSonetLink(telink_ident_t    id);
            ~TeSdhSonetLink(void);

            void         dump(std::string &  prefix);

            bool         setData(const telink_tdm_data_t &      data);
            bool         getData(telink_tdm_data_t &            data);

            bool         setTdmAvailBw(const std::list<uint32_t> & fts);
            bool         getTdmAvailBw(std::list<uint32_t> &    fts);

            bool         fitInConstraints(const cspf_constr_t & data);

    private:
            gmpls_sdhsonet_stdarbcap_t      stdArbConc_;
            uint8_t                         hoMuxCapMask_;
            uint8_t                         loMuxCapMask_;
            uint32_t                        transparencyMask_;
            uint32_t                        blsrRingId_;

            std::list<uint32_t>             freeTimeslots_;
    };
```

10.3.7.2 *LSC G.709 TeLinks*

```
    class TeG709Link: public TeLink {
    public:
            TeG709Link(telink_ident_t id);
            ~TeG709Link(void);

            void         dump(std::string &  prefix,
                              bool           recursive);

            bool         setData(const telink_lscg709_data_t &  data);
            bool         getData(telink_lscg709_data_t &        data);

            bool   setLscG709AvailBw(const std::list<uint32_t> & foduk,
                                const std::list<uint32_t> &    foch);
            bool         getLscG709AvailBw(std::list<uint32_t> & foduk,
                                std::list<uint32_t> &    foch);
```

```
        bool         fitInConstraints(const cspf_constr_t & data);

    private:
        uint8_t                    odukMuxCapMask_;

        std::list<uint32_t>        unallocODUk_;
        std::list<uint32_t>        unallocOCh_;
    };
```

Code 10-16: SDH/SONET TE Links

10.3.7.3 *LSC WDM TE Links*

```
    class TeWdmLink: public TeLink {
    public:
        TeWdmLink(telink_ident_t  id);
        ~TeWdmLink(void);

        void         dump(std::string &  prefix,
                          bool           recursive);

        bool         setData(const telink_lscwdm_data_t &   data);
        bool         getData(telink_lscwdm_data_t &         data);

        bool   setLscWdmAvailBw(const wdm_link_lambdas_bitmap_t & bm);
        bool   getLscWdmAvailBw(wdm_link_lambdas_bitmap_t &     bm);

        bool         fitInConstraints(const cspf_constr_t & data);

    private:
        uint32_t                        dispersionPMD_;
        uint32_t                        spanLength_;
        std::list<wdm_amplifier_data_t> amplifiers_;
        wdm_link_lambdas_bitmap_t       lambdasBitmap_;
    };
```

Code 10-17: LSC WDM TE Links

10.4 G².PCE-RA internal API

10.4.1 Topology update in G².PCE-RA

The dynamic topology update process is generally managed by the routing protocol (i.e. OSPF-TE) through the IPC, but also the VTY interface can inject topology elements for debugging purposes. Focusing on the OSPF case, the G².PCE-RA update can be triggered:

- upon the arrival of a new LSA;

- when generating/re-generating an LSA.

The upload process is basically based on the filling up of G^2.PCE-RA external data structures, depending the type of the information contained in the LSA. These structures are then mangled by the IPC topology servant and translated in internal types of the G^2.PCE-RA process, to be used in the internal topology API of the module.

The list of the topology related APIs is provided in the following.

10.4.1.1 *Topology related*

```
pceraErrorCode_t
topologyGet(topology_summary_data_t &                  data,
          std::string &                          resp);
```

10.4.1.2 *Node generic*

```
pceraErrorCode_t
nodeAdd(const node_ident_t&                            id,
      std::string &                              resp);


pceraErrorCode_t
nodeDel(const node_ident_t&                            id,
      std::string &                              resp);


std::list<node_ident_t>
nodeGetAll(std::string &                         resp);
```

10.4.1.3 *Network Node related*

```
pceraErrorCode_t
netNodeUpdate(uint32_t                                 rId,
            const net_node_data_t &              data,
            std::string &                      resp);


pceraErrorCode_t
netNodeGet(uint32_t                              rId,
         net_node_data_t &                           data,
         std::string &                         resp);
```

10.4.1.4 *Grid Node related*

```
pceraErrorCode_t
gridSiteUpdate(uint32_t                              siteId,
           const grid_site_data_t&                   data,
           std::string &                             resp);


pceraErrorCode_t
gridSiteGet(uint32_t                                 siteId,
        grid_site_data_t &                           data,
        grid_subnodes_t &                            snodes,
        std::string &                          resp);



pceraErrorCode_t
gridSubNodeDel(uint32_t                              siteId,
           uint32_t                                  id,
           std::string &                             resp);


pceraErrorCode_t
gridServiceUpdate(uint32_t                     siteId,
            uint32_t                           id,
            const grid_service_data_t &              data,
            std::string &                            resp);


pceraErrorCode_t
gridServiceGet(uint32_t                              siteId,
           uint32_t                                  id,
           grid_service_data_t &                     data,
           std::string &                              resp);


pceraErrorCode_t
gridCompElemUpdate(uint32_t                          siteId,
             uint32_t                          id,
             const grid_ce_data_t &            data,
             std::string &                     resp);


pceraErrorCode_t
gridCompElemGet(uint32_t                       siteId,
          uint32_t                             id,
          grid_ce_data_t &                     data,
          std::string &                        resp);


pceraErrorCode_t
gridSubClusterUpdate(uint32_t                        siteId,
               uint32_t                        id,
               const grid_subcluster_data_t &        data,
               std::string &                   resp);
```

```
pceraErrorCode_t
gridSubClusterGet(uint32_t                    siteId,
               uint32_t                       id,
               grid_subcluster_data_t &       data,
               std::string &                        resp);


pceraErrorCode_t
gridStorageElemUpdate(uint32_t                      siteId,
                   uint32_t                         id,
                   const grid_se_data_t &           data,
                   std::string &               resp);


pceraErrorCode_t
gridStorageElemGet(uint32_t                         siteId,
                uint32_t                       id,
                grid_se_data_t &               data,
                std::string &                  resp);
```

10.4.1.5 *TNA related*

```
pceraErrorCode_t
tnaAdd(const uint32_t &                             rId,
       const g2mpls_addr_t &                        id,
       std::string &                                resp);


pceraErrorCode_t
tnaDel(const uint32_t &                             rId,
       const g2mpls_addr_t &                        id,
       std::string &                                resp);


std::list<g2mpls_addr_t>
tnaGetAllFromNode(const uint32_t &                  rId,
               std::string &                        resp);
```

10.4.1.6 *TE-Link related*

```
pceraErrorCode_t
linkAdd(const telink_ident_t &                      id,
      std::string &                            resp);


pceraErrorCode_t
linkDel(const telink_ident_t &                      id,
      std::string &                            resp);


std::list<telink_ident_t>
teLinkGetAllFromNode(const uint32_t &               rId,
```

```
                          std::string &                resp);
```

```
        pceraErrorCode_t
        teLinkUpdateCom(const telink_ident_t &                id,
                     const telink_com_data_t &         data,
                     std::string &                     resp);


        pceraErrorCode_t
        teLinkGetCom(const telink_ident_t &                   id,
                    telink_com_data_t &                data,
                    std::string &                      resp);
```

```
        pceraErrorCode_t
        teLinkUpdateTdm(const telink_ident_t &                id,
                     const telink_tdm_data_t &         data,
                     std::string &                     resp);


        pceraErrorCode_t
        teLinkGetTdm(const telink_ident_t &                   id,
                    telink_tdm_data_t &                data,
                    std::string &                      resp);
```

```
        pceraErrorCode_t
        teLinkUpdateLscG709(const telink_ident_t &            id,
                         const telink_lscg709_data_t &     data,
                         std::string &                 resp);


        pceraErrorCode_t
        teLinkGetLscG709(const telink_ident_t &               id,
                       telink_lscg709_data_t &         data,
                       std::string &                          resp);
```

```
        pceraErrorCode_t
        teLinkUpdateLscWdm(const telink_ident_t &             id,
                        const telink_lscwdm_data_t &          data,
                        std::string &                  resp);


        pceraErrorCode_t
        teLinkGetLscWdm(const telink_ident_t &                id,
                     telink_lscwdm_data_t &                   data,
                     std::string &                     resp);
```

```
        pceraErrorCode_t
```

```
teLinkUpdateStates(const telink_ident_t&            id,
                const opstate_t &                   opState,
                const admstate_t &                  admState,
                std::string &               resp);


pceraErrorCode_t
teLinkGetStates(const telink_ident_t&               id,
             opstate_t &                    opState,
             admstate_t &                   admState,
             std::string &                  resp);
```

```
pceraErrorCode_t
teLinkUpdateGenBw(const telink_ident_t &            id,
               const avail_bw_per_prio_t &          bw,
               std::string &                        resp);


pceraErrorCode_t
teLinkGetGenBw(const telink_ident_t &               id,
              avail_bw_per_prio_t &                 bw,
              std::string &                         resp);
```

```
pceraErrorCode_t
teLinkUpdateTdmBw(const telink_ident_t &            id,
               const avail_bw_per_prio_t &          bw,
               const std::list<uint32_t>            freeTS,
               std::string &                        resp);


pceraErrorCode_t
teLinkGetTdmBw(const telink_ident_t &               id,
              avail_bw_per_prio_t &                 bw,
              std::list<uint32_t>           freeTS,
              std::string &                         resp);
```

```
pceraErrorCode_t
teLinkUpdateLscG709Bw(const telink_ident_t &        id,
                   const avail_bw_per_prio_t &      bw,
                   const std::list<uint32_t>        freeODUk,
                   const std::list<uint32_t>        freeOCh,
                   std::string &            resp);
pceraErrorCode_t
teLinkGetLscG709Bw(const telink_ident_t &           id,
                avail_bw_per_prio_t &       bw,
                std::list<uint32_t>                 freeODUk,
                std::list<uint32_t>                 freeOCh,
                std::string &               resp);
```

```
        pceraErrorCode_t
        teLinkUpdateLscWdmBw(const telink_ident_t &         id,
                          const avail_bw_per_prio_t & bw,
                          wdm_link_lambdas_bitmap_t & bm,
                          std::string &               resp);


        pceraErrorCode_t
        teLinkGetLscWdmBw(const telink_ident_t &            id,
                       avail_bw_per_prio_t &               bw,
                       wdm_link_lambdas_bitmap_t &         bm,
                       std::string &                       resp);
```

```
        pceraErrorCode_t
        teLinkAppendSrlgs(const telink_ident_t &            id,
                       const std::list<uint32_t> &         srlgs,
                       std::string &                       resp);


        pceraErrorCode_t
        teLinkGetSrlgs(const telink_ident_t &               ident,
                     std::list<uint32_t> &                 srlgs,
                     std::string &                         resp);
```

```
        pceraErrorCode_t
        teLinkAppendCalendar(const telink_ident_t &                  id,
                          const std::map<uint32_t,avail_bw_per_prio_t> cal,
                          std::string &                      resp);


        pceraErrorCode_t
        teLinkGetCalendar(const telink_ident_t &                  id,
                       std::map<uint32_t,avail_bw_per_prio_t>      cal,
                       std::string &                             resp);
```

```
        pceraErrorCode_t
        teLinkAppendIsc(const telink_ident_t &                  id,
                     const std::list<isc_t> &              iscs,
                     std::string &                         resp);


        pceraErrorCode_t
        teLinkGetIsc(const telink_ident_t &                     id,
                   std::list<isc_t> &                          iscs,
                   std::string &                           resp);
};
```

10.4.2 Computation of routes in G^2.PCE-RA

Path computation is the main task of the G^2.PCE-RA module, triggered by G^2.PCE-RA users in different cases:

- by NCC, for the computation of the route of a call and then the primary and/or secondary LSPs in it;
- by G.RSVP-TE, for the ERO completion and the crankback management;
- by G^2.PCE-RA VTY, for testing purposes.

All the SPF computations are provided by an implementation of the Dijkstra constrained algorithm, described in terms of pseudo-code in Figure 10-5.

```
algorithm Constrained Dijkstra

define V = set of vertices in the given graph
define U = set of unvisited vertices in the given graph
define Γ_I = set of neighbor vertices of vertex I
define P(I) = predecessor of vertex I along the path
define c_IJ = cost of the arc from vertex I to vertex J
define d(I) = cumulative path cost from root vertex S till vertex I
define S/D = source/destination vertex
define l_IJ = arc between vertex I and J
define CONSTR = set of constraints the SPF must satisfy

begin
  step 1. d(S)=0;
          if (I ∈ Γ_S and (l_IJ,CONSTR)=TRUE) then d(I)= c_SI else d(I)= ∞;
          U = V-{S};
          P(I)= S   ∀ I ∈ U;

  step 2. search J ∈ U:(l_P(J)J,CONSTR)= TRUE and d(J)= min d(k),  ∀ k ∈ U ;
          U = U-{J};
          if J = D    then END

  step 3. ∀ (I ∈ Γ_J and I ∈ U) do
         if d(J)+ c_JI < d(I) then {d(I)=d(J)+ c_JI and P(I)=J};
         goto step 2
end
```

Figure 10-5: G^2.PCE-RA constrained Dijkstra pseudo-code.

The list of the topology related APIs is provided in the following.

```
pceraErrorCode_t
nodeGetFromTna(const  net_res_spec_t &                          tnaRes,
               uint32_t     &                                   rId,
               std::string &                                    resp);


pceraErrorCode_t
nodeGetFromGnsTna(const grid_res_spec_t &     tnaGnsRes,
                  const std::list<uint32_t>&     excludeSet,
                  uint32_t    &            netNodeId,
                  uint32_t    &            gridSiteId,
                  std::string &                resp);


pceraErrorCode_t
callRoute(const ero_hop_t &                  srcHop,
          const ero_hop_t &            dstHop,
          const call_ident_t &               callId,
          const call_info_t &                callInfo,
          const recovery_info_t &      recInfo,
          const lsp_info_t &                 lspInfo,
          std::list<ero_hop_t> &       wEro,
          std::string &                      resp);


pceraErrorCode_t
callFlush(const call_ident_t &               callId,
          std::string &                      resp);


pceraErrorCode_t
callConfirm(const call_ident_t &       callId,
            std::string &              resp);


pceraErrorCode_t
lspRoute(const ero_hop_t &             srcHop,
         const ero_hop_t &             dstHop,
         const call_ident_t &                callId,
         const call_info_t &                 callInfo,
         const recovery_info_t &       recInfo,
         const lsp_info_t &            lspInfo,
         const std::list<ero_hop_t> &        excludeEro,
         std::list<ero_hop_t> &              wEro,
         std::list<ero_hop_t> &              pEro,
         std::string &                       resp);
```

Code 10-18: Topology-related APIs

When a callRoute()with the request for computing two disjoint routes in the topology between the ingress and egress TNAs occurs, the G^2.PCE-RA provides two computational strategies:

- the Two Step Algorithm (TSA), applied in case of no strict requirement on the disjointness of the produced pair of routes, if any;
- the Bhandari's algorithm, applied in case of maximally disjoint routes computations.

In both cases (i.e. TSA, Bhandari), the SPF computation (i.e. Dijkstra) is carried out after a specific topology transformation which modifies link metrics. After the computation, topology is reverted to the original state in order to process subsequent computation requests on a reliable topology representation.

The following figures summarize the relevant function call flow in the G^2.PCE-RA code by means of flow diagrams.

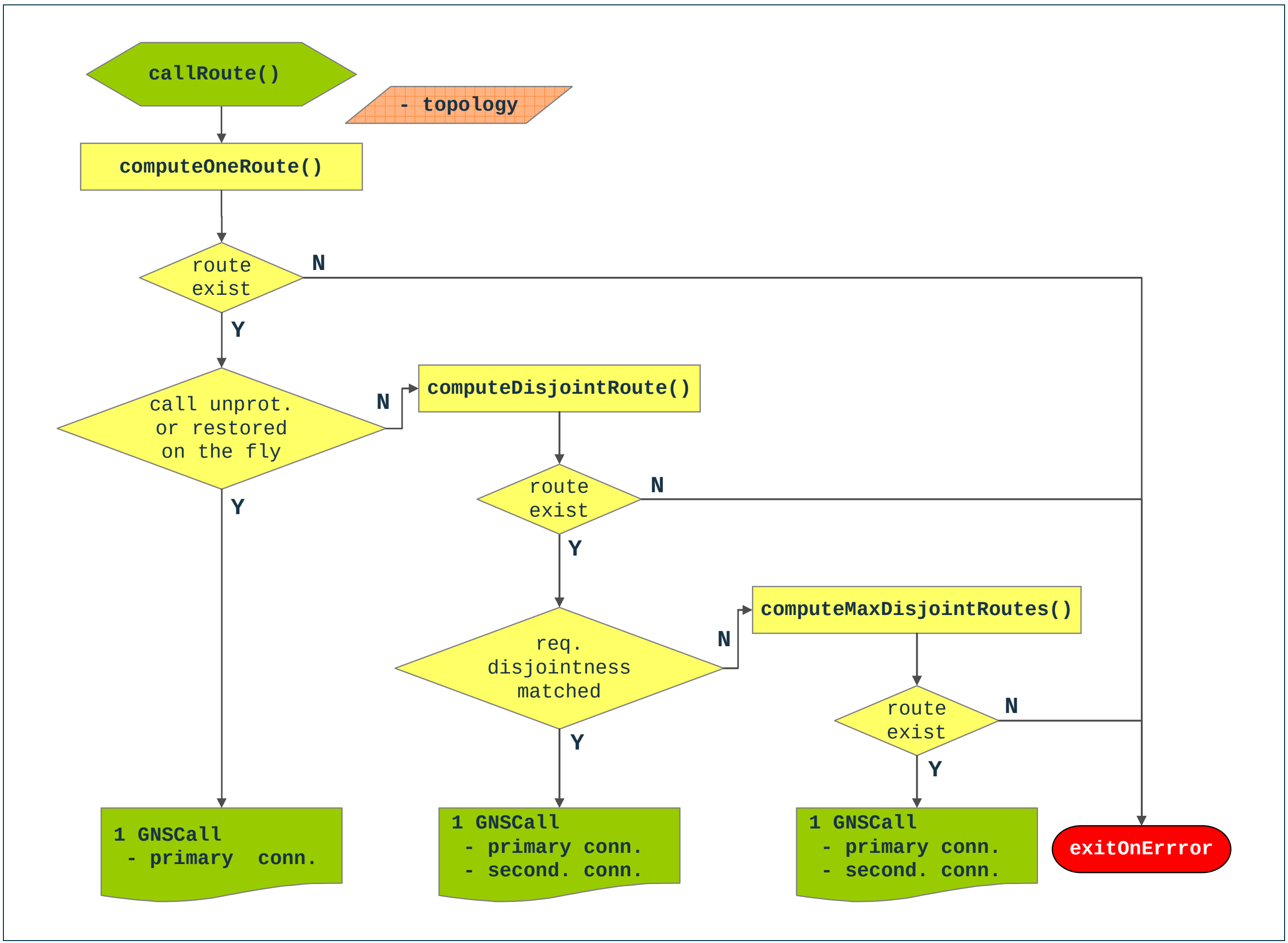

Figure 10-6: Actions on a callRoute().

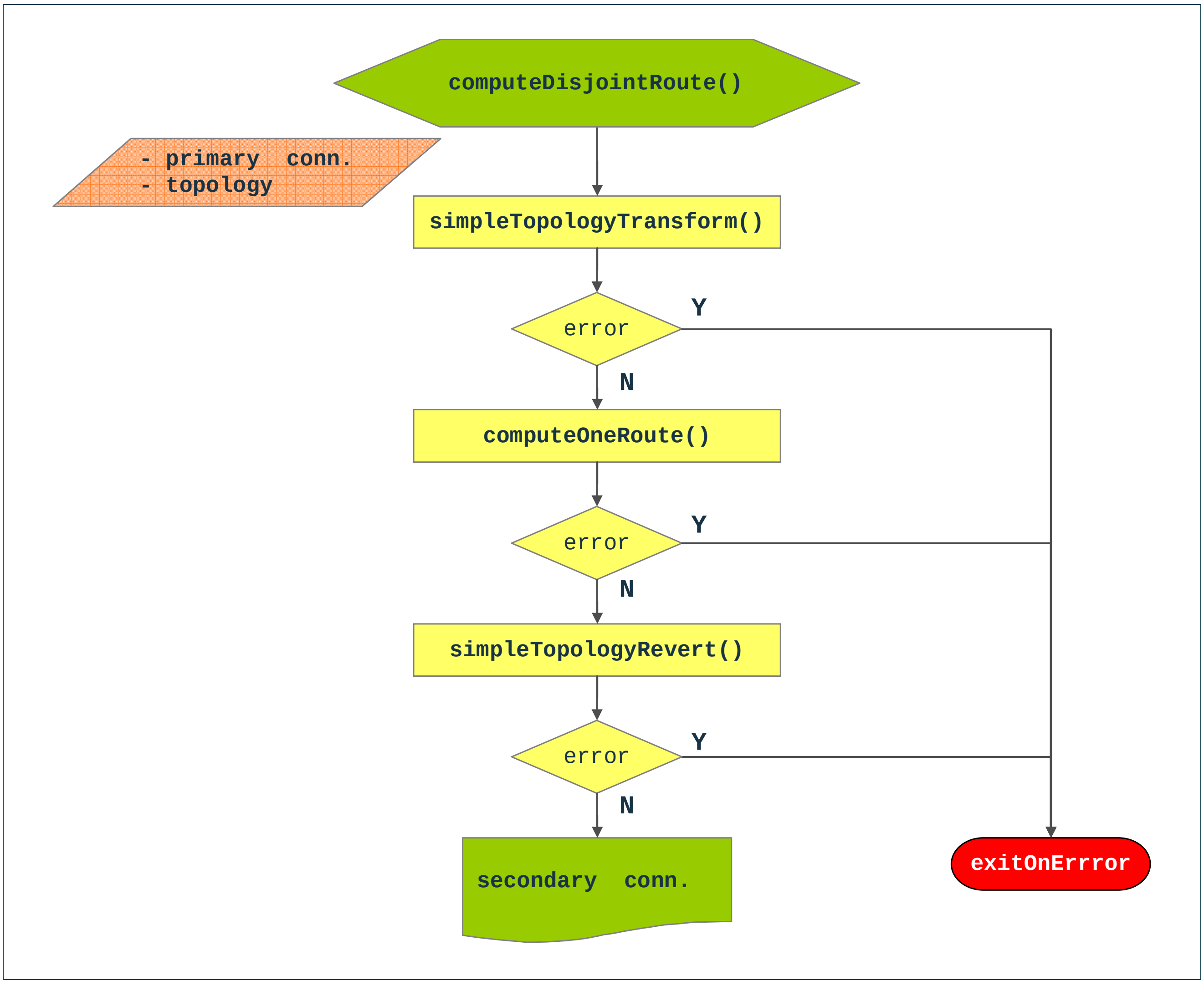

Figure 10-7: Actions on a computeDisjointRoute().

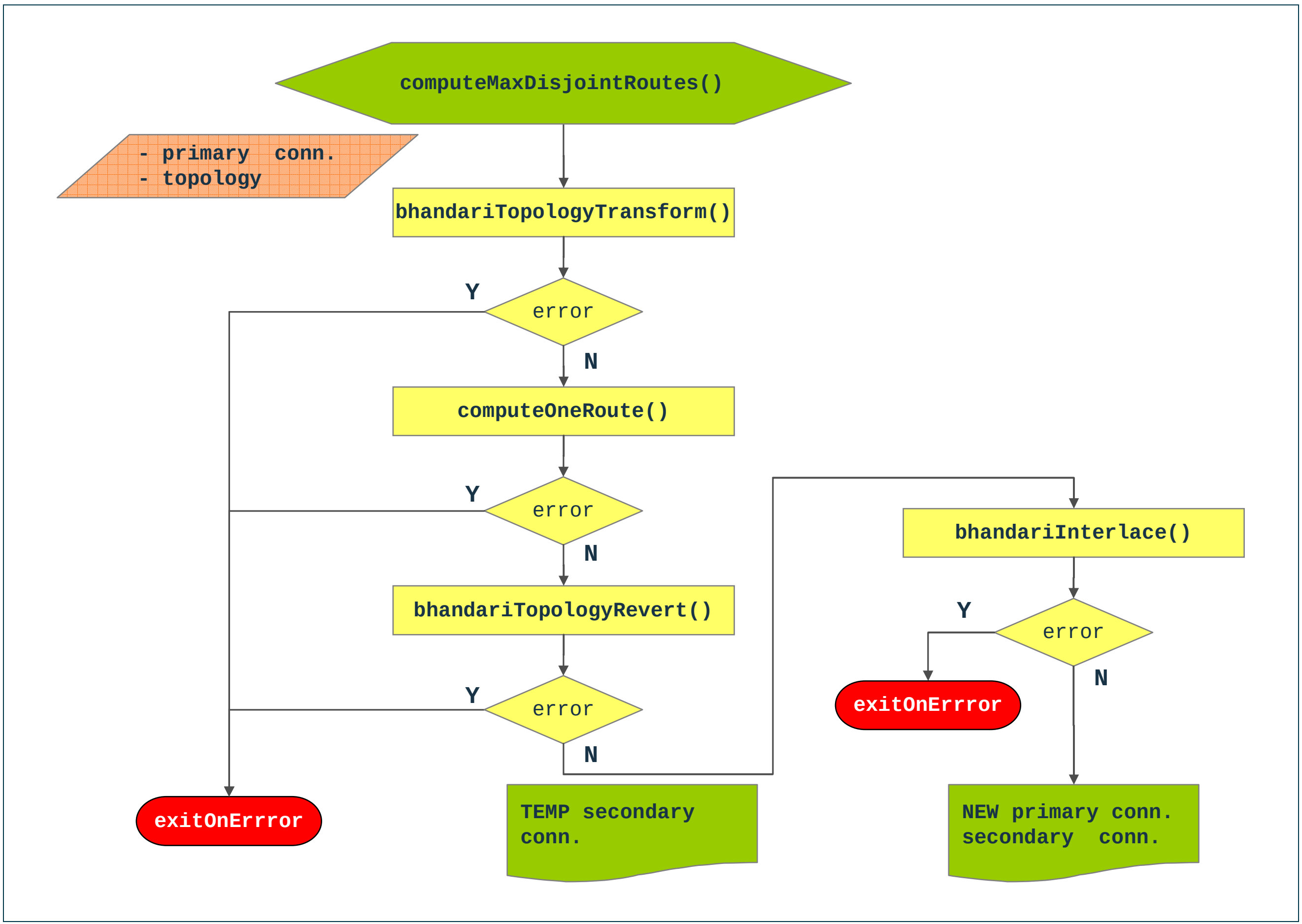

Figure 10-8: Actions on a computeMaxDisjointRoutes().

10.5 G^2.PCE-RA external API

10.5.1 Topology API

The G^2.PCE-RA module exposes an external topology interface by means of CORBA servants. The API for the communication with external modules is specified in the *<sw_root>/idl/g2mplsTopology.idl* and shown below. It is strictly related to the semantic of the internal G^2.PCE-RA API for topology updates.

Common types used in this interface are specified in <sw_root>/idl/g2mplsTypes.idl and reported in Appendix A.

```
#include "types.idl"
#include "g2mplsTypes.idl"
```

```
#ifndef G2MPLSTOPOLOGY_IDL
#define G2MPLSTOPOLOGY_IDL

interface g2mplsTopology {

	exception InternalProblems {
		string					what;
	};

	exception CannotFetchNode {
		g2mplsTypes::nodeId			id;
		string					what;
	};

	exception CannotFetchSubNode {
		g2mplsTypes::nodeId			parentId;
		g2mplsTypes::gridSubNodeId		id;
		string					what;
	};

	exception CannotFetchLink {
		g2mplsTypes::TELinkId			id;
		string					what;
	};

	exception CannotFetchTna {
		g2mplsTypes::tnaId			id;
		string					what;
	};



	//
	//  Topology related calls
	//
	boolean
	nodeAdd(in		g2mplsTypes::nodeIdent		id)
		raises(InternalProblems);

	boolean
	nodeDel(in		g2mplsTypes::nodeIdent		id)
		raises(InternalProblems, CannotFetchNode);

	g2mplsTypes::nodeIdentSeq
	nodeGetAll()
		raises(InternalProblems);

	boolean
	netNodeUpdate(in	g2mplsTypes::nodeId		id,
		      in	g2mplsTypes::netNodeParams	info)
		raises(InternalProblems, CannotFetchNode);

	boolean
	netNodeGet(in			g2mplsTypes::nodeId	id,
		   out		g2mplsTypes::netNodeParams	info)
		raises(InternalProblems, CannotFetchNode);
```

```
// Grid related elements (from GLUE)
boolean
gridSiteUpdate(in   g2mplsTypes::nodeId                    id,
               in    g2mplsTypes::gridSiteParams           info)
      raises(InternalProblems, CannotFetchNode);

boolean
gridSiteGet(in             g2mplsTypes::nodeId             id,
          out              g2mplsTypes::gridSiteParams     info,
          out              g2mplsTypes::gridSubNodes       snodes)
      raises(InternalProblems, CannotFetchNode);

boolean
gridSubNodeDel(in   g2mplsTypes::nodeId                    siteId,
               in    g2mplsTypes::gridSubNodeIdent         id)
      raises(InternalProblems,
             CannotFetchNode,
             CannotFetchSubNode);

boolean
gridServiceUpdate(in      g2mplsTypes::nodeId              siteId,
                  in   g2mplsTypes::gridSubNodeId          id,
                  in   g2mplsTypes::gridServiceParams      info)
      raises(InternalProblems,
             CannotFetchNode,
             CannotFetchSubNode);

boolean
gridServiceGet(in   g2mplsTypes::nodeId                    siteId,
             in     g2mplsTypes::gridSubNodeId             id,
             out    g2mplsTypes::gridServiceParams         info)
      raises(InternalProblems,
             CannotFetchNode,
             CannotFetchSubNode);

boolean
gridCompElemUpdate(in     g2mplsTypes::nodeId              siteId,
                   in  g2mplsTypes::gridSubNodeId          id,
                   in  g2mplsTypes::gridCEParams           info)
      raises(InternalProblems,
             CannotFetchNode,
             CannotFetchSubNode);

boolean
gridCompElemGet(in  g2mplsTypes::nodeId                    siteId,
             in     g2mplsTypes::gridSubNodeId             id,
             out    g2mplsTypes::gridCEParams              info)
      raises(InternalProblems,
             CannotFetchNode,
             CannotFetchSubNode);

boolean
gridSubClusterUpdate(in g2mplsTypes::nodeId                siteId,
                     in g2mplsTypes::gridSubNodeId         id,
                     in g2mplsTypes::gridSubClusterParams  info)
      raises(InternalProblems,
             CannotFetchNode,
```

```
                 CannotFetchSubNode);

        boolean
        gridSubClusterGet(in  g2mplsTypes::nodeId           siteId,
                          in  g2mplsTypes::gridSubNodeId id,
                          out g2mplsTypes::gridSubClusterParams info)
                raises(InternalProblems,
                       CannotFetchNode,
                       CannotFetchSubNode);

        boolean
        gridStorageElemUpdate(in g2mplsTypes::nodeId        siteId,
                              in g2mplsTypes::gridSubNodeId     id,
                              in g2mplsTypes::gridSEParams      info)
                raises(InternalProblems,
                       CannotFetchNode,
                       CannotFetchSubNode);

        boolean
        gridStorageElemGet(in   g2mplsTypes::nodeId         siteId,
                           in   g2mplsTypes::gridSubNodeId      id,
                           out  g2mplsTypes::gridSEParams       info)
                raises(InternalProblems,
                       CannotFetchNode,
                       CannotFetchSubNode);

        // TNA related
        boolean
        tnaIdsAdd(in        g2mplsTypes::nodeIdent            ident,
                  in        g2mplsTypes::tnaIdSeq             seq)
                raises(InternalProblems,
                       CannotFetchNode);

        boolean
        tnaIdsDel(in        g2mplsTypes::nodeIdent            ident,
                  in        g2mplsTypes::tnaIdSeq             seq)
                raises(InternalProblems,
                       CannotFetchNode,
                       CannotFetchTna);

        g2mplsTypes::tnaIdSeq
        tnaIdsGetAllFromNode(in   g2mplsTypes::nodeIdent    ident)
                raises(InternalProblems,
                       CannotFetchNode);

        // Link related
        boolean
        linkAdd(in          g2mplsTypes::teLinkIdent          ident)
                raises(InternalProblems);

        boolean
        linkDel(in          g2mplsTypes::teLinkIdent          ident)
                raises(InternalProblems, CannotFetchLink);

        g2mplsTypes::teLinkIdentSeq
        teLinkGetAllFromNode(in   g2mplsTypes::nodeIdent    ident)
                raises(InternalProblems);
```

```
        // link capabilities
        boolean
        teLinkUpdateCom(in  g2mplsTypes::teLinkIdent         ident,
                    in     g2mplsTypes::teLinkComParams     info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetCom(in           g2mplsTypes::teLinkIdent  ident,
                   out     g2mplsTypes::teLinkComParams     info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateTdm(in  g2mplsTypes::teLinkIdent         ident,
                    in     g2mplsTypes::teLinkTdmParams     info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetTdm(in           g2mplsTypes::teLinkIdent  ident,
                   out     g2mplsTypes::teLinkTdmParams     info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateLscG709(in g2mplsTypes::teLinkIdent            ident,
                        in g2mplsTypes::teLinkLscG709Params       info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetLscG709(in       g2mplsTypes::teLinkIdent         ident,
                      out   g2mplsTypes::teLinkLscG709Params        info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateLscWdm(in   g2mplsTypes::teLinkIdent           ident,
                       in   g2mplsTypes::teLinkLscWdmParams       info)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetLscWdm(in  g2mplsTypes::teLinkIdent         ident,
                    out    g2mplsTypes::teLinkLscWdmParams  info)
              raises(InternalProblems, CannotFetchLink);

        // link states
        boolean
        teLinkUpdateStates(in     g2mplsTypes::teLinkIdent  ident,
                       in  g2mplsTypes::statesBundle        states)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetStates(in  g2mplsTypes::teLinkIdent         ident,
                    out    g2mplsTypes::statesBundle        states)
              raises(InternalProblems, CannotFetchLink);

        // link bandwidth
        boolean
        teLinkUpdateGenBw(in      g2mplsTypes::teLinkIdent  ident,
                      in  g2mplsTypes::availBwPerPrio      bw)
              raises(InternalProblems, CannotFetchLink);
```

```
        boolean
        teLinkGetGenBw(in   g2mplsTypes::teLinkIdent        ident,
                     out    g2mplsTypes::availBwPerPrio     bw)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateTdmBw(in      g2mplsTypes::teLinkIdent  ident,
                       in   g2mplsTypes::availBwPerPrio     bw,
                       in   g2mplsTypes::freeCTPSeq         freeTS)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetTdmBw(in   g2mplsTypes::teLinkIdent        ident,
                     out    g2mplsTypes::availBwPerPrio     bw,
                     out    g2mplsTypes::freeCTPSeq         freeTS)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateLscG709Bw(in g2mplsTypes::teLinkIdent   ident,
                           in g2mplsTypes::availBwPerPrio   bw,
                           in g2mplsTypes::freeCTPSeq       freeODUk,
                           in g2mplsTypes::freeCTPSeq       freeOCh)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetLscG709Bw(in     g2mplsTypes::teLinkIdent  ident,
                        out g2mplsTypes::availBwPerPrio     bw,
                        out g2mplsTypes::freeCTPSeq         freeODUk,
                        out g2mplsTypes::freeCTPSeq         freeOCh)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkUpdateLscWdmBw(in   g2mplsTypes::teLinkIdent              ident,
                          in      g2mplsTypes::availBwPerPrio           bw,
                          in      g2mplsTypes::teLinkWdmLambdasBitmap   bm)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetLscWdmBw(in      g2mplsTypes::teLinkIdent          ident,
                       out  g2mplsTypes::availBwPerPrio             bw,
                       out  g2mplsTypes::teLinkWdmLambdasBitmap     bm)
              raises(InternalProblems, CannotFetchLink);

        // append operations
        boolean
        teLinkAppendSrlgs(in      g2mplsTypes::teLinkIdent          ident,
                       in   g2mplsTypes::srlgSeq                    srlgs)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetSrlgs(in   g2mplsTypes::teLinkIdent                ident,
                     out    g2mplsTypes::srlgSeq                    srlgs)
              raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkAppendCalendar(in   g2mplsTypes::teLinkIdent          ident,
                          in      g2mplsTypes::teLinkCalendarSeq    cal)
              raises(InternalProblems, CannotFetchLink);
```

```
        boolean
        teLinkGetCalendar(in      g2mplsTypes::teLinkIdent         ident,
                    out  g2mplsTypes::teLinkCalendarSeq         cal)
                raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkAppendIsc(in  g2mplsTypes::teLinkIdent                ident,
                  in     g2mplsTypes::iscSeq                        iscs)
                raises(InternalProblems, CannotFetchLink);

        boolean
        teLinkGetIsc(in            g2mplsTypes::teLinkIdent         ident,
                 out     g2mplsTypes::iscSeq                        iscs)
                raises(InternalProblems, CannotFetchLink);
};
#endif // G2MPLSTOPOLOGY_IDL
```

Code 10-19: G².PCE-RA Topology external API IDL.

10.5.2 Computation API

The G².PCE-RA module exposes an external call/LSP interface by means of CORBA servants. The API for the communication with external modules is specified in the *<sw_root>/idl/g2pcera.idl* and shown below. It is strictly related to the semantic of the internal G².PCE-RA API for route computations.

Common types used in this interface are specified in <sw_root>/idl/g2mplsTypes.idl and reported in Appendix A.

```
#include "types.idl"
#include "g2mplsTypes.idl"

#ifndef G2PCERA_IDL
#define G2PCERA_IDL

interface G2PCERA {

        exception InternalProblems {
                string                                  what;
        };

        exception CannotFetchNode {
                g2mplsTypes::nodeId                     id;
                string                                  what;
        };

        exception CannotFetchSubNode {
                g2mplsTypes::nodeId                     parentId;
                g2mplsTypes::gridSubNodeId              id;
                string                                  what;
        };
```

```
        exception CannotFetchLink {
               g2mplsTypes::TELinkId                               id;
               string                                       what;
        };

        exception CannotFetchTna {
               g2mplsTypes::tnaId                           id;
               string                                       what;
        };






        //
        // Computation related calls
        //

        boolean
        nodeGetFromTna(in   g2mplsTypes::tnaResource         tnaRes,
                      out   g2mplsTypes::nodeId              node)
               raises(InternalProblems,
                      CannotFetchNode,
                      CannotFetchSubNode,
                      CannotFetchLink,
                      CannotFetchTna);

        boolean
        nodeGetFromGnsTna(in      g2mplsTypes::gridParams   tnaGnsRes,
                         in   g2mplsTypes::nodeIdentSeq       excludeSet,
                         out  g2mplsTypes::nodeId             netNodeId,
                         out  g2mplsTypes::nodeId             gridSiteId)
               raises(InternalProblems,
                      CannotFetchNode,
                      CannotFetchSubNode,
                      CannotFetchLink,
                      CannotFetchTna);

        boolean
        callRoute(in g2mplsTypes::eroItem             srcHop,
                 in   g2mplsTypes::eroItem            dstHop,
                 in   g2mplsTypes::gridParams         eTnaGnsRes,
                 in   g2mplsTypes::callIdent          callId,
                 in   g2mplsTypes::callParams         callInfo,
                 in   g2mplsTypes::recoveryParams     recInfo,
                 in   g2mplsTypes::lspParams          lspInfo,
                 out g2mplsTypes::eroSeq              wEro)
               raises(InternalProblems,
                      CannotFetchNode,
                      CannotFetchSubNode,
                      CannotFetchLink,
                      CannotFetchTna);

        boolean
        callFlush(in g2mplsTypes::callIdent          callId)
               raises(InternalProblems);
```

```
       boolean
       callConfirm(in      g2mplsTypes::callIdent    callId)
              raises(InternalProblems);

       boolean
       lspRoute(in  g2mplsTypes::eroItem             srcHop,
               in   g2mplsTypes::eroItem             dstHop,
               in   g2mplsTypes::callIdent           callId,
               in   g2mplsTypes::callParams          callInfo,
               in   g2mplsTypes::recoveryParams      recInfo,
               in   g2mplsTypes::lspParams           lspInfo,
               in   g2mplsTypes::eroSeq              excludeEro,
               out  g2mplsTypes::eroSeq              wEro,
               out  g2mplsTypes::eroSeq              pEro)
              raises(InternalProblems,
                     CannotFetchNode,
                     CannotFetchSubNode,
                     CannotFetchLink);
};
#endif // G2PCERA_IDL
```

Code 10-20: G².PCE-RA Computation external API IDL.

11 G.UNI-GW Adapter Design Specification

The main functionality of the G.UNI-GW Adapter is to map signalling and routing information from the WSAG Server to G^2.RSVP and G^2.OSPF protocol controllers. On one side, the G.UNI-GW implements a Web Service that accepts incoming messages from the WSAG Server. On the other, these requests are translated into CORBA IDL calls to control the client Call Controller on the UNI-C side.

11.1 G.UNI-GW Adapter Transactions

The transactions mapped by the G.UNI-GW involve GNS requests and Grid information updates. Figure 11-1 depicts G.UNI-GW adapter design, showing the involved interfaces and transactions. Communications between G.UNI-GW adapter and the rest of the modules is bidirectional, so depending on the situation (local or remote), the information will flow in one way (WSAG Server – G.UNIGW adapter – Call Controller) or the other (Call Controller – G.UNIGW adapter – WSAG Server).

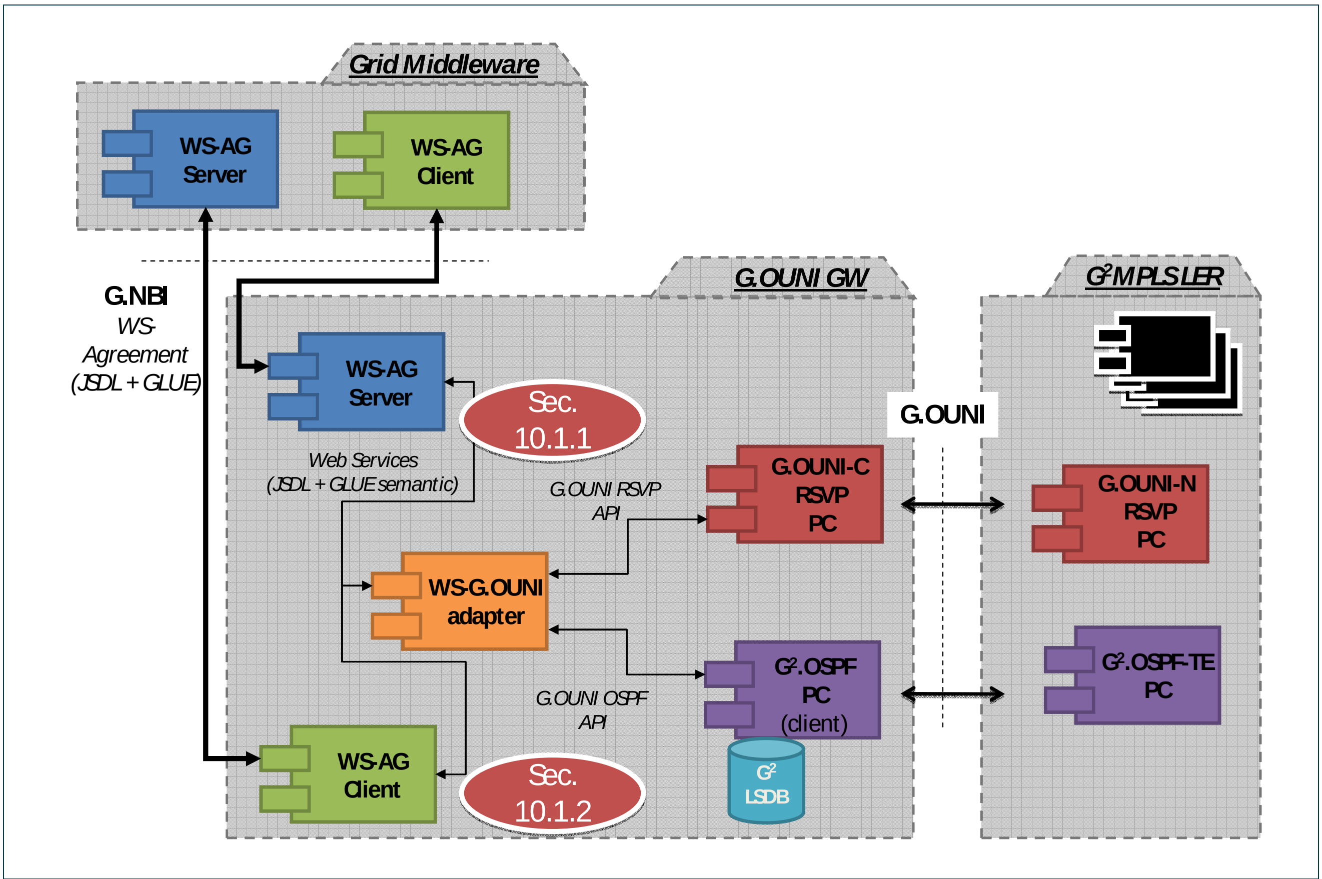

Figure 11-1: The GUNI-GW breakdown and transactions localization.

11.1.1 WSAG – WS-G.UNI Adapter – G.UNI-C RSVP PC (Signalling)

Three methods implement the signalling transactions that enable the creation and deletion of GNS: CreateActivity, GetActivityStatuses, TerminateActivities.

CreateActivity

```
CreateActivity(CreateActivityType *, CreateActivityResponse *)
```

The CreateAcitivty method is used to request a Grid Network Service.

- Incoming parameters: CreateActivityType → Contains Grid and Network information required to provision a GNS.

- Response parameters: CreateActivityResponseType → Contains a unique EndPointreference (EPR) which identifies a certain activity. Usually, the address parameter of an EPR contains the URI of the V-site that created the activity. This is required since the stage out process is initiated later on by the MSS to simplify the workflow for the network scheduler.
- CORBA IDL Mapping:
  - *callCreate*
  - *callSetTna*
  - *callSetGnsTna (ingress)*
  - *callSetGnsTna (egress)*
  - *callEnable*
  - *callSetup*
- WSDL description:

```
<!-- Message Types -->
<xsd:complexType name="CreateActivityType">
  <xsd:sequence>
    <xsd:element ref="bes-factory:ActivityDocument"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
     maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="CreateActivityResponseType">
  <xsd:sequence>
    <xsd:element name="ActivityIdentifier" type="wsa:EndpointReferenceType"/>
    <xsd:element ref="bes-factory:ActivityDocument" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
     maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<!-- Message Elements -->
<xsd:element name="CreateActivity"
     type="bes-factory:CreateActivityType"/>
<xsd:element name="CreateActivityResponse"
     type="bes-factory:CreateActivityResponseType"/>

<!-- Messages -->
<wsdl:message name="CreateActivityRequest">
  <wsdl:part name="parameters" element="bes-factory:CreateActivity"/>
</wsdl:message>
<wsdl:message name="CreateActivityResponse">
  <wsdl:part name="parameters" element="bes-factory:CreateActivityResponse"/>
</wsdl:message>

<!-- Port Type -->
<wsdl:portType name="BESFactoryPortType">
  <wsdl:operation name="CreateActivity">
    <wsdl:input
      name="CreateActivity"
      message="bes-factory:CreateActivityRequest"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
```

```
      bes-factory/BESFactoryPortType/CreateActivity"/>
    <wsdl:output
      name="CreateActivityResponse"
      message="bes-factory:CreateActivityResponse"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
        bes-factory/BESFactoryPortType/CreateActivityResponse"/>
  </wsdl:operation>
</wsdl:portType>

<!-- Bindings -->
<wsdl:binding name="BESFactoryBinding" type="bes-factory:BESFactoryPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="CreateActivity">
    <soap:operation soapAction="http://schemas.ggf.org/bes/2006/08/
      bes-factory/BESFactoryPortType/CreateActivity"/>
    <wsdl:input name="CreateActivity">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="CreateActivityResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

GetActivityStatuses

```
GetActivityStatuses(GetActivityStatusesType *, GetActivityStatusesResponseType *)
```

The GetActivityStatuses method is used to request GNS status.

- Incoming parameters: GetActivityStatusType → Contains the GNS identifier of the activity to be check.
- Response parameters: GetActivityStatusResponseType → Contains the status of the requested GNS: Pending, Running, Cancelled, Failed or Finished.
- CORBA IDL Mapping:
  - *callGetDetails*
- WSDL description:

```
<!-- Message Types -->
<xsd:complexType name="GetActivityStatusesType">
  <xsd:sequence>
    <xsd:element name="ActivityIdentifier" type="wsa:EndpointReferenceType"
      maxOccurs="unbounded" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="GetActivityStatusesResponseType">
  <xsd:sequence>
    <xsd:element name="Response" type="bes-factory:GetActivityStatusResponseType"
```

```
      maxOccurs="unbounded" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<!-- Message Elements -->
<xsd:element name="GetActivityStatuses"
  type="bes-factory:GetActivityStatusesType"/>
<xsd:element name="GetActivityStatusesResponse"
  type="bes-factory:GetActivityStatusesResponseType"/>

<!-- Messages -->
<wsdl:message name="GetActivityStatusesRequest">
  <wsdl:part name="parameters" element="bes-factory:GetActivityStatuses"/>
</wsdl:message>

<wsdl:message name="GetActivityStatusesResponse">
  <wsdl:part name="parameters" element="bes-factory:GetActivityStatusesResponse"/>
</wsdl:message>

<!-- Port Type -->
<wsdl:portType name="BESFactoryPortType">
  <wsdl:operation name="GetActivityStatuses">
    <wsdl:input
      name="GetActivityStatuses"
      message="bes-factory:GetActivityStatusesRequest"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
        bes-factory/BESFactoryPortType/GetActivityStatuses"/>
    <wsdl:output
      name="GetActivityStatusesResponse"
      message="bes-factory:GetActivityStatusesResponse"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
        bes-factory/BESFactoryPortType/GetActivityStatusesResponse"/>
  </wsdl:operation>
</wsdl:portType>

<!-- Bindings -->
<wsdl:binding name="BESFactoryBinding" type="bes-factory:BESFactoryPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetActivityStatuses">
    <soap:operation soapAction="http://schemas.ggf.org/bes/2006/08/
      bes-factory/BESFactoryPortType/GetActivityStatuses"/>
    <wsdl:input name="GetActivityStatuses">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="GetActivityStatusesResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

TerminateActivities

```
TerminateActivities(TerminateActivitiesType *, TerminateActivitiesResponseType *)
```

The TerminateActivities method is used to terminate a Grid Network Service.

- Incoming parameters: GetActivityStatusType → Contains the GNS identifier of the activity to be terminated.
- Response parameters: GetActivityStatusResponseType → Contains the GNS identifier of the activity to be terminated and the acknowledgement of the termination state.
- CORBA IDL Mapping:
  - *callDisable*
  - *callDestroy*
- WSDL description:

```
<!-- Message Types -->
<xsd:complexType name="TerminateActivitiesType">
  <xsd:sequence>
    <xsd:element name="ActivityIdentifier" type="wsa:EndpointReferenceType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="TerminateActivitiesResponseType">
  <xsd:sequence>
    <xsd:element name="Response" type="bes-factory:TerminateActivityResponseType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<!-- Message Elements -->
<xsd:element name="TerminateActivities"
  type="bes-factory:TerminateActivitiesType"/>
<xsd:element name="TerminateActivitiesResponse"
  type="bes-factory:TerminateActivitiesResponseType"/>

<!-- Messages -->
<wsdl:message name="TerminateActivitiesRequest">
  <wsdl:part name="parameters" element="bes-factory:TerminateActivities"/>
</wsdl:message>

<wsdl:message name="TerminateActivitiesResponse">
  <wsdl:part name="parameters" element="bes-factory:TerminateActivitiesResponse"/>
</wsdl:message>

<!-- Port Type -->
<wsdl:portType name="BESFactoryPortType">
  <wsdl:operation name="TerminateActivities">
    <wsdl:input
      name="TerminateActivities"
      message="bes-factory:TerminateActivitiesRequest"
```

```
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
        bes-factory/BESFactoryPortType/TerminateActivities"/>
    <wsdl:output
      name="TerminateActivitiesResponse"
      message="bes-factory:TerminateActivitiesResponse"
      wsa:Action="http://schemas.ggf.org/bes/2006/08/
        bes-factory/BESFactoryPortType/TerminateActivitiesResponse"/>
  </wsdl:operation>
</wsdl:portType>

<!-- Bindings -->
<wsdl:binding name="BESFactoryBinding" type="bes-factory:BESFactoryPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="TerminateActivities">
    <soap:operation soapAction="http://schemas.ggf.org/bes/2006/08/
      bes-factory/BESFactoryPortType/TerminateActivities" />
    <wsdl:input name=" TerminateActivities">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name=" TerminateActivitiesResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

11.2 G.UNI-GW adapter Implementation

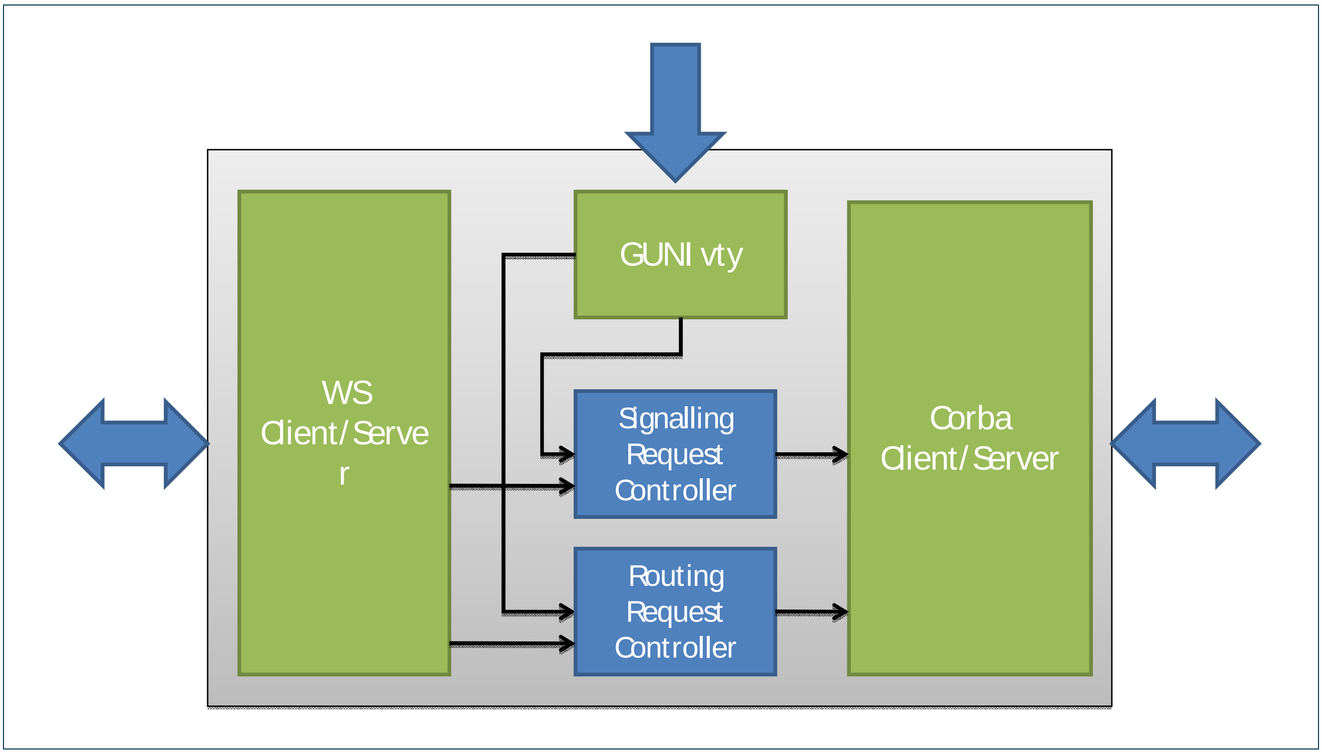

Figure 11-2: GUNI-GW operation flow.

The G.UNI-GW adapter can be divided in five basic functional blocks (Figure 11-2):

- WS Client/Server: This functional block implements the Web Service towards the WSAG Client/Server. The binding structures and stubs have been implemented using the open source gSoap 2.7.10 wsdl compiler.
- Corba Client/Server: This functional block calls the client Call Controller methods. The implementation uses the open source omniORB-4.1.2, which is a CORBA Object Request Broker (ORB) for C++.
- GUNI vty: This functional block implements the virtual terminal interface commands to manage the G.UNI-GW adapter.
- Signalling Request Controller: This block translates WS GNS requests into CORBA IDL calls.
- Routing Request Controller: This block translates WS Grid update information into CORBA IDL calls.

11.2.1 File descriptions

- *gunigw_main.cxx*: Main GUNIGW process file.
  - Initializes vty
  - Starts WS Server
- gunigwd.cxx: Class GUNIGW_Master Implementation file.
- *gunigw_vty*: VTY commands file.
- *BESFactoryBindingServer.cxx*: GUNIGW Server Implementation file.
  - Implements the methods to be called from WSAG-Server.
- *gunigw_corba*.cxx: Corba client source file.
- *soapBESFactoryBindingService.cpp, soapBESFactoryBindingService.h, soapC.cpp, soapH.h, soapStub.h*: WS Binding files automatically generated by gSOAP from *gouni-bes-factory.wsdl* file.
- *gouni-bes-factory.wsdl*: WSDL file describing GUNI-GW Web Service.
- *bes-factory.xsd, jsdl.xsd, ws-addr.xsd*: Schema files for GUNI-GW Web Service types.

11.3 Example

Next, an example of a CreateActivity XML request is shown:

```
<s11:Envelope
xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:bes-factory="http://schemas.ggf.org/bes/2006/08/bes-factory">
<s11:Body>
  <bes-factory:CreateActivity>
  <bes-factory:ActivityDocument>
    <jsdl:JobDefinition>
       <jsdl:JobDescription>
         <jsdl:Application>
            <jsdl:ApplicationName>WISDOM</jsdl:ApplicationName>
            <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
         </jsdl:Application>
         <jsdl:DataStaging>
           <jsdl:FileName>input.dat</jsdl:FileName>
           <jsdl:FilesystemName>HOME</jsdl:FilesystemName>
           <jsdl:CreationFlag>dontOverwrite</jsdl:CreationFlag>
           <jsdl:Source>
             <jsdl:URI>http://source.org/input.dat</jsdl:URI>
           </jsdl:Source>
         </jsdl:DataStaging>
       </jsdl:JobDescription>
     </jsdl:JobDefinition>
    </bes-factory:ActivityDocument>
  </bes-factory:CreateActivity>
</s11:Body>
</s11:Envelope>
```

12 G.UNI and G.E-NNI RSVP-TE

The G.UNI and G.E-NNI RSVP-TE protocol controllers are derived from the I-NNI G.RSVP-TE protocol controller documented in section 7.

This is made possible by a specific design choice: the I-NNI G.RSVP-TE is a superset of G.I-NNI, G.UNI and G.E-NNI objects and functions, specified in D2.2 and D2.7 for the signalling part. This includes (but it is not limited to) the parsing and formatting of G.UNI and G.E-NNI specific objects (e.g. the GENERALIZED_UNI), that could cross the I-NNIs as RSVP opaque objects.

This design choice allowed to obtain a more flexible and complete G.RSVP-TE protocol controller, and easier to maintain.

Some G.UNI and G.E-NNI specificities still exist in the G.UNI and G.E-NNI PCs, but have a limited impact and are not relevant in a high-level software design discussion.

13 G².OSPF-TE (I-NNI, E-NNI and UNI-N/C)

The overall OSPF-TE software architecture and details are documented in the QUAGGA v0.9.9.7 reference documents.

The Phosphurus additions to migrate to G².OSPF-TE mainly consisted of the parsing and formatting of TE LSA and the new Grid LSA, and impacted the following files:

- <sw_root>/ospfd/ospf_te.h
- <sw_root>/ospfd/ospf_te.c
- <sw_root>/ospfd/ospf_grid.h
- <sw_root>/ospfd/ospf_grid.c
- <sw_root>/ospfd/ospf_vty.c

Other areas of intervention concerned the network interface of OSPF, which is now sending and receiving PDUs via its interface to the SCNGW. This work consisted of integrating an SCNGW Client in the OSPF, as explained in section 6.

14 Software structure

The G^2MPLS code is based on the substrate of Quagga v0.99.7 routing suite [QUAGGA-DOC] from which it inherits the base OSPFv2 implementation and some common libraries and tools. Different functionalities and modules are implemented in the form of independent processes. The phosphorus-g2mpls package includes software components developed from scratch, base Quagga protocols extended for Grid and GMPLS support, additional tools for the automatic generation of FSM skeletons, extensions to the Quagga library for GMPLS.

All the processes import the Quagga library and the common framework for Inter-Process Communication (IPC). The main modules are identified in Figure 14-1 and a short description is provided. Detailed software decomposition is specified in the following of this document.

14.1 Configuration process

The Phosphorus software configuration process inherits the Quagga one, which is based upon the commonly called autotools suite. The autotools suite is mainly composed by three different GNU tools: autoconf (http://www.gnu.org/software/autoconf), automake (http://www.gnu.org/software/automake) and libtool (http://www.gnu.org/software/libtool). An in-depth overview for each tool is out of scope for this document. We will present a simple overview of the process in the following chapters

14.1.1 The configuration process from the user perspective

The Phosphorus software package comes with a set of scripts built during the development process. The most important script is "*configure*" and is available in the package root directory.

A user who wants to compile and install the package must run the `configure' script in order to prepare the source tree to be built on a particular system. The actual build process is performed using the make program.

The `configure' script tests system features then makes the results of those tests available to the program while it is being built.

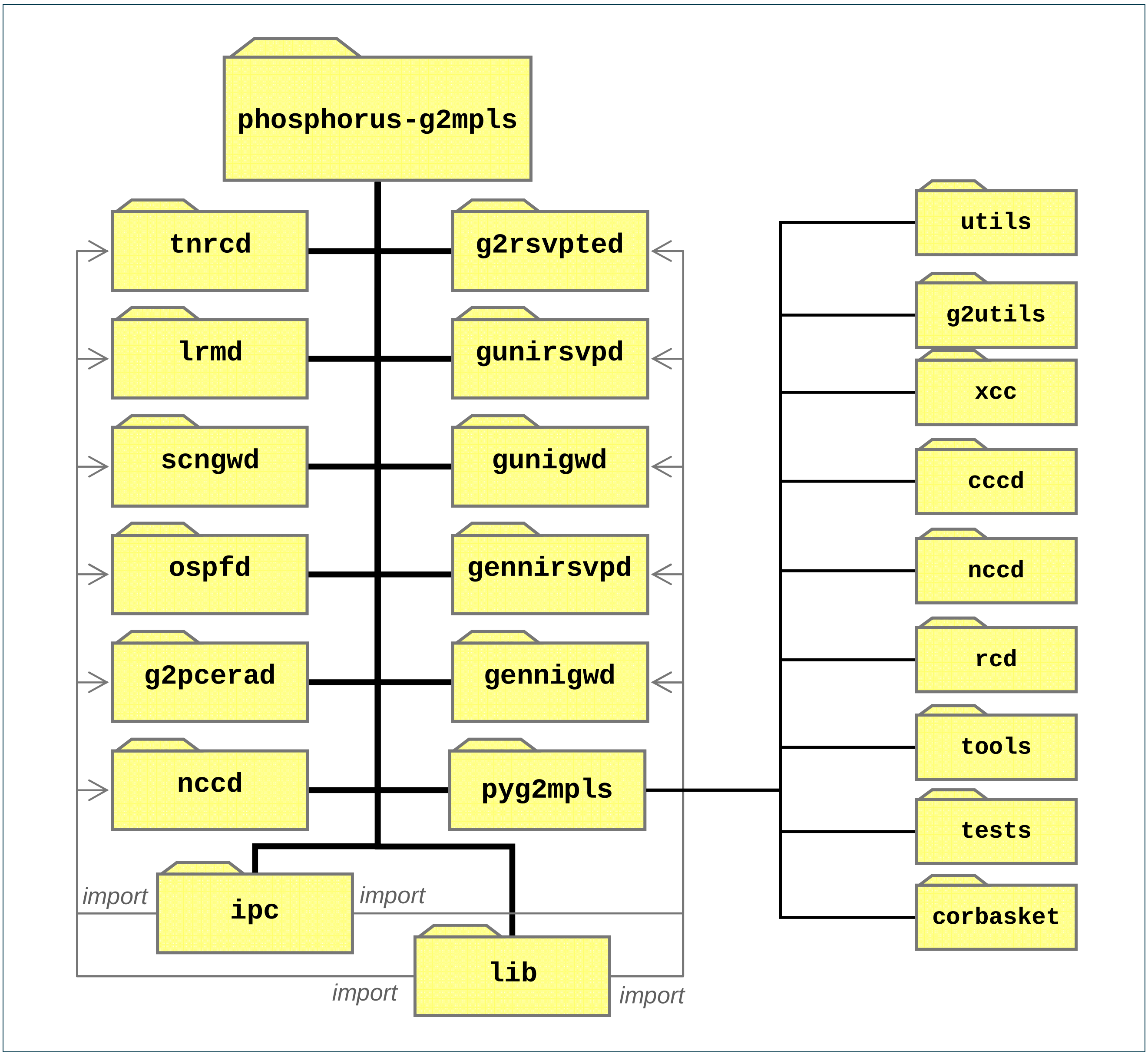

Figure 14-1: Phosphorus G2MPLS code structure.

The usual commands that should be invoked from the root Phosphorus source directory are the following ones:

```
./configure & make all install
```

At the end of the build procedure the software should be built and installed correctly on the host system.

14.1.2 The configuration process from the developer perspective

The main input files for the configuration and building processes are the root `configure.ac' and all `Makefile.am' scattered in the source tree. There is also a bunch of other files required by the autotools suite which are interesting only from the maintainer point of view.

The autotools setup process requires some standard steps which are not needed anymore after the setup. The developer which does not need to tweak the configuration process usually changes a subset of all Makefile.am files of the source tree. The autotools setup is in charge of updating the developer environment consistently upon a Makefile.am update.

The files produced by the autotools are not stored into the repository itself because they depend on the developer versions of the autotools components. Stripping the repository from unnecessary files eases the maintenance and shortens its size.

An 'autogen.sh' script which bootstraps a fresh checkout is provided in the root directory of the package, such script simply rebuilds all the required files using the autotools suite available in the developer system.

14.2 Process start-up and monitoring

In order to start-up, shut-down and monitor the Phosphorus processes the 'monit' program has been selected (http://www.tildeslash.com/monit). Monit is a widely spread utility for managing and monitoring, processes, files, and directories on a UNIX systems. It can start a process if it does not run, restart it if it does not respond and stop it if it uses too many resources.

Monit is controlled via a configuration file based on a free-format, token-oriented syntax. Monit logs messages to syslog or to its own log file and sends notifies about error conditions and recovery status via customizable alerts.

The following excerpt shows the format of the input configuration file:

```
#
# Poll at 1-minute intervals.
#
set daemon  30

set mailserver your.mail.server
```

```
#
# Set syslog logging.
#
set logfile syslog facility log_daemon

#
# Set a default mail from-address for all alert messages emitted by monit.
#
set mail-format { from: mail@domain}

#
# Send alert to system admin on any event
#
set alert mail@domain

#
# Enable http support
#
set httpd port 2621
    allow localhost

#
# check process scngwsd
#
check process scngwsd with pidfile /var/run/scngwsd.pid
       start program = "/etc/monit/scngwsd.start"
       stop program  = "/etc/monit/scngwsd.stop"
       if failed port 2620 type tcp with timeout 15 seconds then restart
       alert mail@domain
       with mail-format {
              from:     mail@domain
              subject:  scngwsd $EVENT - $ACTION
              message:  This event occurred on $HOST at $DATE.
              Regards,
              monit
        }
       if cpu is greater than 60% for 2 cycles then alert
       if cpu > 80% for 5 cycles then restart
       if mem > 20 MB then alert
       if loadavg(5min) greater than 10 for 8 cycles then stop
       if 3 restarts within 5 cycles then timeout
       group quagga

#
# check process lrmd
#
check process LRMd with pidfile /var/run/lrmd.pid
       start program = "/etc/monit/lrmd.start"
       stop program  = "/etc/monit/lrmd.stop"
       if failed port 2610 type tcp with timeout 15 seconds then restart
       alert mail@domain
       with mail-format {
              from:     mail@domain
              subject:  lrmd $EVENT - $ACTION
              message:  This event occurred on $HOST at $DATE.
                     Regards,
                     monit
```

```
         }
       if cpu is greater than 60% for 2 cycles then alert
       if cpu > 80% for 5 cycles then restart
       if mem > 20 MB then alert
       if loadavg(5min) greater than 10 for 8 cycles then stop
       if 3 restarts within 5 cycles then timeout
       group quagga
#
# check process tnrcd
#
check process tnrcd with pidfile /var/run/tnrcd.pid
       start program = "/etc/monit/tnrcd.start"
       stop program  = "/etc/monit/tnrcd.stop"
       if failed port 2610 type tcp with timeout 15 seconds then restart
       alert mail@domain
       with mail-format {
              from:     mail@domain
              subject:  tnrcd $EVENT - $ACTION
              message:  This event occurred on $HOST at $DATE.
                     Regards,
                     monit
         }
       if cpu is greater than 60% for 2 cycles then alert
       if cpu > 80% for 5 cycles then restart
       if mem > 20 MB then alert
       if loadavg(5min) greater than 10 for 8 cycles then stop
       if 3 restarts within 5 cycles then timeout
       group quagga
```

Code 14-1: Configuration file for stack start-up and monitoring.

14.3 Inter-process communication

The Quagga software is composed by a multitude of processes, all of them use a socket based intercommunication library to exchange messages. The involved software is located in the ‘zebra‘and ‘lib’ directories (zebra/zserv.c, zebra/zserv.h, lib/zclient.c and lib/zclient.h files).

Such mechanism is easy to extend and simple to use in communication environments characterized by simple, fixed size and unstructured messages. In a GMPLS context like the Phosphorus one it cannot be used because messages present the opposite nature: they are usually highly structured, variable sized and often unstructured.

In order to cope with such an environment the Quagga IPC mechanisms has been replaced using the CORBA middleware.

The **CORBA** framework [CORBA] is an industry-level middleware, defined by the Object Management Group (OMG), which allows to normalize the method-call semantics (in a language-independent fashion) among application objects that are located either in the same address space (i.e. application) or remote address space (i.e. local or remote host). The CORBA framework adopts and *Interface Definition Language (IDL)* to specify the interfaces between different modules, and are translated by IDL compilers into client and servant side code in specific programming languages. The client code acts as a proxy in order to contact the server side. The servant code is a skeleton which usually must be inherited and expanded to process the clients requests. An ORB is usually provided with all the libraries needed to handle CORBA communications, the ORB user simply fills the logic portions involved in the communication.

CORBA has been selected because it is complete and powerful inter-process and inter-platform communication architecture. The ORB adopted by the Phosphorus team is the one developed within the **omniORB** project [omniORB], which is an LGPL (Lesser GPL) CORBA ORB for C++ and Python. It has been chosen for its ability to provide CORBA features in a sufficiently light and manageable suite.

14.3.1 omniORB

OmniOrb is a robust CORBA ORB with C++ and Python bindings, it is largely CORBA 2.6 compliant and it is fully interoperable with other CORBA ORBs.

omniORB is fully multithreaded. To achieve low call overhead, unnecessary call-multiplexing is eliminated. With the default policies, there is at most one call in-flight in each communication channel between two address spaces at any one time. To do this without limiting the level of concurrency, new channels connecting the two address spaces are created on demand and cached when there are concurrent calls in progress. Each channel is served by a dedicated thread. This arrangement provides maximal concurrency and eliminates any thread switching in either of the address spaces to process a call.

Furthermore, to maximise the throughput in processing large call arguments, large data elements are sent as soon as they are processed while the other arguments are being marshalled. With GIOP 1.2, large messages are fragmented, so the marshaller can start transmission before it knows how large the entire message will be.

From version 4.0 onwards, omniORB also supports a flexible thread pooling policy, and supports sending multiple interleaved calls on a single connection. This policy leads to a small amount of additional call overhead, compared to the default thread per connection model, but allows omniORB to scale to extremely large numbers of concurrent clients.

omniORB uses real C++ exceptions and nested classes. It keeps to the CORBA specification's standard mapping as much as possible and does not use the alternative mappings for C++ dialects. The only exception is the mapping of IDL modules, which can use either namespaces or nested classes.

omniORB relies on native thread libraries to provide multithreading capability. A small class library (omnithread) is used to encapsulate the APIs of the native thread libraries. In application code, it is recommended but not mandatory to use this class library for thread management. It should be easy to port omnithread to any platform that either supports the POSIX thread standard or has a thread package that supports similar capabilities.

omniORB is available for download at the following URL: http://omniorb.sourceforge.net

14.3.2 Quagga daemons and threads

Traditional routing software is made as a one process program which provides all of the routing protocol functionalities. Quagga takes a different design approach: it is made from a collection of several daemons that work together to build the routing table. There may be several protocol-specific routing daemons and zebra the kernel routing manager.

For changing the kernel routing table and for redistribution of routes between different routing protocols, there is a kernel routing table manager zebra daemon. It is easy to add a new daemon to the system without affecting any other software. There is no need for these daemons to be running on the same machine.

At the moment the Quagga software was planned, the thread library which comes with GNU/Linux or FreeBSD had some problems running reliable services such as routing software. The Quagga team decided to avoid threads at all, preferring a select() approach for multiplexing system events.

Quagga software is divided into daemons. Each daemon run as a separate process and exchanges its data with the others via a socket based communication. Each process is divided into quagga-threads, a quagga-thread is a software simulated thread which use the select() approach. Each daemon is linked with the Quagga library which provides a thread/event scheduler for the running process. The scheduler selects a timer, an event, a thread or a network operation and runs its related handling procedure. Each running object must be cooperative with the others, it must explicitly yield to the CPU in order to let the others run in multithread-like environment.

14.3.3 omniORB integration in Phosphorus

While an ORB is multi-threaded by nature, the Quagga software is single-threaded by design.

In order to integrate omniORB with Quagga, without modifying the whole Quagga software base, a mutex approach has been selected. The mutex separates the Quagga scheduler from the ORB main loop and let them run in different time slices. The Quagga scheduler works as usual, serving pending tasks if available. In the meanwhile the ORB is stuck to the mutex which prevents the ORB and a Quagga tasks to run in parallel.

When the scheduler detects an idle status (no pending threads to serve) unblocks the ORB by releasing the mutex. The ORB main loop starts running, serving CORBA requests for a specific amount of time. When the allocated time-slice elapses, the ORB gives back the control to the Quagga scheduler.

In order to implement the described behaviour the CORBA servants must follow the software-contract described in 14.3.3.4

14.3.3.1 *CORBA clients and servers common calls*

Both CORBA clients and CORBA servers (which could be composed by a set of CORBA servant) should call the library provided corba_init() function at startup and corba_fini() function at shutdown.

The corba_init() function initializes the ORB data structures and resolves initial references to the root POA. The corba_fini() function is provided for symmetry and should perform clean-up actions if needed.

14.3.3.2 *CORBA clients utility library*

Client side software should follow a standard initialization phase which is composed by the following calls:

a) corba_init()

b) corba_client_setup(): Retrieves the ORB reference, fetches the involved servant IOR, narrows the reference and setups relevant data structures

The finalization phase is composed by the following calls:

a) corba_client_shutdown(): Performs clean-up actions if needed

b) corba_client_fini()

14.3.3.3 *CORBA server utility library*

A Quagga based CORBA server must adhere to the call sequence that follows:

a) corba_init()

b) corba_server_setup(): Retrieves the POA reference, activates the servant and builds the IOR file describing the servant access point, stores the POA Manager reference for later usage and finally activates the POA Manager

The finalization phase is composed by the following calls:

a) corba_server_shutdown(): Removes the IOR files which has been generated by corba_server_setup()

b) corba_fini()

14.3.3.4 *CORBA servants requirements*

Each servant must use the following skeleton on each method:

```
type servant::method(parms)
{
        STACK_LOCK();
        …
        STACK_UNLOCK();
}
```

Code 14-2: CORBA servant requirements

The STACK_LOCK() and STACK_UNLOCK() provide the locking/unlocking mechanism which drives the CORBA/Quagga behaviour. A missing STACK_LOCK()/STACK_UNLOCK() will cause unpredictable results in the whole server process

14.4 G²MPLS base Python modules

The founding Python modules developed for the G²MPLS project in WP2 are briefly explained in the following:

baseobj

The *baseobj* module introduce a number of basic object to be used by the Python-based protocol controllers, such as: *BasicObject:* a wrapper for the native Python *object* with a number of extra features (e.g. logging facilities); *BasicLock* and *BasicLocksTable:* wraps the *thread* locks to make deadlocks easily debuggable; *BasicTable* and *ParmsBox*: wraps basic dicts with locking facilities, in order to provide a powerful tool to prevent a simultaneous access to critical objects (e.g. the table of Calls at the CCC or NCC).

bits

Introduces some classes for bitmask and address (node IDs, IPv4, IPv6, NSAP, MAC) manipulation.

corbahelper

An extensive wrapper to ease the creation of omniORB servants and clients. I.e. it provides safe wrappers for client method invocations (e.g. retrying to read the IOR on transient exceptions), or the powerful and flexible creation of omniORB servant classes and related methods, with minimum involvement in details of the omniORB inner workings required from the programmers of protocol controllers.

fsm

This module implements a flexible and configurable Finite State Machine engine. The FSM architecture is in line with that specified for the G²MPLS framework (e.g. based on root events and detailed events, and supporting both inbound and outbound transitions). The FSM has a queue of incoming events that can be posted in a blocking or non-blocking fashion, and are executed by a thread sleeping on the queue. When multiple instances of FSM exist (e.g. one per Call), the scalability of the system is greatly enhanced by configuring the execution of all the transitions with a single thread, rather than multiple threads (one per FSM).

logger

A module implementing tracing facilities, with log classes and differentiated tracing levels for each class.

netutils

Allows to retrieve some info about the SCN interfaces of the G²MPLS controller.

protocol

The classes in this module incorporate some basic functionalities in order to simplify the development of protocol controllers and protocol objects. In particular, the *Protocol* class already include a number of functions related to the logging facility, the initialization and handling of CORBA, timers, network communication and FSM. Any protocol class derived from this (e.g. the *NetworkCallController*) will inherit all these functionalities automatically. The *ProtoObject* class does the same for protocol objects, such as the *Call*.

timer

The timer module implements a calendar of timer events where all the timers of a protocol controller are scheduled. This solution drastically increases the scalability of the timers management: just one timer (i.e. thread) is needed for the whole calendar independently of the number of scheduled events, compared to the standard solution where 1 thread exists per each scheduled timer.

udpcomm

Implements an UDP client and server.

xmlmsg

Implements a parser and formatter of XML-based signalling messages.

g2types

This module includes a number of classes for the management of G²MPLS stuff; e.g. identifiers (Data Link, TE Link, labels, TNA, LSP, Call, Recovery Bundle, etc.) and clusters of parameters (transport network resources, LSP parameters, Call parameters, recovery parameters).

14.5 Software daemons

14.5.1 lrmd

This module is responsible for the management of the relationships among TE-links, Data-links, Control channels and SCN interfaces. The TE-links are the result of a bundling procedure applied to a number of physical component data-links with the eligibility for being part of the same logical construct.

The functionalities of the LRM comprise:

- Selection and allocation/de-allocation of resources (<Data-link, label>) in TE-link for signalling purposes,
- Management of the TE-link status and bundling information for topology purposes.

lrmd exposes interfaces to gunirsvpd, g2rsvpted, tnrcd, ospfd, scngwd and g2nccd.

lrmd is not part of Quagga routing suite and is developed from scratch.

14.5.2 scngwd

This module is responsible for the management of the dualism between the Transport Network and Signalling Network. In transmission, it correlates SDUs sent by the G^{2}MPLS protocols towards TE-link source/destinations to the actual and active control channel and SCN interface configured on the G^{2}MPLS controllers for that TE-links pair.

In reception, scngwd selects the protocol instance and TE-link on which the SDUs received from the SCN interface must be sent to.

The scngwd module is further broken down in two sub-modules as described in Table 14-1.

module	sub-module	short description

SCNGW (SCN Gateway)	SCNGW client (SCNGWC)	Library offering a wrapped socket API, to be linked by each protocol wanting communication across the SCN. It acts as an access i/f to the SCNGW server, and has 2 channels with it: 1 for data, 1 for control (e.g. open/close sockets, etc.)
	SCNGW server (SCNGWS)	Separate process (i.e. a socket manager) handling (tunnelled) communication through the SCN for one or more clients. It maps TN resources (TE links) into SCN resources (control i/fs) via the TE links <-> CCs association.

Table 14-1: SCNGW breakdown in sub-modules.

scngwd exposes interfaces to gunirsvpd, gennirsvpd, g2rsvpted, ospfd, lrmd.

scngwd is not part of Quagga routing suite and is developed from scratch.

14.5.3 tnrcd

This module is responsible for abstracting the technology specific details of the transport network resources for control plane use. The main functionalities of the Transport Network Resource Controller are:

- translation and maintenance of the bindings between the technology specific name space for transport resources (e.g. in DWDM equipments: <port, wavelength>; in TDM: <port, virtual container>; in Ethernet: <port, VLANs>) and the G2MPLS name space (<data-link, label>)
- translation between the technology specific configurations for transport resources (e.g. cross-connections, protections, etc.) and the G^2MPLS corresponding actions
- binding maintenance among the resources (e.g. cross-connections, bookings, protections/restorations, etc.).

The tnrcd module is further broken down in two sub-modules as described in Table 14-2.

module	sub-module	short description
TNRC (Transport Network Resource Controller)	TNRC-AP (TNRC Abstract Part)	Process offering a generic API for the configuration & monitoring of the TN resources. It will abstract the TN resource description, and provide an atomic grouping of actions that might be composed by a set of local management sub-actions on the equipment.

	TNRC-SP (TNRC Specific Part))	Lower part of the process, loaded as plug-in, and offering the upper part an API specific to the equipment considered. It will name resources based on the underlying TN technology and SwCap. The core part of the TNRC-SP is likely to be dependent on the controlled equipment (e.g. based on some proprietary SNMP MIB sub-tree supported for configuration and monitoring).

Table 14-2: TNRC breakdown in sub-modules.

tnrcd exposes interfaces to lrmd, gunirsvpd, gennirsvpd, g2rsvpted and g2nccd.

tnrcd is not part of Quagga routing suite and is developed from scratch.

14.5.4 ospfd

This module is the OSPF routing protocol extended with GMPLS TE and Grid-GMPLS extensions (derived from the GLUE schema mapping). The module implements the routing instance for the I-NNI interface between G^2MPLS nodes. Some preliminary E-NNI extensions and control of two instances (the I-NNI's and the E-NNI's one) is also implemented as part of the extensions for G^2MPLS interfacing (Task 2.2 - Activity A2.2.2).

ospfd exposes interfaces to lrmd, pcerad and scngwd.

ospfd in phosphorus-g2mpls is extended for G^2MPLS with respect to the Quaggav0.99.7 baseline.

14.5.5 g2rsvpted

This module is the RSVP-TE signalling protocol extended with GMPLS TE and Grid-GMPLS extensions (derived from the JSDL schema mapping). The module implements the I-NNI signalling between G^2MPLS nodes.

g2rsvpted exposes interfaces to lrmd, tnrcd, g2nccd, pcerad and scngwd.

g2rsvpted is not part of Quagga routing suite and is developed from scratch.

14.5.6 gunirsvpd

This module is the UNI RSVP signalling protocol extended with OIF UNI-RSVP and Grid-GMPLS extensions (derived from the JSDL schema mapping). The module implements the G.UNI signalling between a G^2MPLS user and the node at the edge of a G^2MPLS domain.

gunirsvpd exposes interfaces to lrmd, tnrcd, g2nccd and scngwd.

gunirsvpd is not part of Quagga routing suite and is developed from scratch.

14.5.7 gennirsvpd

This module is the E-NNI RSVP signalling protocol extended with OIF ENNI-RSVP and Grid-GMPLS extensions (derived from the JSDL schema mapping). The module implements the G.E-NNI signalling between two border nodes of adjacent G^2MPLS domains.

gennirsvpd exposes interfaces to lrmd, tnrcd, g2nccd and scngwd.

gennirsvpd is not part of Quagga routing suite and is developed from scratch.

14.5.8 g2nccd

This module is the GNS Transaction and G^2MPLS Call Controller. It controls (setup and recovery) the end-to-end call and in particular the segment implemented by the G^2MPLS domain in which it operated.

g2nccd exposes interfaces to lrmd, tnrcd, g2rsvpted, gunirsvpd, gennirsvpd and pcerad.

g2nccd is not part of Quagga routing suite and is developed from scratch.

14.5.9 g2pcerad

This module implements the routing algorithm for the path computation of call segments.

g2pcera exposes interfaces to g2rsvpted, g2nccd and ospfd.

g2pcera is not part of Quagga routing suite and is developed from scratch.

14.5.10 lib

This library contains many common utilities of the Quagga framework that have been extended for G^2MPLS purposes. The core VTY implementation as well as the zebra client/server and the redefinition and control of zebra pseudo-threads are part of the original Quagga v0.997 baseline. Common GMPLS types and addresses as well as some related set/get/print utilities have been added to the Quagga baseline. The library is linked by all the processes in the phosphorus-g2mpls.

lib in phosphorus-g2mpls is extended for G^2MPLS with respect to the Quaggav0.99.7 baseline.

14.5.11 pyg2mpls

This folder is the collection of Python-based protocol controllers (CCC, NCC and RC), plus a number of common utilities (utils/, g2utils/, xcc/). The protocol controllers are contained in cccd, nccd and rcd, respectively.

The NCC VTY is implemented in <sw_root>/nccd/.

15 References

As explained in section 1, the references listed here are only those directly functional to this document. For a list of the references to standards appearing in this document, please point to D2.1, D2.2 and D2.7.

[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".

[QUAGGA-DOC] The Quagga Software Routing Suite documentation. http://www.quagga.net/docs/docs-info.php

[CORBA] http://www.corba.org/

[omniORB] http://omniorb.sourceforge.net/

16 Acronyms

AAA	Authentication, Authorisation, and Accounting
AAI	Authentication and Authorization Infrastructure
ANSI	American National Standards Institute
API	Application Programming Interface
ARGON	Allocation and Reservations in Grid-enabled Optical Networks
ASON	Automatically Switched Optical Network
BB	Bandwidth Broker
BGRP	Border Gateway Reservation Protocol
BoD	Bandwidth on Demand
BR	Border Router
CE	Computing Element
CIM	Computer Integrated Manufacturing
COPS	Common Open Policy Protocol
CORBA	Common Object Request Broker Architecture
CP	Control Plane
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CR-LDP	Constraint-based Label Distribution Protocol
DCM	Distributed Call and Connection Management
DCN	Data Communication Network
DRAC	Dynamic Resource Allocation Controller
DVB	Digital Video Broadcasting
DWDM	Dense Wavelength Division Multiplexing
EGEE	Enabling Grids for E-sciencE
EC	European Commission
EMS	Execution Management Services
E-NNI	Exterior NNI
ERO	Explicit Route Object
ETSI	European Telecommunications Standards Institute

EU	European Union
FCAPS	Fault, Configuration, Accounting, Performance, Security
G.CR-LDP	G^2MPLS CR-LDP
G.OSPF-TE	GMPLS OSPF-TE
G.UNI	Grid UNI
G.UNI-C	G.UNI - Client
G.UNI-N	G.UNI - Network
G.RSVP-TE	GMPLS RSVP-TE
G^2MPLS	Grid-GMPLS (enhancements to GMPLS for Grid support)
GE	Gigabit Ethernet
GÉANT	Pan-European Gigabit Research Network
GGF	Global Grid Forum
GHPN	Grid High Performance Networking
GIS	Grid Information Service
GLUE	Grid Laboratory Uniform Environment
GMPLS	Generalized MPLS
GNS	Grid Network Service
GRAM	Grid Resource Allocation and Management
GSMP	General Switch Management Protocol
HW	Hardware
IANA	Internet Assigned Numbers Authority
IDM	GÉANT2 Inter-domain Manager
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
I-NNI	Interior NNI
IP	Internet Protocol
IPR	Intellectual Property Right
IPSec	IP security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IS-IS	Intermediate System to Intermediate System
ITU	International Telecommunication Union
JSDL	Job Submission Description Language
LAN	Local Area Network
LDP	Label Distribution Protocol
LRMS	Local Resource Management System
LSA	Link State Advertisement
LSDB	Link State Database
LSP	Label Switched Path
LSR	Label Switch Router

MAC	Media Access Control
MAN	Metropolitan Area Network
MP	Management Plane
MPLS	Multi Protocol Label Switching
MPI	Message Passing Interface
NCP	Network Control Plane
NJS	Network Job Supervisor
NMS	Network Management System
NNI	Network to Network Interface
NO	Network Operator
NREN	National Research and Education Network
NRPS	Network Resource Provisioning Systems
NSAP	Network Service Access Point
NSP	Network Service Plane
NTP	Network Time Protocol
OAM	Operations, Administration and Maintenance
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OIF	Optical Internetworking Forum
OS	Operating System
OSPF	Open Shortest Path First protocol
OSPF-TE	OSPF with Traffic Engineering extensions
O-UNI	Optical UNI
P2MP	Point to Multi Point
PON	Passive Optical Network
POSIX	Portable Operating System Interface
QoS	Quality of Service
RB	Recovery Bundle (aka RecoBundle)
RC	Routing Controller
RFC	Request for Comments
RSVP	Resource reSerVation Protocol
RSVP-TE	RSVP with Traffic Engineering extensions
RTP	Real-time Transport Protocol
SDO	Standard Developing Organizations
SE	Storage Element
SLA	Service Level Agreement
SLS	Service Level Specification
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
SPF	Sender Policy Framework

SW	Software
TE	Traffic Engineering
TGC	Trusted Computing Group
TL-1	Transaction Language 1
TLS	Transport Layer Security
TLV	Type-Length-Value protocol fields
TMF	Tele Management Forum
TO	Telecom Operator
TP	Transport Plane
UCLP	User-Controlled Lightpath Provisioning system
UNI	User to Network Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VLAN	Virtual LAN
VPN	Virtual Private Network
WAN	Wide Area Network
WG	Working Group
WP	Work Package
WS	Web Service
WSON	Wavelength Switched Optical Network
XML	Extensible Markup Language

Appendix A Common types

The Phosphorus G^2MPLS common types used on the external interfaces among processes are specified in the *<sw_root>/idl/g2mplsTypes.idl* file.

It is useful to report this detailed information here, since it can be easily read by humans, and provide an interesting insight of the overall G²MPLS data model.

A.1 Identifiers

```
	// Neighbour & adjacency
	typedef Types::uint32			nodeId;
	typedef nodeId				adjacencyId;


	// generic address
	typedef Types::uint32			addrIPv4;
	typedef Types::uint32			addrIPv6[4];
	typedef Types::uint32			addrUnnum;
	typedef Types::uint8			addrNSAP[20];
	typedef Types::uint8			addrMAC[6];

//	struct addrUnnum {
//		nodeId			node;
//		Types::uint32		addr;
//	};

	enum addrType {
		ADDRTYPE_IPV4,
		ADDRTYPE_IPV6,
		ADDRTYPE_UNNUM,
		ADDRTYPE_NSAP,
		ADDRTYPE_MAC
	};

	union addr switch (addrType) {
		case ADDRTYPE_IPV4:		addrIPv4	ipv4;
```

```
        case ADDRTYPE_IPV6:                addrIPv6     ipv6;
        case ADDRTYPE_UNNUM:               addrUnnum    unnum;
        case ADDRTYPE_NSAP:                addrNSAP     nsap;
        case ADDRTYPE_MAC:                 addrMAC      mac;
};
```

A.2 Label identifier

```
enum labelType {
        LABELTYPE_L32,
        LABELTYPE_L60
};

union labelId switch (labelType) {
        case LABELTYPE_L32:                Types::uint32       label32;
        case LABELTYPE_L60:                Types::uint64       label60;
};
```

A.3 TE-Link and Data Link

```
enum linkIdType {
        LINKIDTYPE_IPV4,
        LINKIDTYPE_IPV6,
        LINKIDTYPE_UNNUM
};

union linkId switch (linkIdType) {
        case LINKIDTYPE_IPV4:              addrIPv4     ipv4;
        case LINKIDTYPE_IPV6:              addrIPv6     ipv6;
        case LINKIDTYPE_UNNUM:             addrUnnum    unnum;
};

typedef linkId             TELinkId;
typedef linkId             DLinkId;

enum adjType {
        ADJTYPE_UNI,
        ADJTYPE_INNI,
        ADJTYPE_ENNI
};
```

A.4 TNA identifier

```
enum tnaIdType {
        TNAIDTYPE_IPV4,
        TNAIDTYPE_IPV6,
```

```
        TNAIDTYPE_NSAP
    };

    union tnaId switch (tnaIdType) {
        case TNAIDTYPE_IPV4:            addrIPv4     ipv4;
        case TNAIDTYPE_IPV6:            addrIPv6     ipv6;
        case TNAIDTYPE_NSAP:            addrNSAP     nsap;
    };
    typedef sequence<tnaId>             tnaIdSeq;
```

A.5 Call, Recovery Bundle and LSP identifiers

```
    enum callIdType {
        CALLIDTYPE_NULL,
        CALLIDTYPE_OPSPEC,
        CALLIDTYPE_GLOBUNIQ
    };

    enum sourceIdType {
        SOURCEIDTYPE_IPV4,
        SOURCEIDTYPE_IPV6,
        SOURCEIDTYPE_NSAP,
        SOURCEIDTYPE_MAC
    };

    union sourceId switch (sourceIdType) {
        case SOURCEIDTYPE_IPV4:         addrIPv4     ipv4;
        case SOURCEIDTYPE_IPV6:         addrIPv6     ipv6;
        case SOURCEIDTYPE_NSAP:         addrNSAP     nsap;
        case SOURCEIDTYPE_MAC:          addrMAC      mac;
    };

    struct segments {
        Types::uint8            intlSeg[3];
        Types::uint32           natlSeg[3];
    };

    struct callIdent {
        callIdType              idType;
        segments                segs;
        sourceId                srcId;
        Types::uint64           localId;
    };


    struct recoBundleIdent {
        nodeId                  srcAddr;
        nodeId                  dstAddr;
        Types::uint32           tunId;
    };


    struct lspIdent {
        nodeId                  dstNodeId;
```

```
            nodeId                      srcNodeId;
            Types::uint32               tunId;
            Types::uint32               extTid;
            Types::uint32               lspId;
    };
```

A.6 GMPLS extensions

```
    enum labelState {
            LABELSTATE_FREE,
            LABELSTATE_BOOKED,
            LABELSTATE_XCONNECTED,
            LABELSTATE_BUSY
    };

    enum resourcePosition {
            RESOURCEPOSITION_INGRESS,
            RESOURCEPOSITION_EGRESS
    };

    enum operState {
            OPERSTATE_UP,
            OPERSTATE_DOWN
    };

    enum adminState {
            ADMINSTATE_DISABLED,
            ADMINSTATE_ENABLED
    };

    struct statesBundle {
            operState                   opState;
            adminState                  admState;
    };

    enum recoveryType {
            RECOVERYTYPE_UNPROTECTED,
            RECOVERYTYPE_PROTECTION,
            RECOVERYTYPE_PREPLANNED,
            RECOVERYTYPE_OTF,
            RECOVERYTYPE_OTF_REVERTIVE
    };

    enum disjointness {
            DISJOINTNESS_NONE,
            DISJOINTNESS_LINK,
            DISJOINTNESS_NODE,
            DISJOINTNESS_SRLG
    };

    enum switchingCap {
            SWITCHINGCAP_PSC_1, // = 1,
            SWITCHINGCAP_PSC_2, // = 2,
```

```
        SWITCHINGCAP_PSC_3, // = 3,
        SWITCHINGCAP_PSC_4, // = 4,
        SWITCHINGCAP_L2SC , // = 51,
        SWITCHINGCAP_TDM  , // = 100,
        SWITCHINGCAP_LSC  , // = 150,
        SWITCHINGCAP_FSC    // = 200
};

enum encodingType {
        ENCODINGTYPE_PACKET     , // = 1,
        ENCODINGTYPE_ETHERNET          , // = 2,
        ENCODINGTYPE_ANSI_ETSI_PDH  , // = 3,
        ENCODINGTYPE_RESERVED_1        , // = 4,
        ENCODINGTYPE_SDH_SONET         , // = 5,
        ENCODINGTYPE_RESERVED_2        , // = 6,
        ENCODINGTYPE_DIGITAL_WRAPPER, // = 7,
        ENCODINGTYPE_LAMBDA     , // = 8,
        ENCODINGTYPE_FIBER      , // = 9,
        ENCODINGTYPE_RESERVED_3        , // = 10,
        ENCODINGTYPE_FIBERCHANNEL   , // = 11,
        ENCODINGTYPE_G709_ODU          , // = 12,
        ENCODINGTYPE_G709_OC             // = 13,
};

enum genPid {
        GPID_ASYNCH_E4                , // =  5,
        GPID_ASYNCH_DS3_T3     , // =  6,
        GPID_ASYNCH_E3                , // =  7,
        GPID_BIT_SYNCH_E3      , // =  8,
        GPID_BYTE_SYNCH_E3     , // =  9,
        GPID_ASYNCH_DS2_T2     , // = 10,
        GPID_BIT_SYNCH_DS2_T2         , // = 11,
        GPID_ASYNCH_E1                , // = 13,
        GPID_BYTE_SYNCH_E1     , // = 14,
        GPID_BYTE_SYNCH_31DS0         , // = 15,
        GPID_ASYNCH_DS1_T1     , // = 16,
        GPID_BIT_SYNCH_DS1_T1         , // = 17,
        GPID_BYTE_SYNCH_DS1_T1        , // = 18,
        GPID_VC_11_IN_VC_12    , // = 19,
        GPID_DS1_SF_ASYNCH     , // = 22,
        GPID_DS1_ESF_ASYNCH    , // = 23,
        GPID_DS3_M23_ASYNCH    , // = 24,
        GPID_DS3_C_PARITY_ASYNCH   , // = 25,
        GPID_VT_LOVC           , // = 26,
        GPID_STSSPE_HOVC       , // = 27,
        GPID_POS_NOSCRAMBLING_16CRC, // = 28,
        GPID_POS_NOSCRAMBLING_32CRC, // = 29,
        GPID_POS_SCRAMBLING_16CRC  , // = 30,
        GPID_POS_SCRAMBLING_32CRC  , // = 31,
        GPID_ATM_MAPPING       , // = 32,
        GPID_ETHERNET          , // = 33,
        GPID_SONET_SDH                , // = 34,
        GPID_DIGITAL_WRAPPER          , // = 36,
        GPID_LAMBDA            , // = 37,
        GPID_ANSI_ETSI_PDH     , // = 38,
        GPID_LAPS_X85_X86      , // = 40,
        GPID_FDDI              , // = 41,
        GPID_DQDB              , // = 42,
```

```
        GPID_FIBERCHANNEL_3    , // = 43,
        GPID_HDLC              , // = 44,
        GPID_ETH_V2_DIX              , // = 45,
        GPID_ETH_802_3               , // = 46,
        GPID_G709_ODUJ               , // = 47,
        GPID_G709_OTUK               , // = 48,
        GPID_CBR_CBRA          , // = 49,
        GPID_CBRB              , // = 50,
        GPID_BSOT              , // = 51,
        GPID_BSNT              , // = 52,
        GPID_IP_PPP_GFP              , // = 53,
        GPID_ETHMAC_GFP              , // = 54,
        GPID_ETHPHY_GFP              , // = 55,
        GPID_ESCON              , // = 56,
        GPID_FICON                // = 57,
};

enum protType {
        PROTTYPE_NONE        , //   = 0x00,
        PROTTYPE_EXTRA             , //   = 0x01,
        PROTTYPE_UNPROTECTED       , //   = 0x02,
        PROTTYPE_SHARED            , //   = 0x04,
        PROTTYPE_DEDICATED_1TO1    , //   = 0x08,
        PROTTYPE_DEDICATED_1PLUS1, //   = 0x10,
        PROTTYPE_ENHANCED      //   = 0x20,
};


enum crankbackScope {
        CRANCKBACKSCOPE_NONE         ,
        CRANCKBACKSCOPE_E2E          ,
        CRANCKBACKSCOPE_BOUNDARY     ,
        CRANCKBACKSCOPE_SEGMENTBASED
};

enum issuerType {
        ISSUERTYPE_MANAGEMENT_IF,
        ISSUERTYPE_UNI_IF,
        ISSUERTYPE_ENNI_IF
};

struct actorInfo {
        issuerType                issuer;
        boolean                         forceCommand;
};

enum lspType {
        LSPTYPE_SPC, // Soft permanent connection
        LSPTYPE_PC,  // Permanent connection
        LSPTYPE_SC   // Switched connection
};

enum lspResourceAction {
        LSPRESOURCEACTION_XCONNECT,
        LSPRESOURCEACTION_BOOK
};

enum lspRroMode {
```

```
        LSPRROMODE_OFF,             // no RRO recording
        LSPRROMODE_TEL_DETAIL,      // recoding just up to TE-links
        LSPRROMODE_DL_DETAIL,       // recoding just up to Data-links
        LSPRROMODE_ALL              // recoding all up to labels
};
// Transport Network resource
struct tnResource {
        TELinkId                    teLink;
        DLinkId                             dataLink;
        labelId                             label;
};

struct tnaResource {
        tnaId                       tna;
        DLinkId                             dataLink;// only if _v != 0
        labelId                             label;   // only if _v != 0
};
```

A.7 Grid extensions

A.7.1 Signalling-specific

```
// Grid Site Network Assigned address
typedef Types::uint32                       gsnaId;

struct rangeSpec {
        boolean                     valid;
        Types::uint32               lowerBound;
        boolean                             lbIncluded;
        Types::uint32               upperBound;
        boolean                             ubIncluded;
};

//  GRID APPLICATION
enum gridApplicationType {
        GRIDAPPLICATIONTYPE_UNKNOWN, // = 0x0000,
        GRIDAPPLICATIONTYPE_WISDOM , // = 0x0001,
        GRIDAPPLICATIONTYPE_KODAVIS, // = 0x0002,
        GRIDAPPLICATIONTYPE_TOPS   , // = 0x0003,
        GRIDAPPLICATIONTYPE_DDSS   , // = 0x0004,
        GRIDAPPLICATIONTYPE_INCA   , // = 0x0005,
        GRIDAPPLICATIONTYPE_OTHER    // = 0xFFFF,
};

struct gridApplication {
        boolean                     valid;
        gridApplicationType         type;
        Types::uint32               mjrRev;
        Types::uint32               mnrRev;
        Types::uint32               bldFix;
};
```

```
// GRID HOST ID
enum gridHostType {
      GRIDHOSTTYPE_UNDEFINED,
      GRIDHOSTTYPE_IPV4,
      GRIDHOSTTYPE_IPV6,
      GRIDHOSTTYPE_NSAP
};

union gridHostId switch (gridHostType) {
      case GRIDHOSTTYPE_UNDEFINED:     long         value;
      case GRIDHOSTTYPE_IPV4:   addrIPv4     ipv4;
      case GRIDHOSTTYPE_IPV6:   addrIPv6     ipv6;
      case GRIDHOSTTYPE_NSAP:   addrNSAP     nsap;
};

// FS RESOURCES
enum gridFsName {
      GRIDFSNAME_UNKNOWN, // = 0x00,
      GRIDFSNAME_HOME    , // = 0x01,
      GRIDFSNAME_ROOT    , // = 0x02,
      GRIDFSNAME_SCRATCH, // = 0x03,
      GRIDFSNAME_TMP     , // = 0x04,
      GRIDFSNAME_OTHER     // = 0xFF
};

enum gridFsType {
      GRIDFSTYPE_UNKNOWN   , //= 0x00,
      GRIDFSTYPE_SWAP      , //= 0x01,
      GRIDFSTYPE_TEMPORARY , //= 0x02,
      GRIDFSTYPE_SPOOL     , //= 0x03,
      GRIDFSTYPE_NORMAL    , //= 0x04,
      GRIDFSTYPE_OTHER       //= 0xFF
};

struct gridFsResources {
      boolean                 valid;
      gridFsName              fsName;
      gridFsType              fsType;
      rangeSpec               diskSpace;
      string                  mountPoint;
      string                  mountSource;
};

// SYSTEM CAPABILITIES
enum gridOsType {
      GRIDOSTYPE_UNKNOWN           , //= 0x0000,
      GRIDOSTYPE_MACOS             , //= 0x0001,
      GRIDOSTYPE_ATTUNIX           , //= 0x0002,
      GRIDOSTYPE_DGUX              , //= 0x0003,
      GRIDOSTYPE_DECNT             , //= 0x0004,
      GRIDOSTYPE_TRU64_UNIX        , //= 0x0005,
      GRIDOSTYPE_OPENVMS           , //= 0x0006,
      GRIDOSTYPE_HPUX              , //= 0x0007,
      GRIDOSTYPE_AIX               , //= 0x0008,
      GRIDOSTYPE_MVS               , //= 0x0009,
      GRIDOSTYPE_OS400             , //= 0x000A,
      GRIDOSTYPE_OS_2              , //= 0x000B,
```

```
            GRIDOSTYPE_JAVAVM              , //= 0x000C,
            GRIDOSTYPE_MSDOS               , //= 0x000D,
            GRIDOSTYPE_WIN3X               , //= 0x000E,
            GRIDOSTYPE_WIN95               , //= 0x000F,
            GRIDOSTYPE_WIN98               , //= 0x0010,
            GRIDOSTYPE_WINNT               , //= 0x0011,
            GRIDOSTYPE_WINCE               , //= 0x0012,
            GRIDOSTYPE_NCR3000             , //= 0x0013,
            GRIDOSTYPE_NETWARE             , //= 0x0014,
            GRIDOSTYPE_OSF                 , //= 0x0015,
            GRIDOSTYPE_DC_OS               , //= 0x0016,
            GRIDOSTYPE_RELIANT_UNIX        , //= 0x0017,
            GRIDOSTYPE_SCO_UNIXWARE        , //= 0x0018,
            GRIDOSTYPE_SCO_OPENSERVER      , //= 0x0019,
            GRIDOSTYPE_SEQUENT             , //= 0x001A,
            GRIDOSTYPE_IRIX                , //= 0x001B,
            GRIDOSTYPE_SOLARIS             , //= 0x001C,
            GRIDOSTYPE_SUNOS               , //= 0x001D,
            GRIDOSTYPE_U6000               , //= 0x001E,
            GRIDOSTYPE_ASERIES             , //= 0x001F,
            GRIDOSTYPE_TANDEMNSK           , //= 0x0020,
            GRIDOSTYPE_TANDEMNT            , //= 0x0021,
            GRIDOSTYPE_BS2000              , //= 0x0022,
            GRIDOSTYPE_LINUX               , //= 0x0023,
            GRIDOSTYPE_LYNX                , //= 0x0024,
            GRIDOSTYPE_XENIX               , //= 0x0025,
            GRIDOSTYPE_VM                  , //= 0x0026,
            GRIDOSTYPE_INTERACTIVE_UNIX    , //= 0x0027,
            GRIDOSTYPE_BSDUNIX             , //= 0x0028,
            GRIDOSTYPE_FREEBSD             , //= 0x0029,
            GRIDOSTYPE_NETBSD              , //= 0x002A,
            GRIDOSTYPE_GNU_HURD            , //= 0x002B,
            GRIDOSTYPE_OS9                 , //= 0x002C,
            GRIDOSTYPE_MACH_KERNEL         , //= 0x002D,
            GRIDOSTYPE_INFERNO             , //= 0x002E,
            GRIDOSTYPE_QNX                 , //= 0x002F,
            GRIDOSTYPE_EPOC                , //= 0x0030,
            GRIDOSTYPE_IXWORKS             , //= 0x0031,
            GRIDOSTYPE_VXWORKS             , //= 0x0032,
            GRIDOSTYPE_MINT                , //= 0x0033,
            GRIDOSTYPE_BEOS                , //= 0x0034,
            GRIDOSTYPE_HP_MPE              , //= 0x0035,
            GRIDOSTYPE_NEXTSTEP            , //= 0x0036,
            GRIDOSTYPE_PALMPILOT           , //= 0x0037,
            GRIDOSTYPE_RHAPSODY            , //= 0x0038,
            GRIDOSTYPE_WINDOWS_2000        , //= 0x0039,
            GRIDOSTYPE_DEDICATED           , //= 0x003A,
            GRIDOSTYPE_OS_390              , //= 0x003B,
            GRIDOSTYPE_VSE                 , //= 0x003C,
            GRIDOSTYPE_TPF                 , //= 0x003D,
            GRIDOSTYPE_WINDOWS_R_ME        , //= 0x003E,
            GRIDOSTYPE_CALDERA_OPEN_UNIX   , //= 0x003F,
            GRIDOSTYPE_OPENBSD             , //= 0x0040,
            GRIDOSTYPE_WINDOWS_XP          , //= 0x0042,
            GRIDOSTYPE_Z_OS                , //= 0x0043,
            GRIDOSTYPE_OTHER                 //= 0xFFFF,
      };
```

```
        enum gridCpuArch {
                GRIDCPUARCH_UNKNOWN , //= 0X00,
                GRIDCPUARCH_SPARC   , //= 0X01,
                GRIDCPUARCH_POWERPC , //= 0X02,
                GRIDCPUARCH_X86     , //= 0X03,
                GRIDCPUARCH_X86_32  , //= 0X04,
                GRIDCPUARCH_X86_64  , //= 0X05,
                GRIDCPUARCH_PARISC  , //= 0X06,
                GRIDCPUARCH_MIPS    , //= 0X07,
                GRIDCPUARCH_IA64    , //= 0X08,
                GRIDCPUARCH_ARM     , //= 0X09,
                GRIDCPUARCH_OTHER     //= 0XFF,
        };

        struct gridOsInfo {
                boolean                         valid;
                gridOsType              type;
                Types::uint32           mjrRev;
                Types::uint32           mnrRev;
                Types::uint32           bldFix;
        };


        struct gridSysCap {
                boolean                 valid;
                gridOsInfo              os;
                gridCpuArch             cpuArch;
                boolean                         exclusiveAccess;
        };

        // DATA STAGING
        enum gridStagingCreationFlag {
                GRIDSTAGINGCF_UNKNOWN       , //0x1
                GRIDSTAGINGCF_OVERWRITE     , //0x1
                GRIDSTAGINGCF_APPEND        , //0x2
                GRIDSTAGINGCF_DONTOVERWRITE   //0x4
        };

        struct gridDataStaging {
                boolean                 valid;
                gridFsName              fsName;
                gridStagingCreationFlag         cf;
                boolean                         delOnTermination;
                string                  fileName;
                string                  source;
                string                  target;
        };

        struct gridParams {
                gridApplication                 application; // #01
                gridHostId              candHost;    // #02
                gridFsResources                 fileSystemRes;      // #03
                gridSysCap              systemCaps;  // #04
                rangeSpec               indCpuSpeed; // #05
                rangeSpec               indCpuTime;  // #06
                rangeSpec               indCpuCount; // #07
                rangeSpec               indNetBw;    // #08
                rangeSpec               indPhyMem;   // #09
```

```
        rangeSpec                  indVirMem;   // #10
        rangeSpec                  indDiskSpace;// #11
        rangeSpec                  totCpuTime;  // #12
        rangeSpec                  totCpuCount; // #13
        rangeSpec                  totPhyMem;   // #14
        rangeSpec                  totVirMem;   // #15
        rangeSpec                  totDiskSpace;// #16
        rangeSpec                  totResCount; // #17
        gridDataStaging                  dataStaging; // #18
        gsnaId                     gridSite;       // #19
};
```

A.7.2 Routing-specific

```
struct geoCoords {
        boolean                          valid;
        Types::uint32              latResolution;
        Types::uint64              latitute;
        Types::uint32              lonResolution;
        Types::uint64              longitude;
};

struct gridSiteParams {
        string                     name;
        geoCoords                  location;
        nodeId                     peRouterId;
};

typedef Types::uint32                    gridSubNodeId;

enum gridSubNodeType {
        GRIDSUBNODETYPE_UNKNOWN,
        GRIDSUBNODETYPE_SERVICE,
        GRIDSUBNODETYPE_COMPUTINGELEMENT,
        GRIDSUBNODETYPE_SUBCLUSTER,
        GRIDSUBNODETYPE_STORAGEELEMENT
};

struct gridSubNodeIdent {
        gridSubNodeId              id;
        gridSubNodeType                  type;
};
typedef sequence<gridSubNodeIdent>     gridSubNodeIdentSeq;

struct gridSubNodes {
        gridSubNodeIdentSeq        services;
        gridSubNodeIdentSeq        compElems;
        gridSubNodeIdentSeq        subClusters;
        gridSubNodeIdentSeq        storageElems;
};


enum gridServiceType {
        SERVICE_UNKNOWN                                ,
        ORG_GLITE_WMS                                  ,
```

```
        ORG_GLITE_RGMA_LATESTPRODUCER                              ,
        ORG_GLITE_RGMA_STREAMPRODUCER                              ,
        ORG_GLITE_RGMA_DBPRODUCER                                  ,
        ORG_GLITE_RGMA_CANONICALPRODUCER                           ,
        ORG_GLITE_RGMA_ARCHIVER                                    ,
        ORG_GLITE_RGMA_CONSUMER                                    ,
        ORG_GLITE_RGMA_REGISTRY                                    ,
        ORG_GLITE_RGMA_SCHEMA                                      ,
        ORG_GLITE_RGMA_BROWSER                                     ,
        ORG_GLITE_RGMA_PRIMARYPRODUCER                             ,
        ORG_GLITE_RGMA_SECONDARYPRODUCER                           ,
        ORG_GLITE_RGMA_ONDEMANDPRODUCER                            ,
        ORG_GLITE_VOMS                                             ,
        ORG_GLITE_FIREMANCATALOG                                   ,
        ORG_GLITE_SEINDEX                                          ,
        ORG_GLITE_METADATA                                         ,
        ORG_GLITE_CHANNELMANAGEMENT                                ,
        ORG_GLITE_FILETRANSFER                                     ,
        ORG_GLITE_FILETRANSFERSTATS                                ,
        ORG_GLITE_CHANNELAGENT                                     ,
        ORG_GLITE_KEYSTORE                                         ,
        ORG_GLITE_FAS                                              ,
        ORG_GLITE_GLITEIO                                          ,
        SRM                                                        ,
        GSIFTP                                                     ,
        ORG_EDG_LOCAL_REPLICA_CATALOG                              ,
        ORG_EDG_REPLICA_METADATA_CATALOG                           ,
        ORG_EDG_SE                                                 ,
        IT_INFN_GRIDICE                                            ,
        MYPROXY                                                    ,
        GUMS                                                       ,
        GRIDCAT                                                    ,
        EDU_CALTECH_CACR_MONALISA                                  ,
        OPENSSH                                                    ,
        MDS_GIIS                                                   ,
        BDII                                                       ,
        RLS                                                        ,
        DATA_LOCATION_INTERFACE                                    ,
        PBS_TORQUE_SERVER                                          ,
        PBS_TORQUE_MAUI                                            ,
        UNICORE_CORE_TARGETSYSTEMFACTORY                           ,
        UNICORE_CORE_TARGETSYSTEM                                  ,
        UNICORE_CORE_STORAGEMANAGEMENT                             ,
        UNICORE_CORE_FILETRANSFER                                  ,
        UNICORE_CORE_JOBMANAGEMENT                                 ,
        UNICORE_CORE_REGISTRY                                      ,
        UNICORE_WORKFLOW_WORKFLOWFACTORY                           ,
        UNICORE_WORKFLOW_WORKFLOWMANAGEMENT                        ,
        UNICORE_WORKFLOW_SERVICEORCHESTRATOR                       ,
        UNICORE_WORKFLOW_GRIDRESOURCEINFORMATIONSERVICE,
        UNICORE_CISINFORMATIONPROVIDER                             ,
        SERVICE_OTHER
};

struct gridServiceInfo {
        boolean                         valid;
        gridServiceType                 type;
        Types::uint32           mjrRev;
```

```
        Types::uint32            mnrRev;
        Types::uint32            bldFix;
};

enum gridServiceState {
        GRIDSERVICESTATE_UNKNOWN,
        GRIDSERVICESTATE_OK,
        GRIDSERVICESTATE_WARNING,
        GRIDSERVICESTATE_CRITICAL,
        GRIDSERVICESTATE_OTHER
};

struct gridServiceParams {
        gridServiceInfo                  data;
        gridServiceState         state;
        gridHostId               endPointAddr;
};

enum gridLrmsType {
        GRIDLRMSTYPE_UNKNOWN,
        GRIDLRMSTYPE_OPENPBS,
        GRIDLRMSTYPE_LSF    ,
        GRIDLRMSTYPE_CONDOR ,
        GRIDLRMSTYPE_BQS    ,
        GRIDLRMSTYPE_CONDORG,
        GRIDLRMSTYPE_FBSNG  ,
        GRIDLRMSTYPE_TORQUE ,
        GRIDLRMSTYPE_PBSPRO ,
        GRIDLRMSTYPE_SGE    ,
        GRIDLRMSTYPE_NQE    ,
        GRIDLRMSTYPE_FORK   ,
        GRIDLRMSTYPE_OTHER
};

struct gridLrmsInfo {
        boolean                          valid;
        gridLrmsType             type;
        Types::uint32            mjrRev;
        Types::uint32            mnrRev;
        Types::uint32            bldFix;
};

enum gridCeSeState {
        GRIDCESESTATE_UNKNOWN    ,
        GRIDCESESTATE_QUEUING    ,
        GRIDCESESTATE_PRODUCTION,
        GRIDCESESTATE_CLOSED     ,
        GRIDCESESTATE_DRAINING
};

struct gridJobsState {
        boolean                          valid;
        Types::uint32            freeJobSlots; // just 16 lsbs
        gridCeSeState            state;
};

struct gridJobsStats {
        boolean                          valid;
```

```
		Types::uint32			runningJobs;
		Types::uint32			waitingJobs;
		Types::uint32			totalJobs;
	};

	struct gridJobsTimePerf {
		boolean				valid;
		Types::uint32			estimatedResponseTime;
		Types::uint32			worstResponseTime;
	};

	struct gridJobsTimePolicy {
		boolean				valid;
		Types::uint32			maxWallclocktime;
		Types::uint32			maxObtainableWallclockTime;
		Types::uint32			maxCpuTime;
		Types::uint32			maxObtainableCpuTime;
	};

	struct gridJobsLoadPolicy {
		boolean				valid;
		Types::uint32			maxTotalJobs;
		Types::uint32			maxRunningJobs;
		Types::uint32			maxWaitingJobs;
		Types::uint32			assignedJobSlots; // 16 lsbs
		Types::uint32			maxSlotsPerJobs;  // 16 lsbs
		Types::uint8			priority;
		boolean				preemptionFlag;
	};

	struct JobSlotsCalendarEvent {
		Types::uint32			unixTime;
		Types::uint32			JobSlots; // just 16 lsbs
	};

	typedef sequence<JobSlotsCalendarEvent>JobSlotsCalendarSeq;

	struct gridCEParams {
		gridLrmsInfo			lrmsInfo;
		gridHostId			hostAddr;
		Types::uint32			gatekeeperPort;
		string				jobManager;
		string				dataDir;
		gridSubNodeId			defaultStorageElemId;
		gridJobsState			jobsState;
		gridJobsStats			jobsStats;
		gridJobsTimePerf		jobsTimePerf;
		gridJobsTimePolicy		jobsTimePolicy;
		gridJobsLoadPolicy		jobsLoadPolicy;
		JobSlotsCalendarSeq		freeJobSlotsCalendar;
	};

	struct gridCpuCount {
		Types::uint32			physical;
		Types::uint32			logical;
	};

	struct subClusterCalendarEvent {
```

```
		Types::uint32			unixTime;
		gridCpuCount			cpuCount;
	};

	typedef sequence<subClusterCalendarEvent>	subClusterCalendarSeq;

	struct gridCpuInfo {
		boolean					valid;
		gridCpuCount			cpuCounts;
		gridCpuArch			cpuArch;
	};

	struct gridMemoryInfo {
		boolean					valid;
		Types::uint32			ramSize;
		Types::uint32			virtualMemorySize;
	};


	struct gridSubClusterParams {
		gridCpuInfo			cpu;
		gridOsInfo			os;
		gridMemoryInfo				memory;
		gridApplication				software;
		string				softwareEnvironmentSetup;
		subClusterCalendarSeq			subClusterCalendar;
	};

	enum gridStorageArch {
		GRIDSTORAGEARCH_UNKNOWN  ,
		GRIDSTORAGEARCH_DISK     ,
		GRIDSTORAGEARCH_TAPE     ,
		GRIDSTORAGEARCH_MULTIDISK,
		GRIDSTORAGEARCH_OTHER
	};

	 struct gridStorageInfo {
		boolean				valid;
		gridStorageArch				arch;
		gridCeSeState			state;
		Types::uint32			accessProtocolsMask;
		Types::uint32			controlProtocolsMask;
	};

	struct gridStorageSize {
		boolean				valid;
		Types::uint32			total;
		Types::uint32			used;
	};

	enum gridStorageRetentionPolicy {
		GRIDSTORAGERETENTIONPOLICY_UNKNOWN  ,
		GRIDSTORAGERETENTIONPOLICY_CUSTODIAL,
		GRIDSTORAGERETENTIONPOLICY_OUTPUT   ,
		GRIDSTORAGERETENTIONPOLICY_REPLICA
	};

	enum gridStorageAccessLatency {
```

```
        GRIDATORAGEACCESSLATENCY_UNKNOWN ,
        GRIDATORAGEACCESSLATENCY_ONLINE  ,
        GRIDATORAGEACCESSLATENCY_NEARLINE,
        GRIDATORAGEACCESSLATENCY_OFFLINE
};

enum gridStorageExpirationMode {
        GRIDSTORAGEEXPIRATIONMODE_UNKNOWN             ,
        GRIDSTORAGEEXPIRATIONMODE_NEVER_EXPIRE        ,
        GRIDSTORAGEEXPIRATIONMODE_WARN_WHEN_EXPIRED   ,
        GRIDSTORAGEEXPIRATIONMODE_RELEASE_WHEN_EXPIRED
};

struct gridStorageAreaInfo {
        boolean                   valid;
        Types::uint32             totalOnlineSize;
        Types::uint32             freeOnlineSize;
        Types::uint32             reservedTotalOnlineSize;
        Types::uint32             totalNearlineSize;
        Types::uint32             freeNearlineSize;
        Types::uint32             reservedNearlineSize;
        gridStorageRetentionPolicy retentionPolicy;
        gridStorageAccessLatency  accessLatency;
        gridStorageExpirationMode expirationMode;
};

struct gridStorageCount {
        Types::uint32             freeOnlineSize;
        Types::uint32             logicalCpus;
};

struct seCalendarEvent {
        Types::uint32             unixTime;
        gridStorageCount          storageCount;
};

typedef sequence<seCalendarEvent>       seCalendarSeq;

struct gridSEParams {
        gridStorageInfo                 storageInfo;
        gridStorageSize                 onlineSize;
        gridStorageSize                 nearlineSize;
        string                    storageAreaName;
        string                    storageAreaPath;
        gridStorageAreaInfo       storageAreaInfo;
        seCalendarSeq             seCalendar;
};
```

A.8 GNS call parameters

```
struct callParams {
        string                    name;
        Types::uint32             startTime;
```

```
		Types::uint32			endTime;
		string				jobName;	// GNS Call
		string				jobProject;	// GNS Call
		//nodeId			destNid;
		//endPoint			iEp;		// ing endpt
		//endPoint			eEp;		// egr endpt
	};
```

A.9 Recovery parameters

```
	struct recoveryParams {
		recoveryType			recType;
		disjointness			disjType;
	};

	enum lspRole {
		LSPROLE_UNDEFINED,
		LSPROLE_WORKER,
		LSPROLE_BACKUP
	};
```

A.10 LSP parameters

```
	struct lspParams {
		lspType						type;
		lspRole					role;
		switchingCap			swCap;
		encodingType			encType;
		genPid				gpid;
		Types::uint32			bw; // encoded IEEE FP
		Types::uint32			setupPrio;
		Types::uint32			holdingPrio;
		Types::uint32			excludeAny;
		Types::uint32			includeAny;
		Types::uint32			includeAll;
		protType			linkProtMask;
		crankbackScope				crankback;
		Types::uint32			maxCbackRetriesSrc;
		Types::uint32			maxCbackRetriesIntmd;
		lspResourceAction		action;
		lspRroMode			rroMode;
		Types::uint32			refreshInterval;
		boolean					activateAck;
		Types::uint32			rapidRetransmInterval;
		Types::uint32			rapidRetryLimit;
		Types::uint32			incrementValueDelta;
	};
```

A.11 ERO

```
        struct eroItem {
                nodeId                          node;
                TELinkId                        teLink;
                DLinkId                                 upstreamDataLink;
                DLinkId                                 downstreamDataLink;
                labelId                                 upstreamLabel;
                labelId                                 downstreamLabel;
                boolean                         loose;
        };

        typedef sequence<eroItem>       eroSeq;
```

A.12 LRM specific

```
        typedef sequence<TELinkId>      TELinkIdSeq;
        typedef sequence<DLinkId>       DLinkIdSeq;

        struct TELinkParameters {
                statesBundle                    states;
                // XXX ADD TE info
        };

        struct DLinkParameters {
                statesBundle                    states;
                switchingCap                          swCap;
                encodingType                          encType;
                Types::uint32                         maxBandwidth;
                Types::uint32                         maxResBandwidth;
                Types::uint32                         availBandwidthPerPrio[8];
                Types::uint32                         maxLSPbandwidth[8];
                  Types::uint32                         minLSPbandwidth;
        };

        struct TELinkData {
                TELinkId                              localId;
                TELinkId                              remoteId;
                nodeId                                neighbour;
                TELinkParameters                      parms;
        };

        typedef sequence<TELinkData>            TELinkDataSeq;

        struct DLinkData {
                DLinkId                               localId;
                DLinkId                               remoteId;
                DLinkParameters                       parms;
        };

        typedef sequence<DLinkData>             DLinkDataSeq;
```

A.13 TNRC specific

```
enum xcDirection {
        XCDIR_UNIDIRECTIONAL,
        XCDIR_BIDIRECTIONAL,
        XCDIR_BCAST
};

enum tnrcResult {
        TNRC_RESULT_NOERROR,
        TNRC_RESULT_EQPTDOWN,
        TNRC_RESULT_PARAMERROR,
        TNRC_RESULT_NOTCAPABLE,
        TNRC_RESULT_BUSYRESOURCES,
        TNRC_RESULT_INTERNALERROR,
        TNRC_RESULT_GENERICERROR
};
```

A.14 G^2.PCE-RA specific

```
typedef sequence<Types::uint32>  areaSeq;

enum nodeType {
        NODETYPE_UNKNOWN,
        NODETYPE_NETWORK,
        NODETYPE_GRID
};

struct nodeIdent {
        nodeId                      id;
        nodeType                    type;
};
typedef sequence<nodeIdent>             nodeIdentSeq;

struct netNodeParams {
        boolean                             isDomain;
        statesBundle                state;
        Types::uint32               colors;
        areaSeq                             areas;
};

enum linkType {
        LINKTYPE_UNKNOWN,
        LINKTYPE_TE,
        LINKTYPE_TE_SDHSONET,
        LINKTYPE_TE_G709,
        LINKTYPE_TE_WDM
};

enum linkMode {
        LINKMODE_UNKNOWN         ,
```

```
                LINKMODE_P2P_UNNUMBERED  ,
                LINKMODE_P2P_NUMBERED    ,
                LINKMODE_MULTIACCESS     ,
                LINKMODE_ENNI_INTERDOMAIN,
                LINKMODE_ENNI_INTRADOMAIN
        };

        struct iscParamsGen {
                switchingCap                swCap;
                encodingType                encType;
                Types::uint32                       maxLSPbandwidth[8];
        };

        struct iscParamsPsc {
                switchingCap                swCap;
                encodingType                encType;
                Types::uint32               maxLSPbandwidth[8];
                Types::uint32               minLSPbandwidth;
                Types::uint32               interfaceMTU; // 16 lsbs
        };

        struct iscParamsTdm {
                switchingCap                swCap;
                encodingType                encType;
                Types::uint32               maxLSPbandwidth[8];
                Types::uint32               minLSPbandwidth;
                Types::uint8                indication;
        };

        union isc switch (switchingCap) {
                case SWITCHINGCAP_PSC_1:
                case SWITCHINGCAP_PSC_2:
                case SWITCHINGCAP_PSC_3:
                case SWITCHINGCAP_PSC_4:
                        iscParamsPsc        psc;
                case SWITCHINGCAP_TDM  :
                        iscParamsTdm        tdm;
                case SWITCHINGCAP_L2SC :
                case SWITCHINGCAP_LSC  :
                case SWITCHINGCAP_FSC  :
                        iscParamsGen        gen;
        };
        typedef sequence<isc>                       iscSeq;


        typedef Types::uint32                       availBwPerPrio[8];

        struct teLinkCalendarEvent {
                Types::uint32               unixTime;
                availBwPerPrio                      availBw;
        };

        typedef sequence<teLinkCalendarEvent>  teLinkCalendarSeq;

        typedef sequence<Types::uint32>         srlgSeq;

        struct teLinkIdent {
                nodeId                      localNodeId;
```

```
            TELinkId                    localId;
            nodeId                      remoteNodeId;
            TELinkId                    remoteId;
            linkType                    type;
      };
      typedef sequence<teLinkIdent>           teLinkIdentSeq;

      struct teLinkComParams {
            linkMode                    mode;
            Types::uint32               adminMetric;
            Types::uint32               teMetric;
            Types::uint32               teColorMask;
            Types::uint8                teProtectionTypeMask;
            Types::uint32               teMaxBw;
            Types::uint32               teMaxResvBw;
      };

      struct freeCTPEntry {
            Types::uint8                sigType;
            Types::uint32               ctps; // 24 lsbs
      };
      typedef sequence<freeCTPEntry>          freeCTPSeq;

      struct teLinkTdmParams {
            Types::uint32               hoMuxCapMask;
            Types::uint32               loMuxCapMask;
            Types::uint32               transparencyMask;
            Types::uint32               blsrRingId;
      };

      struct teLinkLscG709Params {
            Types::uint32               odukMuxCapMask;
      };

      struct teLinkWdmAmplifierEntry {
            Types::uint32               gain;
            Types::uint32               noiseFigure;
      };
      typedef sequence<teLinkWdmAmplifierEntry>     amplifiersSeq;

      struct teLinkLscWdmParams {
            Types::uint32               dispersionPMD;
            Types::uint32               spanLength;
            amplifiersSeq               amplifiers;
      };

      typedef sequence<Types::uint8>          bitmapSeq; // numLambdas/32 +1

      struct teLinkWdmLambdasBitmap {
            // in ITU DWDM format
            Types::uint32               baseLambda; // ITU DWDM format
            Types::uint32               numLambdas; // 16 lsbs
            bitmapSeq                   bitmap;
      };
```

Appendix B Automatic FSM skeleton generation

This tool provides a framework for the human-readable definition of Finite State Machines (FSM) and automatic generation of the skeleton code for its implementation. The tool also provides a Graphviz .dot output file, which can be used to produce a graphical representation of FSM states and transitions events to improve readability. Some of the G^2MPLS FSMs have been briefly described in the sections above.

The FSM automatic generation tool is a framework based on three main parts:

- Configuration file: describes states, events and transitions of the FSM.
- Template file: the core of the generation tool, it is responsible of reading the configuration file and generating the skeleton code according to design pattern strategy for *state* pattern.
- Generated code: both core generated files, that must not be modified and the partial skeleton files, where users must add the specific code for state transitions.

The following sections describe a case-study to generate a really simple FSM, made of four states and three events.

B.1 Configuration file

If the *graphviz-file* is specified the tool provides a Graphviz .dot file to generate a traditional graphical representation of the FSM. If the *start-state* is not set, the first state is the beginning state.

The event description allows both the definition of simple *root_events* (that can be mapped 1:1 with the derived ones) and complex *root_events* that can be split into derived ones at run time, according to specific transitions code. In the last case a support virtual state is created.

```
#
#  FSM definition
#
#    st/ev    event1   event2   event3
#
#    stateI   state1   state2   state3
#    state1   state1     -        -
#    state2     -      state2     -
#    state3     -        -      state3

{ FSM }

name = TEST_FSM
definition-file = test.def
graphviz-file = test.dot
include-name = test.h
start-state = stateI

#
# Events
#

{ Events }

root_event123 = event1,  event2,  event3


#
# States
#

{ States }

State = stateI
      event1   -> state1
      event2   -> state2
      event3   -> state3

State = state1
      event1   -> state1

State = state2
      event2   -> state2

State = state3
      event3   -> state3
```

B.2 Template file

Template file is the core file in charge of generating the skeleton of FSM according to *State* pattern.

The *State* pattern is a solution to the problem of how to make behaviour depend on state. The main steps are:

- Define a "context" class to present a single interface to the outside world.
- Define a State abstract base class that holds all the transition of the state machine.
- Represent the different "states" of the state machine as derived classes of the State base class.
- Define state-specific behaviour in the appropriate State derived classes.
- Maintain a pointer to the current "state" in the "context" class.
- The “context” class does nothing more that immediately delegate to the current “state”.
- To change the state of the state machine, change the current "state" pointer.

The State pattern does not specify where the state transitions will be defined. The choices are two: the "context" object, or each individual State derived class. The advantage of the latter option is ease of adding new State derived classes. The disadvantage is each State derived class has knowledge of (coupling to) its siblings, which introduces dependencies between subclasses.

The FSM skeleton generation tool uses the first approach, storing all the transitions in a *Matrix* template class shown in Code 16-1.

```
#ifndef FSMGEN_UTILITY
#define FSMGEN_UTILITY

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <list>
#include <stdlib.h>

/******************************************************************************/
/*                           Utility - Matrix class                          */
/******************************************************************************/

template <class ROW, class COLUMN, class DATA>
class Matrix {
public:
       Matrix() { }
       ~Matrix() { }

       //friend class Fsm;

       // Copy operator
       Matrix(const Matrix<ROW, COLUMN, DATA>& m) { matrix_ = m.matrix_; }

       //
       // Type definitions
       //
       typedef ROW *          rowIter;
       typedef const ROW *    const_rowIter;
       typedef COLUMN *       colIter;
       typedef const COLUMN * const_colIter;
       typedef DATA *         dataIter;
       typedef const DATA *   const_dataIter;
```

```
        //
        // Iterator support
        //
        rowIter begin(void);
        const_rowIter begin(void) const;
        rowIter end(void);
        const_rowIter end(void) const;
        rowIter next(rowIter rowIt);
        const_rowIter next(const_rowIter rowIt) const;
        colIter begin(rowIter rowIt);
        const_colIter begin(const_rowIter rowIt) const;
        colIter end(rowIter rowIt);
        const_colIter end(const_rowIter rowIt) const;
        colIter next(rowIter rowIt, colIter colIt);
        const_colIter next(const_rowIter rowIt, const_colIter colIt) const;


        // Return the number of deleted cells within the row
        size_t removeRow(const ROW& row);

        // Return the number of deleted cells within the column
        size_t removeCol(const COLUMN& column);

        // Return the number (1 or 0) of deleted data for this pair row/column
        size_t remove(const ROW& row, const COLUMN& column);

        // Return the number of deleted data
        size_t remove(const DATA& data);

        bool remove(dataIter dIt);

        void insert(const ROW& row, const COLUMN& column, const DATA& data);

        dataIter find(const ROW& row, const COLUMN& column);

        // Return the number of data
        size_t size(void) const;

        bool empty(void) const;

        // Assignment operator
        Matrix<ROW, COLUMN, DATA>& operator=(const Matrix<ROW,COLUMN,DATA>& o);

        std::map<COLUMN, DATA>& operator[](const ROW s);

        friend std::ostream& operator<<(std::ostream& s, const Matrix& m);

private:
        std::map< ROW, std::map<COLUMN, DATA> > matrix_;
};
```

Code 16-1: Matrix class

A table-driven approach to design finite state machines is a good choice to specify state transitions but, in this case, it is more difficult to add actions that come with the state transitions.

B.3 Generated code

There are two kind of generated files:

- the core ones, shown in Code 16-2, and

```
the skeleton to be filled in, shown in #ifndef TEST_H
#define TEST_H

#include <iostream>
#include <stdio.h>
#include "test_gen.h"

class state1_i : public fsm::base_TEST_FSM::state1
{
 public:
	fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
	void after_event1_from_virt_state1(void * context);
	void after_event1_from_virt_stateI(void * context);
};

class stateI_i : public fsm::base_TEST_FSM::stateI
{
 public:
	fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
};

class state3_i : public fsm::base_TEST_FSM::state3
{
 public:
	void after_event3_from_virt_stateI(void * context);
	fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
	void after_event3_from_virt_state3(void * context);
};

class state2_i : public fsm::base_TEST_FSM::state2
{
 public:
	void after_event2_from_virt_stateI(void * context);
	fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
	void after_event2_from_virt_state2(void * context);
};

class virt_state1_i : public fsm::base_TEST_FSM::virt_state1
{
 public:
	void after_root_event123_from_state1(void * context);
	bool event1(void* context);
};

class virt_stateI_i : public fsm::base_TEST_FSM::virt_stateI
{
 public:
	void after_root_event123_from_stateI(void * context);
	bool event1(void* context);
```

```
	bool event3(void* context);
	bool event2(void* context);
};

class virt_state3_i : public fsm::base_TEST_FSM::virt_state3
{
 public:
	void after_root_event123_from_state3(void * context);
	bool event3(void* context);
};

class virt_state2_i : public fsm::base_TEST_FSM::virt_state2
{
 public:
	void after_root_event123_from_state2(void * context);
	bool event2(void* context);
};

#endif // TEST_GEN
```

- Code 16-3.

```
namespace fsm {


#ifndef NAMESPACE_BASE_TEST_FSM
#define NAMESPACE_BASE_TEST_FSM

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
#include <list>
#include <stdlib.h>

/******************************************************************************/
/*                              Finite State Machine                          */
/******************************************************************************/
	/**************************************/
	/*    Finite State Machine - Core    */
	/**************************************/
	namespace base_TEST_FSM {

		enum nextEvFor_root_event123_t {
			TEST_FSM_from_root_event123_to_InvalidEvent = 0,
			TEST_FSM_from_root_event123_to_event1,
			TEST_FSM_from_root_event123_to_event2,
			TEST_FSM_from_root_event123_to_event3,
		};


		class State {
		public:
			State(std::string name = "Base state");
			virtual ~State(void);
```

```
        std::string name(void);

        // On event
        virtual bool event1(void * context);
        virtual bool event2(void * context);
        virtual bool event3(void * context);
        virtual nextEvFor_root_event123_t root_event123(void * context) {


        // After event from state
        virtual void after_root_event123_from_state1(void * context);
        virtual void after_event1_from_virt_state1(void * context);
        virtual void after_root_event123_from_stateI(void * context);
        virtual void after_event1_from_virt_stateI(void * context);
        virtual void after_event3_from_virt_stateI(void * context);
        virtual void after_event2_from_virt_stateI(void * context);
        virtual void after_root_event123_from_state3(void * context);
        virtual void after_event3_from_virt_state3(void * context);
        virtual void after_root_event123_from_state2(void * context);
        virtual void after_event2_from_virt_state2(void * context);

private:
        std::string name_;
};

/*
 *  Classes that MUST be derived!!! - START
 */
class state1_i;
class stateI_i;
class state3_i;
class state2_i;

class virt_state1_i;
class virt_stateI_i;
class virt_state3_i;
class virt_state2_i;


class state1 : public State {
public:
        state1() :
                State(std::string("state1"));
        virtual ~state1();

        virtual fsm::base_TEST_FSM::nextEvFor_root_event123_t
                                     root_event123(void* context) = 0;

        virtual void after_event1_from_virt_state1(void * context) = 0;

        virtual void after_event1_from_virt_stateI(void * context) = 0;
};

class stateI : public State {
public:
        stateI() :
                State(std::string("stateI"));
```

```
        virtual ~stateI();

        virtual fsm::base_TEST_FSM::nextEvFor_root_event123_t
                                root_event123(void* context) = 0;
    };

    class state3 : public State {
    public:
        state3() :
            State(std::string("state3"));
        virtual ~state3();

        virtual void after_event3_from_virt_stateI(void * context) = 0;

        virtual fsm::base_TEST_FSM::nextEvFor_root_event123_t
                                root_event123(void* context) = 0;

        virtual void after_event3_from_virt_state3(void * context) = 0;

    };

    class state2 : public State {
    public:
        state2() :
            State(std::string("state2"));
        virtual ~state2();

        virtual void after_event2_from_virt_stateI(void * context) = 0;

        virtual fsm::base_TEST_FSM::nextEvFor_root_event123_t
                                root_event123(void* context) = 0;

        virtual void after_event2_from_virt_state2(void * context) = 0;
    };

    class virt_state1 : public State {
    public:
        virt_state1() :
            State(std::string("virt_state1"));
        virtual ~virt_state1();

        virtual void after_root_event123_from_state1(void * context) = 0;

        virtual bool event1(void* context) = 0;
    };

    class virt_stateI : public State {
    public:
        virt_stateI() :
            State(std::string("virt_stateI"));
        virtual ~virt_stateI();

        virtual void after_root_event123_from_stateI(void * context) = 0;

        virtual bool event1(void* context) = 0;

        virtual bool event3(void* context) = 0;
```

```
        virtual bool event2(void* context) = 0;
};

class virt_state3 : public State {
public:
        virt_state3() :
                State(std::string("virt_state3"));
        virtual ~virt_state3();

        virtual void after_root_event123_from_state3(void * context) = 0;

        virtual bool event3(void* context) = 0;
};

class virt_state2 : public State {
public:
        virt_state2() :
                State(std::string("virt_state2"));
        virtual ~virt_state2();

        virtual void after_root_event123_from_state2(void * context) = 0;

        virtual bool event2(void* context) = 0;
};

/*
 *  Classes that MUST be derived!!! - END
 */

class BaseFSM {
public:
        enum traceLevel_t {
                TRACE_DBG = 0,
                TRACE_LOG,
                TRACE_INF,
                TRACE_WRN,
                TRACE_ERR
        };

        BaseFSM(traceLevel_t level = TRACE_DBG);
        virtual ~BaseFSM(void);

        std::string name(void);
        traceLevel_t traceLevel(void);

        void traceLevel(traceLevel_t level) { level_ = level; }

        void dbg(std::string text);
        void log(std::string text);
        void inf(std::string text);
        void wrn(std::string text)
        void err(std::string text);

private:
        traceLevel_t level_;
protected:
        std::string  name_;
};
```

```
/*
 *  Class for checking FSM integrity
 */
class GenericFSM : public BaseFSM {
public:
      GenericFSM(traceLevel_t level = TRACE_DBG);
      virtual ~GenericFSM(void) { }

      bool startModify(void);
      bool endModify(void);

      typedef void (* callback_t) (std::string from_state,
                                   std::string to_state,
                                   std::string on_event,
                                   void *      context);


      // States
      bool addState(std::string state);
      bool remState(std::string state);

      // Events
      bool addEvent(std::string event);
      bool remEvent(std::string event);

      // Transitions
      bool addTransition(std::string from,
                         std::string to,
                         std::string event);
      bool remTransition(std::string from,
                         std::string to,
                         std::string event);

      // General
      bool setStartState(std::string state);

private:
      bool check(void);

      struct state_data_t {
             callback_t  pre;
             callback_t  post;
             callback_t  in;
      };

      Matrix<std::string,
             std::string,
             std::string>                 transitions_;
      std::map<callback_t, void *>        contexts_;
      std::map<std::string, state_data_t> states_;
      std::list<std::string>              events_;
      bool                                changeInProgress_;
      std::string                         startState_;
};

class Fsm : public GenericFSM {
public:
```

```
                    /*  add/rem of states/events/callback for
                     *  checking FSM consistency
                     */
                    Fsm(traceLevel_t level = TRACE_DBG)
                          throw(std::string);
                    virtual ~Fsm(void);

                    friend std::ostream& operator<<(std::ostream&  s,
                                              const Fsm& f);


                    bool event1(void * context);
                    bool event2(void * context);
                    bool event3(void * context);

                    nextEvFor_root_event123_t root_event123(void * context);



                    State * currentState(void);
                    bool    go2prevState(void);

              private:
                    enum states_t {
                          TEST_FSM_state1,
                          TEST_FSM_stateI,
                          TEST_FSM_state3,
                          TEST_FSM_state2,
                          TEST_FSM_virt_state1,
                          TEST_FSM_virt_stateI,
                          TEST_FSM_virt_state3,
                          TEST_FSM_virt_state2,

                    };

                    enum events_t {
                          TEST_FSM_event1,
                          TEST_FSM_event2,
                          TEST_FSM_event3,
                          TEST_FSM_root_event123,

                    };

                    friend std::ostream& operator<<(std::ostream&   s,
                                              const states_t& st);

                    friend std::ostream& operator<<(std::ostream&   s,
                                              const events_t& ev);

                    states_t                             currentState_;
                    states_t                             prevState_;
                    std::map<states_t, State *>          states_;
                    Matrix<states_t, events_t, states_t> nextState_;
              };
        }
#endif // NAMESPACE_TEST_FSM
```

```
        /************************************/
        /*   Finite State Machine - Wrapper  */
        /************************************/



#ifndef NAMESPACE_TEST_FSM
#define NAMESPACE_TEST_FSM

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string>
#include <map>
#include <list>
#include <iostream>

        namespace TEST_FSM {

                class virtFsm {
                public:
                        virtFsm(base_TEST_FSM::BaseFSM::traceLevel_t
                                                level = base_TEST_FSM::BaseFSM::TRACE_DBG)
                                throw(std::string);
                        virtual ~virtFsm(void);

                        friend std::ostream& operator<<(std::ostream&  s,
                                                        const virtFsm& f);

                        enum root_events_t {
                                TEST_FSM_root_event123,
                        };

                        void post(root_events_t ev, void * context, bool enqueue = false);
                        std::string currentState(void);

                private:
                        void runPendingWork(void);

                        void root_event123(void * context);

                        typedef struct {
                                root_events_t ev;
                                void *        context;
                        } data_event_t;

                        base_TEST_FSM::Fsm *       fsm_;
                        std::list<data_event_t *>  events_;
                };

        }
#endif // NAMESPACE_TEST_FSM

}

#endif // FSMGEN_H
```

Code 16-2: Core generated file.

```
#ifndef TEST_H
#define TEST_H

#include <iostream>
#include <stdio.h>
#include "test_gen.h"

class state1_i : public fsm::base_TEST_FSM::state1
{
 public:
       fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
       void after_event1_from_virt_state1(void * context);
       void after_event1_from_virt_stateI(void * context);
};

class stateI_i : public fsm::base_TEST_FSM::stateI
{
 public:
       fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
};

class state3_i : public fsm::base_TEST_FSM::state3
{
 public:
       void after_event3_from_virt_stateI(void * context);
       fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
       void after_event3_from_virt_state3(void * context);
};

class state2_i : public fsm::base_TEST_FSM::state2
{
 public:
       void after_event2_from_virt_stateI(void * context);
       fsm::base_TEST_FSM::nextEvFor_root_event123_t root_event123(void* context);
       void after_event2_from_virt_state2(void * context);
};

class virt_state1_i : public fsm::base_TEST_FSM::virt_state1
{
 public:
       void after_root_event123_from_state1(void * context);
       bool event1(void* context);
};

class virt_stateI_i : public fsm::base_TEST_FSM::virt_stateI
{
 public:
       void after_root_event123_from_stateI(void * context);
       bool event1(void* context);
       bool event3(void* context);
       bool event2(void* context);
};

class virt_state3_i : public fsm::base_TEST_FSM::virt_state3
{
```

```
 public:
      void after_root_event123_from_state3(void * context);
      bool event3(void* context);
};

class virt_state2_i : public fsm::base_TEST_FSM::virt_state2
{
 public:
      void after_root_event123_from_state2(void * context);
      bool event2(void* context);
};

#endif // TEST_GEN
```

Code 16-3: Skeleton generated file.

The .dot file is shown in Code 16-4 and can be used to have a canonical graphical representation of the FSM, as shown in Figure 16-1.

```
digraph finite_state_machine {
    ordering=in;
    concentrate=true;
    rankdir=TB;
    ranksep=1.25;
    node[height = 1.3];
    node [fontsize=12 fixedsize=true shape=circle color=lightsteelblue3 style=filled];
    edge [fontsize=9];

    state1 -> state1 [ label = "event1" ];
    stateI -> state1 [ label = "event1" ];
    stateI -> state3 [ label = "event3" ];
    stateI -> state2 [ label = "event2" ];
    state3 -> state3 [ label = "event3" ];
    state2 -> state2 [ label = "event2" ];
}
```

Code 16-4: *test.dot* graphviz file.

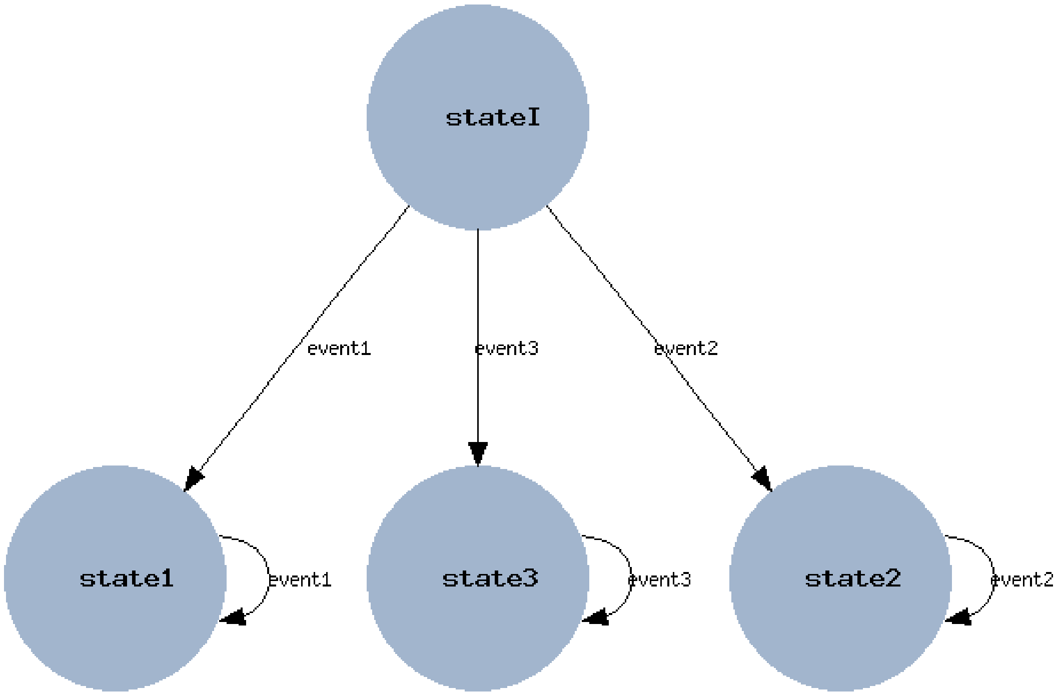

Figure 16-1: Test FSM.

Appendix C TNRC Specific Part for ADVA FSP 3000RE-II

C.1 API Data structures

This section specifies the TNRC_SP API for operation on LSC ADVA FSP 3000RE-II device.

```
typedef unsigned int tnrcsp_lsc_evmask_t; /* values TBD */
typedef unsigned short tnrcsp_lsc_eqplane_t;

typedef enum {
       TNRCSP_LISTTYPE_UNSPECIFIED,
       TNRCSP_LISTTYPE_RESOURCES
} tnrcsp_list_type_t;

typedef enum {
       TNRC_SP_LSC_OLD,
       TNRC_SP_LSC_XCVR
} tnrcsp_lsc_eqtype_t;

typedef enum {
       TNRCSP_LSC_XCSTATE_RESERVED,
       TNRCSP_LSC_XCSTATE_ACTIVE,
       TNRCSP_LSC_XCSTATE_FAILED
} tnrcsp_lsc_xc_state_t;

typedef struct {
       tnrc_portid_t             portid;
       label_t                   labelid;
       tnrc_operstate_t          oper_state;
       tnrc_adminstate_t         admin_state;
       tnrcsp_lsc_evmask_t events;
} tnrcsp_lsc_event_t;

typedef struct {
       tnrc_portid_t             portid;
} tnrcsp_lsc_resource_id_t;

typedef struct {
       tnrc_operstate_t          oper_state;
```

```
        tnrc_adminstate_t          admin_state;
        tnrcsp_lsc_evmask_t      last_event;
        tnrcsp_lsc_eqtype_t      equip_type;
        tnrcsp_lsc_eqplane_t     equip_plane;
} tnrcsp_lsc_resource_detail_t;
```

Note: SLIST_HDR is an header implementing a simple list, and contains the pointers to the next element in the list. TBD immediately in a separate document about global design specifications.

C.2 Summary of TNRC_SP LSC ADVA API functions

- tnrcsp_lsc_advafsp_make_xc
- tnrcsp_lsc_advafsp_destroy_xc
- tnrcsp_lsc_advafsp_reserve_xc
- tnrcsp_lsc_advafsp_unreserve_xc
- tnrcsp_lsc_advafsp_register_async_cb
- tnrcsp_lsc_advafsp_get_resource_list
- tnrcsp_lsc_advafsp_get_resource_details
- tnrcsp_lsp_advafsp_get_labels

C.3 Detailed specification of TNRC_SP LSC ADVA API functions

The following functions should be included in the API:

XC creation	tnrcsp_result_t **tnrcsp_lsc_advafsp_make_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t virtual, tnrc_boolean_t activate, tnrcsp_response_cb_t response_cb, void *response_cxt, tnrcsp_notification_cb_t async_cb, void *async_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
virtual	In	non-physical XC; for future usage (e.g. adoption of existing XCs)
activate	In	turn a couple of reserved ports into a XC
response_cb	In	pseudo-synchronous callback function provided by the TNRC

		AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
async_cb	In	asynchronous notification function provided by the TNRC AP, to be called whenever something asyn occurs on the XC or some of its elements
async_cxt	In	asynchronous context provided by the TNRC AP, to be returned in the async notification callback
Description		
This function will create the XC, with the following behaviour: ▪ It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device ▪ Later, when the XC has been completed or failed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation, ▪ XC creation is composed from few TL1 commands sequence, ▪ If XC creation failed, all resources are released, and device should be in the same state as before XC creation, ▪ Correctness of XC creation is checked at the end of action, ▪ XC activation (activate=True) will success only if there was XC reservation called before, ▪ Any future event related to the XC or one of its components (e.g. ports) will be reported to the TNRC AP with the asynchronous callback. ADVA uses TL1 autonomous messages to inform about events and alarms.		
Used TL1 commands		
ASC-CHANNEL	Assign channel	Normal situation
RST-CHANNEL	Restore channel	Normal situation if XC activation
RTRV-CHANNEL	Retrieve channel	Normal situation
RMV-CHANNEL	Remove channel	Exceptional situation if XC activation
DLT-CHANNEL	Delete channel	Exceptional situation
Synchronous function results		
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid	
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in	
TNRCSP_RESULT_PARMERROR	Not valid arguments	
Pseudo-synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	
TNRCSP_RESULT_PARMERROR	Wrong argument value	
TNRCSP_RESULT_BUSYRESOURCES	Resources not available	
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure	
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid and resources are available)	

XC removal	tnrcsp_result_t **tnrcsp_lsc_advafsp_destroy_xc**(tnrcsp_handle_t *handlep,

	tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t virtual, tnrc_boolean_t deactivate, tnrcsp_response_cb_t response_cb, void *response_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
virtual	In	non-physical XC removal; for future usage (e.g. release of existing XCs)
deactivate	In	turn an XC into a couple of reserved ports
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
Description		
This function will destroy the XC, with the following behaviour: ▪ It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device, ▪ Later, when the XC removal has been completed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation, ▪ XC deletion is composed from few TL1 commands sequence, ▪ In case of any unsuccessful processing of command the release of resources is continued, ▪ XC deactivation (deactivate=True) will success only if there was active XC, ▪ Correctness of XC deletion is checked at the end of action.		
Used TL1 commands		
RMV-CHANNEL	Remove channel	Normal situation
DLT-CHANNEL	Delete channel	Normal situation
RTRV-CHANNEL	Retrieve channel	Normal situation
Synchronous function results		
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid	
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in	
TNRCSP_RESULT_PARMERROR	Not valid arguments	
Pseudo-synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	
TNRCSP_RESULT_PARMERROR	Wrong argument value	
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure	
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid)	

XC reservation	tnrcsp_result_t **tnrcsp_lsc_advafsp_reserve_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction,

	tnrc_boolean_t virtual, tnrcsp_response_cb_t response_cb, void *response_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
virtual	In	non-physical XC; for future usage (e.g. adoption of existing XCs)
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback

Description

This function will reserve the XC, with the following behaviour:

- It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device,
- Later, when the XC reservation has been completed or failed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation,
- XC reservation is composed from few TL1 commands sequence,
- If XC reservation failed, all resources are released, and device should be in the same state as before XC creation,
- Correctness of XC reservation is checked at the end of action.

Used TL1 commands		
ASC-CHANNEL	Assign channel	Normal situation
RTRV-CHANNEL	Retrieve channel	Normal situation
DLT-CHANNEL	Delete channel	Exceptional situation

Synchronous function results	
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in
TNRCSP_RESULT_PARMERROR	Not valid arguments

Pseudo-synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value
TNRCSP_RESULT_BUSYRESOURCES	Resources not available
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid and resources are available)

XC unreservation	tnrcsp_result_t **tnrcsp_lsc_advafsp_unreserve_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t virtual, tnrcsp_response_cb_t response_cb, void

	*response_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
virtual	In	non-physical XC removal; for future usage (e.g. release of existing XCs)
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
Description		
This function will unreserve the XC, with the following behaviour: ▪ It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device, ▪ Later, when the XC removal has been completed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation, ▪ XC unreservation is composed from few TL1 commands sequence, ▪ In case of any unsuccessful processing of command the release of resources is continued, ▪ XC unreservation will success only if XC is not active, ▪ Correctness of XC unreservation is checked at the end of action.		
Used TL1 commands		
DLT-CHANNEL	Delete channel	Normal situation
RTRV-CHANNEL	Retrieve channel	Normal situation
Synchronous function results		
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid	
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in	
TNRCSP_RESULT_PARMERROR	Not valid arguments	
Pseudo-synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	
TNRCSP_RESULT_PARMERROR	Wrong argument value	
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure	
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid)	

Register events notification	tnrcsp_result_t **tnrcsp_lsc_advafsp_register_async_cb**(tnrcsp_lsc_event_t *events, unsigned int num)	
Parameters		
events	In	List of events to be notified to the TNRC AP; each event item focuses on a port and reports about states (operational, administrative) and occurred events (using a bitmask)

Num	In	number of events

Description
This function will register events to be notified to TNRC AP; Notification mechanism is invoked asynchronously by the TNRC SP when: • TL1 autonomous alarm notification appear, • operation state occur, • administration state occur. The administrative and operational status are periodically polled and states are compared with registered values. This function doesn't use any TL1 command.
Synchronous function results

TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_PARMERROR	Wrong argument value
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure

Fetching of resources list	tnrcsp_result_t **tnrcsp_lsc_advafsp_get_resource_list**(tnrcsp_lsc_resource_id_t **resource_listp, unsigned int* num)

Parameters		
resource_listp	Out	to be returned as pointer to the list of resource ids
num	Out	number of returned resource ids

Description

This function allows to fetch the list of underlying resources. Each resource will be assigned an id by the TNRC_SP. The resource identifier is composed from Access Identifier code (AID) of card.
For example: AID = 1-1-13 ("bay-shelve-slot") then port id = 010113 (each value is represented by 2 digits). This transformation generates unique ids and it is easily reversible.
Example of available resources in the one of PSNC's devices named 'Cracow':

Card description	AID of card	Port ID
OLD in Plane 0	1-1-8	10108
XCVR/XPDR *in Plane 0*	1-1-9	10109
OLD in Plane 1	1-1-13	10113
XCVR/XPDR *in Plane 1*	1-1-14	10114
XCVR/XPDR *in Plane 1*	1-1-16	10116

This function doesn't send any TL1 command. It used gathered information by periodically send RTRV-EQPT-ALL command.

Synchronous function results

TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost

Fetching of details about a specific resource	tnrcsp_result_t **tnrcsp_lsc_advalsp_get_resource_detail**(tnrcsp_lsc_resource_id_t resource_id, tnrcsp_lsc_resource_detail_t *resource_detailp)

Parameters		
resource_id	In	identifier of the resource whose details are fetched
resource_detailp	Out	to be returned as pointer to the structure of resource details

Description

This function allows to fetch the details of a specific resource. The details contains information about:

- equipment plane (plane 0, plane 1),
- equipment type (OLD, XCVR/XPDR),
- current administrative state (disabled, enabled),
- current operational state (disabled, enabled),
- last event.

ADVA Add/Drop Multiplexer architecture information (wavelength, equipment plane and type) are needed because not all couple of resources can be crossconnected. A crossconnection is possible only for scenarios presented in the table bellow:

Connection type	Ingress resource		Egress resource	
	Equipment type	Equipment plane	Equipment type	Equipment plane
Pass-through	OLD	0	OLD	1
Drop	OLD	0	XCVR/XPDR	0
Pass-through	OLD	1	OLD	0
Drop	OLD	1	XCVR/XPDR	1
Add	XCVR/XPDR	0	OLD	0
Add	XCVR/XPDR	1	OLD	1

There is also second condition for crossconnection possibility – labels (wavelengths) for both resources must be the same.

Administrative state depends on PrimaryState returned by device:

Administrative State	PrimaryState	Description
TNRC_ADMINSTATE_ENABLED	IS	In-service
	IS-ANR	In-service, abnormal
	IS-ANRST	In-service, abnormal and restricted
	IS-NR	In-service, normal
	IS-RST	In-service, restricted
	MA	Management
TNRC_ADMINSTATE_DISABLED	OSS	Out-of-service
	OSS-AU	Out-of-service, autonomous
	OOS-AUMA	Out-of-service, autonomous and management
	OOS-AURST	Out-of-service, autonomous and restricted
	OOS-MA	Out-of-service, management
	OOS-MAANR	Out-of-service, management and abnormal

Operational state depends on SecondaryState returned by device:

Administrative State	SecondaryState	Description
TNRC_OPERSTATE_ENABLED	ACT	Active
TNRC_OPERSTATE_DISABLED	ASWDL	Automatic Software Download
	DGN	Diagnostic
	DSBLD	Data Sync

	FLT	Fault
	LPBK-FAC	Loopback Facility
	LPBK-TERM	Loopback Terminal
	MISM	Mismatched
	NALM	No Alarm
	PRBS	PRBS test
	SGEO	Supporting entity outage
	STBY	Supporting entity outage
	SWDL	Software download
	TCAI	TCA Inhibited
	TUNE	Indicates laser is in the process of turning on
	UAS	Unassigned
	UEQ	Unequipped

Last event present last non-alarm or alarm condition. Alarm values are presented in the error table section of annex. Non-alarm events are not listed yet (lack in documentation).

This function doesn't send any TL1 command. It used gathered information by periodically send RTRV-EQPT-ALL command.

Synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value

Fetching of crossconnections list	tnrcsp_result_t **tnrcsp_lsc_advalsp_get_label_list**(tnrcsp_resource_id_t resource_id, label_t** label_listp, unsigned int* num)	
Parameters		
resource_id	In	identifier of the resource whose labels are fetched
label_listp	Out	to be returned as pointer to the list of labels
num	Out	number of returned resource ids
Description		

ADVA TL1 commands use Channel ID for any operation. Channel ID and corresponding wavelength is presented in the table:

Channel ID	Wavelength	Channel ID	Wavelength	Channel ID	Wavelength
20	1561.42 nm	34	1550.12 nm	48	1538.98 nm
21	1560.61 nm	35	1549.32 nm	49	1538.19 nm
22	1559.79 nm	36	1548.52 nm	50	1537.40 nm
23	1558.98 nm	37	1547.72 nm	51	1536.61 nm
24	1558.17 nm	38	1546.92 nm	52	1535.82 nm
25	1557.36 nm	39	1546.12 nm	53	1535.04 nm
26	1556.56 nm	40	1545.32 nm	54	1534.25 nm
27	1555.75 nm	41	1544.53 nm	55	1533.47 nm
28	1554.94 nm	42	1543.73 nm	56	1532.68 nm
29	1554.13 nm	43	1542.94 nm	57	1531.90 nm
30	1553.33 nm	44	1542.14 nm	58	1531.12 nm

31	1552.52 nm	45	1541.35 nm	59	1530.33 nm
32	1551.72 nm	46	1540.56 nm		
33	1550.92 nm	47	1539.77 nm		

However, Channel ID is specific identifier for ADVA device only. To be equipment independent ADVA TNRC SP API uses generic label format introduced by [draft-otani-labels]. Label value corresponding to channel ids is presented in the table:

Channel ID	Label value	Channel ID	Label value	Channel ID	Label value
20	687865867	34	671088643	48	671088657
21	687865866	35	671088644	49	671088658
22	687865865	36	671088645	50	671088659
23	687865864	37	671088646	51	671088660
24	687865863	38	671088647	52	671088661
25	687865862	39	671088648	53	671088662
26	687865861	40	671088649	54	671088663
27	687865860	41	671088650	55	671088664
28	687865859	42	671088651	56	671088665
29	687865858	43	671088652	57	671088666
30	687865857	44	671088653	58	671088667
31	671088640	45	671088654	59	671088668
32	671088641	46	671088655		
33	671088642	47	671088656		

This function doesn't send any TL1 command. It used gathered information by periodically send RTRV-EQPT-ALL command.

Synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value

C.4 ADVA FSP 3000RE-II device

C.4.1 Overview

The FSP 3000RE-II is Reconfigurable Optical Add/Drop Multiplexer (ROADM). The FSP 3000RE-II offers scalable means to support a broad range of services. On the line-side, they can receive and transmit up to 40 protected wavelengths. On the tributary-side, they can drop up to four line-protected wavelengths, eight line-unprotected wavelengths, or a combination of both. The interfaces of these tributaries range from SONET/ SDH, to Gigabit Ethernet, to reshaping, regenerating, and retiming (3R) transparent Service Interface Module (SIM). The FSP 3000RE-I/FSP 3000RE-II shelf consists of a combination of optical line drivers (OLDs), transponders (XPDRs), transceivers (XCVRs), SIMs, optical protection switches (OPS), Transponder Protection Modules (XPMs), shelf processors (SPs), and other circuit packs on a chassis/ backplane. The ADVA FSP 3000RE-II is shown on Figure 16-2.

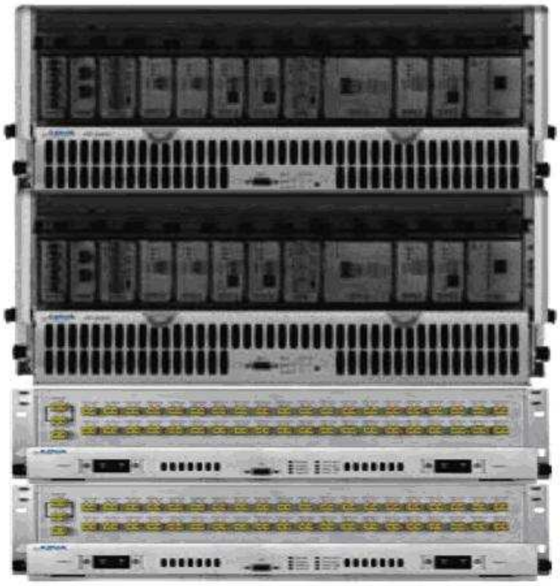

Figure 16-2: ADVA FSP 3000RE-II device.

From perspective of Phosphorus project, the more interesting is optical device architecture presented on Figure 16-3 . It is composed of two planes containing one OLD, one or few XCVRs and one ROADM filter. The DWDM fiber is connected always to OLD. Currently all PSNC ADVA have 40 channels in DWDM link (100GHz spacing between channels, wavelength from 192.00 to 195.90 THz). There are 2 planes so device can work with 2 DWDM links. There can be configured crossconnection for each lambda that enables light passing between OLDs. The other possibility is Add/Drop configuration at each eROADM separately that enables lambda dropping or adding to XCVRs. Each plane can have different number of transceivers (from one to four XCVRs).

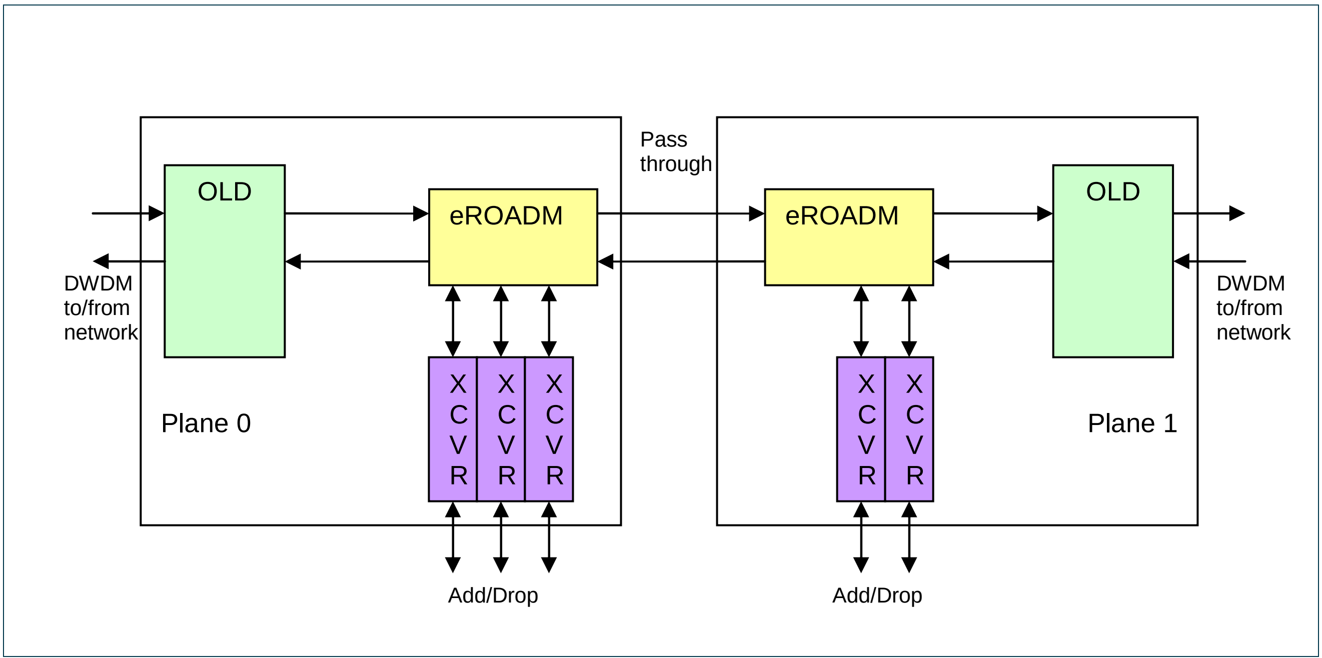

Figure 16-3: ADVA FSP 3000RE-II architecture.

The ADVA TNRC SP takes care of configuring filters in eROADMs by PASS-THRU, ADD or DROP operations. Example of configured connections is shown of Figure 16-4, where Plane 0 has 3 transceivers (channels: 34,41,59), Plane 1 has 2 transceivers (channels: 50, 59). There are configured in the way:

- 3 pass-through connections for channel 1 (Plane 0<->1) and channel 2 (Plane 0->1),
- 3 drop connections for channel 34 and 59 in Plane 0 and channel 59 in Plane 1,
- 3 add connections for channels 34 and 41 in Plane 0 and channel 59 in Plane 1.

Each connection is unidirectional. To configure bidirectional connection there is a need to configure two connection for both direction in independently. From configuration point of view, OLD equipments have 2 port: RX and TX. XCVR has always one bidirectional port: RX/TX.

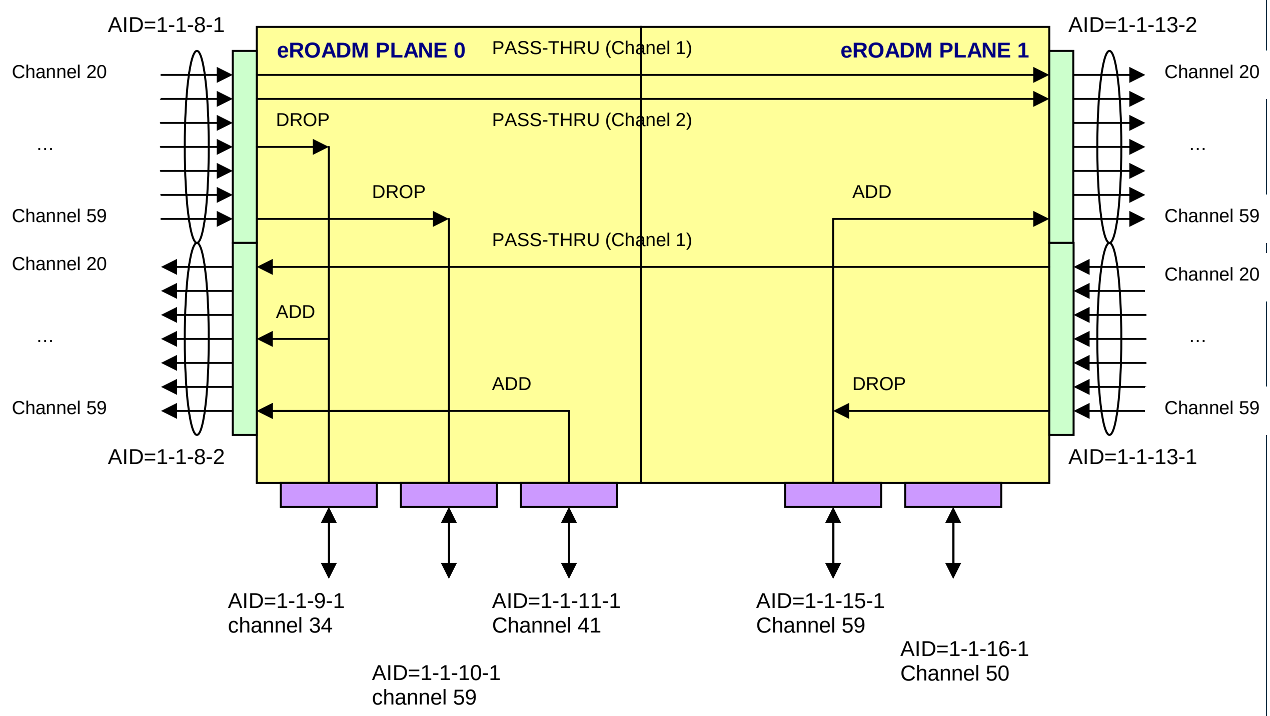

Figure 16-4: ADVA FSP 3000RE-II eROADM connections configuration (AID are “bay-shelve-slot-port”).

C.4.2 Implementation details

For device configuration it is used TL1-RAW (port 3082) or TL1-TELNET (port 3083). The ADVA SP opens one permanent TCP session and sends TL1 login command.

Crossconnect operations are not fast. They need from one to few seconds to complete, because of channel equalization process. One operation takes much more time to be completed. It is bidirectional xc activation which is also part of make_xc operation. Because of long XC operation time, all XC operation are processed in non-blocking way. The rest of functions are blocking functions.

The list of operation times is listed in the table:

Operation	Type	Operation time
login	-	1-5 sec
make_xc	unidirectional	1-5 sec
	bidirectional	15-21 sec
destroy_xc	unidirectional	2-10 sec

	bidirectional	4-11 sec
reserve_xc	unidirectional	1-2 sec
	bidirectional	1-2 sec
unreserve_xc	unidirectional	1-2 sec
	bidirectional	1-3 sec
activate_xc	unidirectional	1-4 sec
	bidirectional	16-21 sec
deativate_xc	unidirectional	1-2 sec
	bidirectional	2-3 sec
register_async_vb	-	<< 1sec
get_resource_list	-	<< 1sec
get_resource_detail	-	<< 1sec
get_xc_list	-	<< 1sec
flush_list	-	<< 1 sec

The TNRC ADVA SP is composed of several cooperative threads:

Thread name	Duration	Count	Description
TNRCSP_ADVA	permanent	1	create TL1 listing thread and check connection to device status if there is no TCP session then open the TCP session and login • the thread is able to periodical sending of retrieving TL1 commands, information are written to the internal data structures by TNRCSP_adva_listen thread
TNRCSP_adva_listen	almost permanent	1	listen the incoming TL1 messages: acknowledgments, responses and autonomous messages match responses which commands and activates finite state machine related to the command write information about device equipment to internal data structures calls asynchronous callbacks for fault notifications
external AP thread calling ADVA SP operation	unblocking (very short)	0..*	validate arguments check device connectivity in case of xc operation, activate finite state machine corresponding the operation and return initial operation result in case of information retrieving, look internal SP data structure and return needed information

			return operation result

The common data structures available inside different threads are protected by lock object.

The threads cooperation sequences in case of XC creation, fault notification are presented on Figure 16-5.

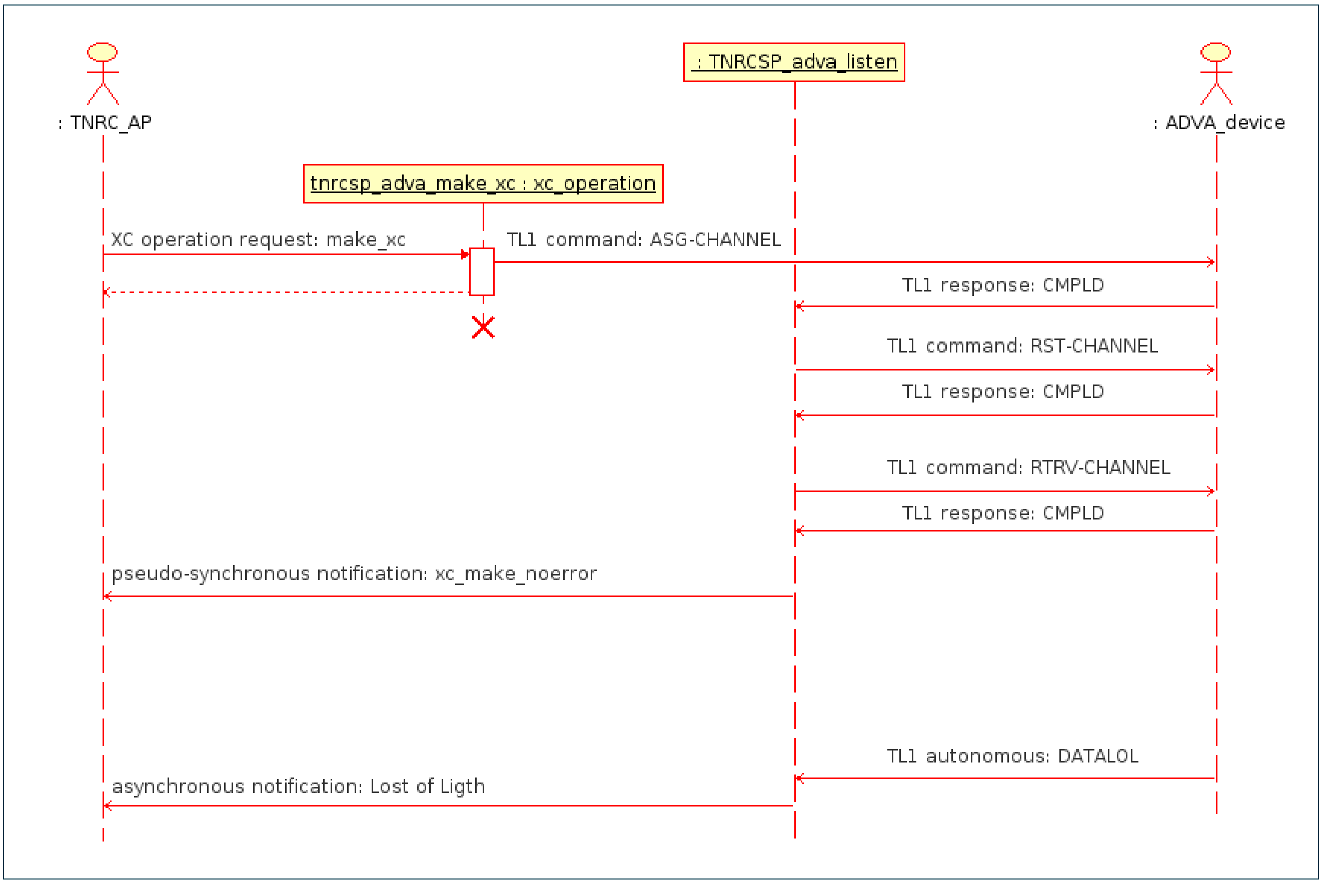

Figure 16-5: TNRC SP ADVA sequence diagram for XC creation and fault notification.

The threads cooperation sequences in case of retrieving of resource details are presented on Figure 16-6.

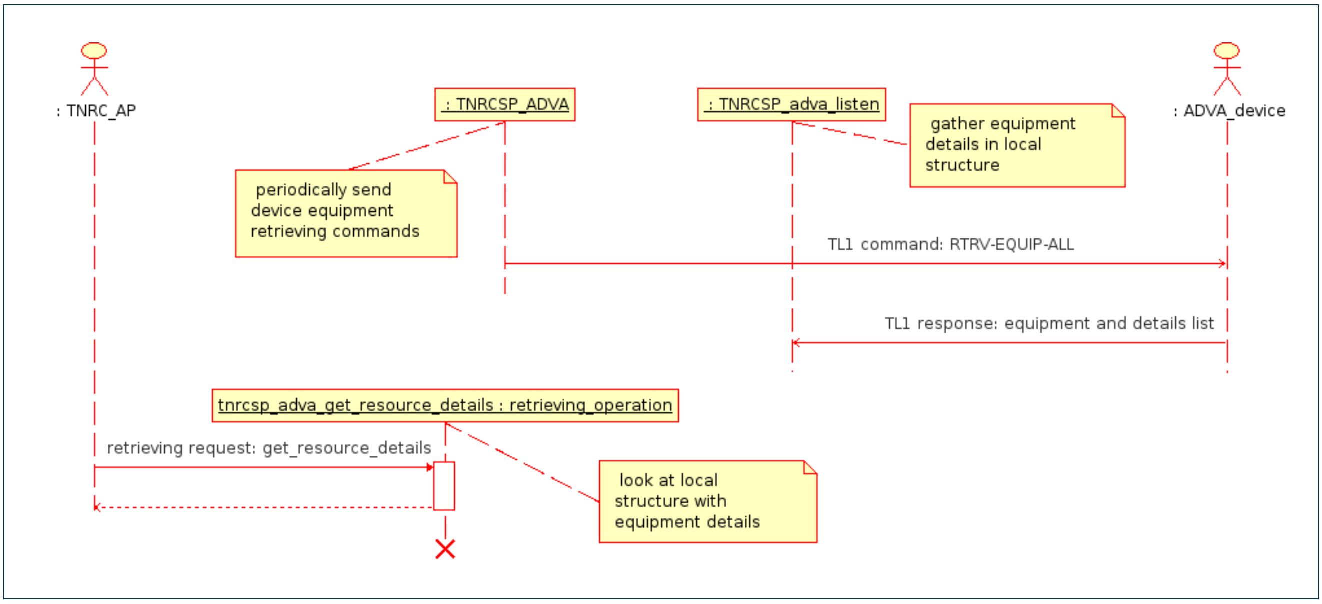

Figure 16-6: TNRC SP ADVA sequence diagram for information retrieve (get_resource_list, get_resource_detail, get_label_list).

All crossconnect operations are presented in form of finite state machines. Diagrams of these state machines are presented on Figure 16-7, Figure 16-8, Figure 16-9, and Figure 16-10.

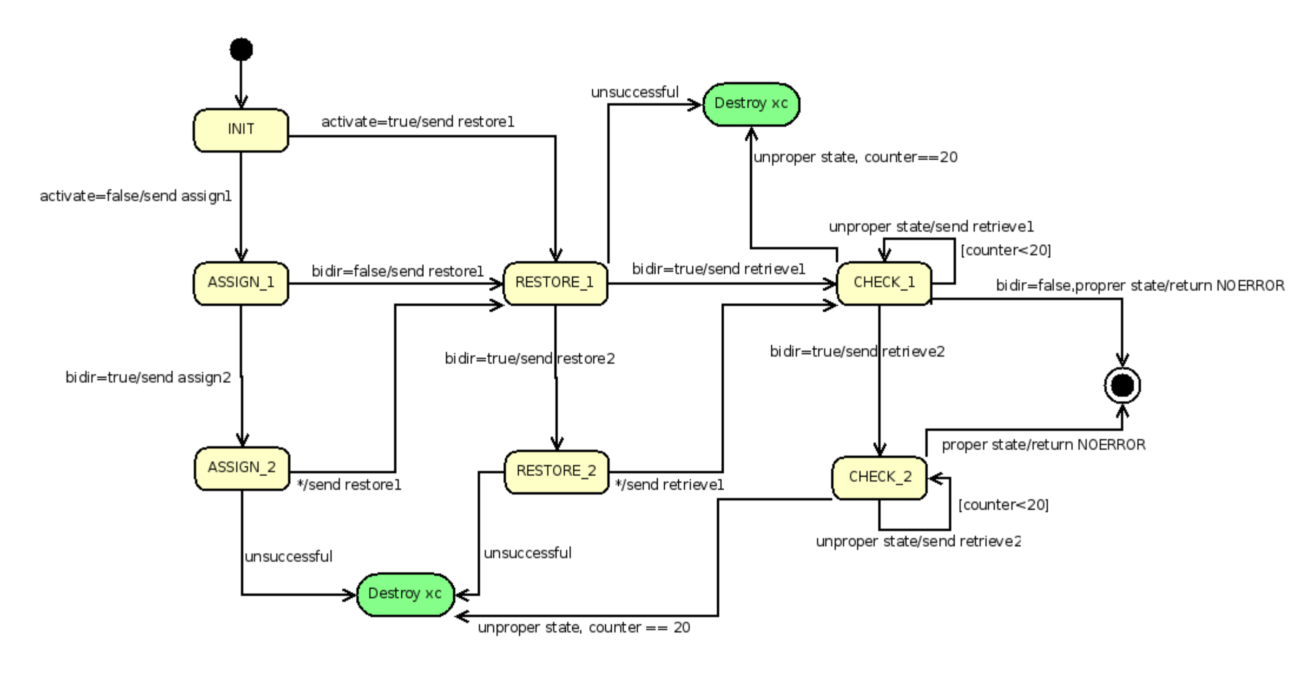

Figure 16-7: TNRC SP ADVA make xc finite state machine (‘Destroy xc’ is entry point to destroy xc finite state machine).

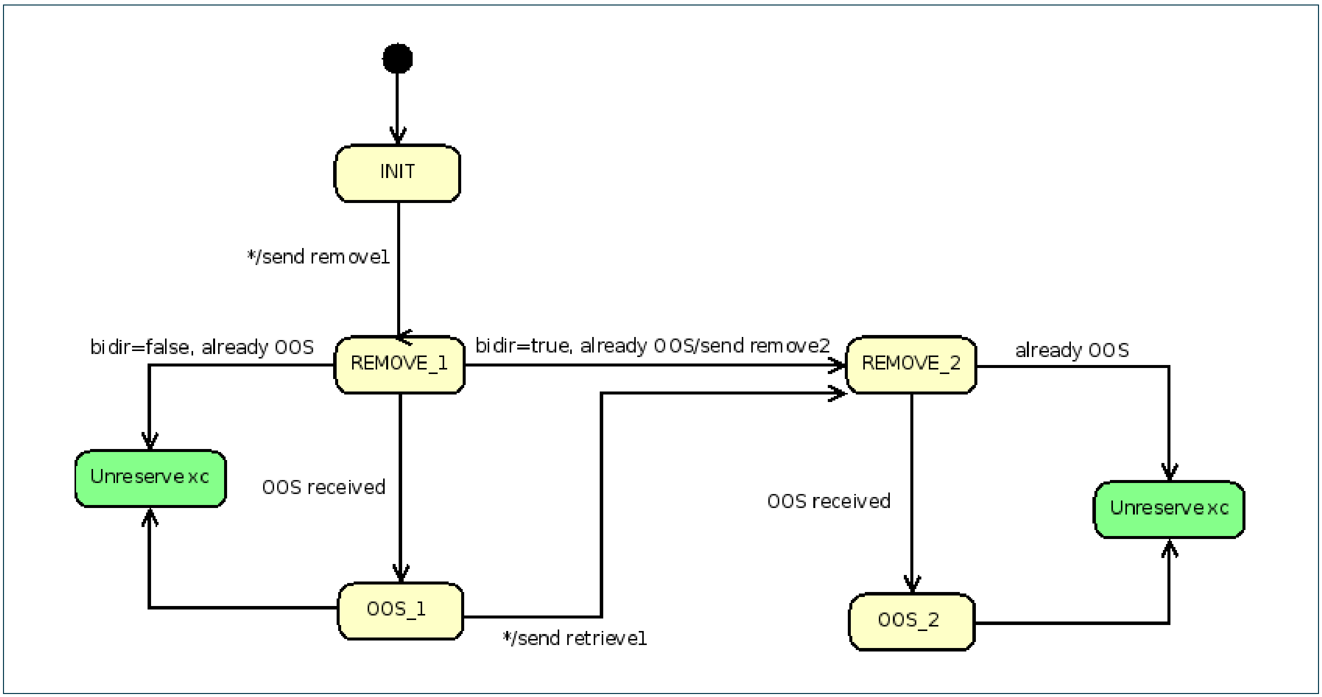

Figure 16-8: TNRC SP ADVA destroy xc finite state machine ('Unreserve xc' is entry point to unreserve xc finite state machine).

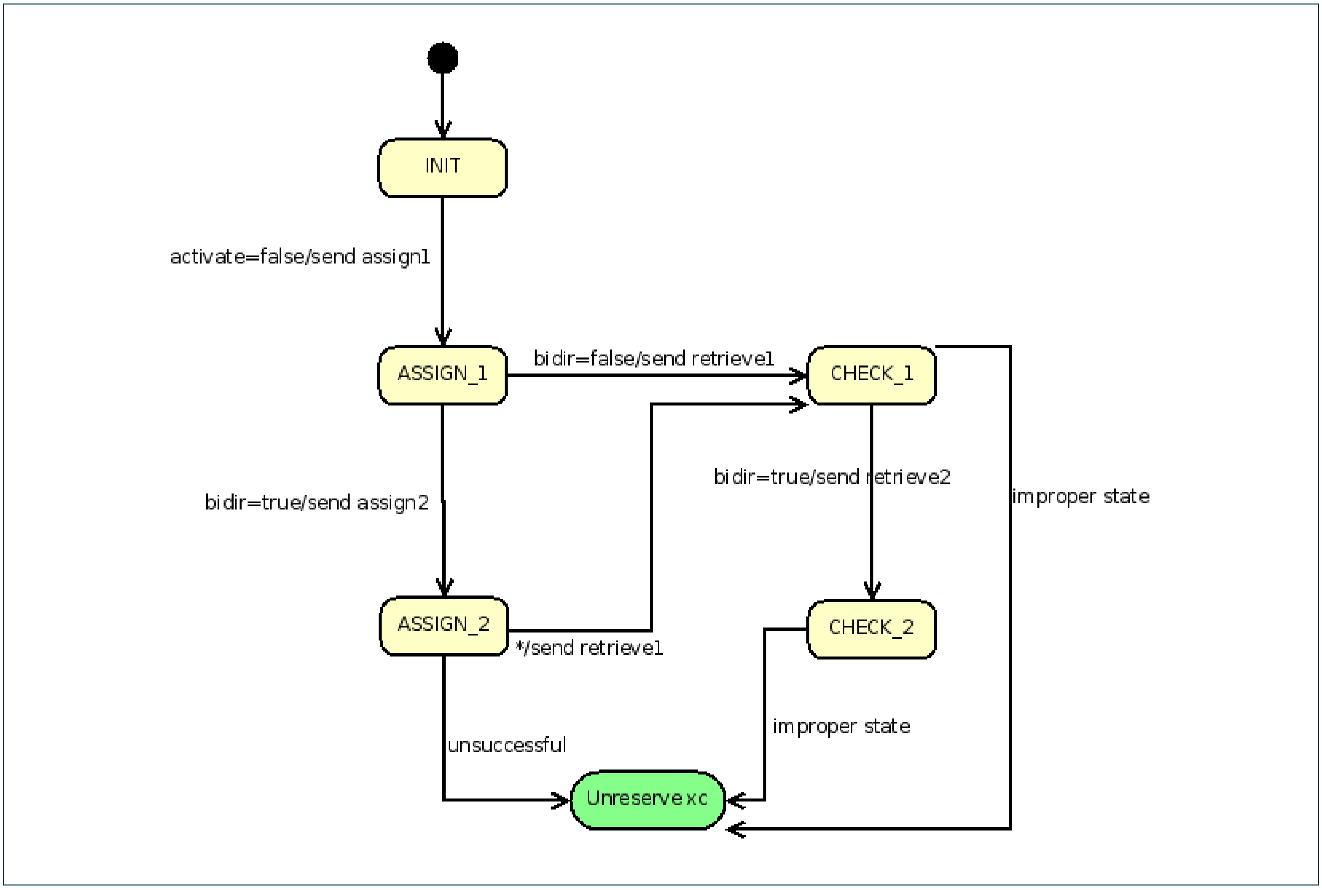

Figure 16-9: TNRC SP ADVA reserve xc finite state machine (‘Unreserve xc’ is entry point to unreserve xc finite state machine).

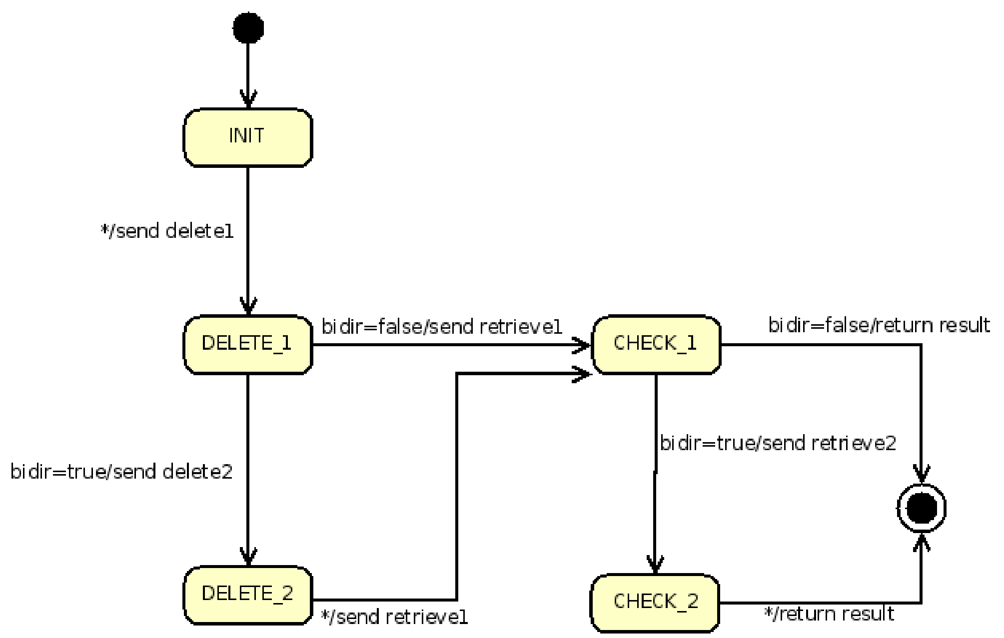

Figure 16-10: TNRC SP ADVA unreserve xc finite state.

C.4.3 TL1 commands

ACT-USER	Activate User (Login)	
Description	This command is used to set up a session (i.e. login) to the specified Network Element.	
Input format	ACT-USER:[TID]:<uid>:CTAG::<pid>; example : ACT-USER::user:6334::****;	
Input parameters	uid	The User Identifier, or login ID.
	pid	The Private Identifier, or password.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).

ASG-CHANNEL	Assign Channel
Description	This command creates a connection between an egress port (to a OLD or XCVR, etc.).
Input format	ASG-

	CHANNEL:[<tid>]::<ctag>::<ingressPortAID>,<egressPortAID>,<channelID>,<connectionType>; example : ASG-CHANNEL:::353431::1-1-8-1,1-1-13-2,20,PASSTHRU;	
Input parameters	ingressPortAID	AID of the ingress OLD, XCVR, or XPDR port (Bay-Shelf-Slot-Port format). For ADD connections, the ingress AID should be a XCVR or XPDR port, and for DROP or PASSTHRU connections, the ingress AID should be an OLD input (Rx) port.
	egressPortAID	AID of the egress OLD, XCVR, or XPDR port (Bay-Shelf-Slot-Port format). For ADD or PASSTHRU connections, the egress AID should be an OLD output (Tx) port, and for DROP connections, the egress AID should be a XCVR or XPDR port.
	channelID	Channel number (20-59) or corresponding wavelength (xxxx.xx), in nanometers.
	connectionType	Identifies the type of connection. Valid values are: • ADD, • DROP, • PASSTHRU.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).

RST-CHANNEL	Restore Channel	
Description	This command places a channel associated with an OLD or XCVR/XPDR port in-service	
Input format	RST-CHANNEL:[<tid>]:<aid>:<ctag>::<channelID>; example :RST-CHANNEL:WEST01:1-1-8-2:CT01::20;	
Input parameters	aid	AID of the egress OLD or XCVR/XPDR port (Bay-Shelf-Slot-Port format).
	channelID	Channel number (20-59) or corresponding wavelength (xxxx.xx), in nanometers.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).

RMV-CHANNEL	Remove Channel
Description	This command places a channel associated with an OLD or XCVR/XPDR port out-of-service
Input format	RMV-CHANNEL:[<tid>]:<aid>:<ctag>::<channelID>;

	example :RMV-CHANNEL:WEST01:1-1-8-2:CT01::20;	
Input parameters	aid	AID of the egress OLD or XCVR/XPDR port (Bay-Shelf-Slot-Port format).
	channelID	Channel number (20-59) or corresponding wavelength (xxxx.xx), in nanometers.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).

DLT-CHANNEL	Delete Channel	
Description	This command deletes a connection between an ingress port (from an OLD or XCVR,etc.) and an egress port (to an OLD or XCVR, etc.). The command only requires the egress port be specified. If the FORCE option is specified, the connection is deleted regardless of state.	
Input format	DLT-CHANNEL:[<tid>]::<ctag>::<ingressPortAID>,<egressPortAID>,<channelID>,,[<force>]; example :DLT-CHANNEL:NODE-1::CT01::,1-1-13-2,20;	
Input parameters	ingressPortAID	Not Supported - AID of the ingress OLD, XCVR, or XPDR port (Bay- Shelf-Slot-Port format). For ADD connections, the ingress AID should be a XCVR or XPDR port, and for DROP or PASSTHRU connections, the ingress AID should be an OLD input (Rx) port.
	egressPortAID	AID of the egress OLD, XCVR, or XDPR port (Bay-Shelf-Slot-Port format). For ADD or PASSTHRU connections, the egress AID should be an OLD output (Tx) port, and for DROP connections, the egress AID should be a XCVR or XPDR port.
	channelID	Channel number (20-59) or corresponding wavelength (xxxx.xx), in nanometers.
	force	Indicates whether deletion may over-ride channel ownership and state. Valid values are: TRUE or FALSE. Default is FALSE.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).

RTRV-CHANNEL	Delete Channel
Description	This command retrieves attributes of a channel associated with an egress OLD, XCVR, or XPDR port. If the channel ID is omitted, then attributes for all assigned channels are retrieved.
Input format	RTRV-CHANNEL:[<tid>]:[<aid>]:<ctag>::[<channelID>];

	example : RTRV-CHANNEL:WEST01:1-1-13-2:CT01::;	
Input parameters	aid	Identifies the AID of the egress OLD port (for ADD or PASSTHRU connections), or egress XCVR/XPDR port (for DROP connections). The AID field can be omitted or specified as ALL to retrieve all assigned channels.
	channelID	Channel number (20-59) or corresponding wavelength (xxxx.xx), in nanometers.
Output parameters	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
	ingressAid	AID of the ingress port
	egressAid	AID of the egress port
	channelID	Channel number: 20-59
	wavelength	Wavelength of the channel (xxxx.xx nm)
	connectionType	Identifies the type of connection. Valid values: ADD DROP PASSTHRU.
	connectionStatus	Status of the connection. Values are: Connected-OOS EQ-in-progress Equalized-IS EQ-High EQ-Low EQ-LOL EQ-Failure EQ-Failure-APR
	owner	Owner (creator) of the connection. Values are: NONE CLI MPLS SNMP TL1 WEB ROOT UNKNOWN
	status	Channel status. Valid values: IS or OOS

RTRV-EQPT-ALL	Retrieve Equipment All
Description	The RTRV-EQPT-ALL command is used to retrieve basic information about all provisioned circuit-packs on the NE.

Input format	RTRV-EQPT-ALL:[TID]::CTAG; example :RTRV-EQPT-ALL:FUTURE1::12345;	
Input parameters	None	
	respCode	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
	aid	Identifies the entity in the NE to which the command pertains. For the EQPT commands, the AID should specify the location identifier (in bay-shelf-slot format) for a particular circuit-pack on the NE. aid is the AID Inventory. as ALL to retrieve all assigned channels.
	typeid	The circuit-pack type or name. This is an alphanumeric string of up to 16 characters. typeid is a string.
	alarmadminstate	The alarm reporting state for this circuit pack. Alarm admin state is of type EnabledDisabled. [DISABLED, ENABLED]
	primarystate	The primary circuit-pack state. Primary state is of type PrimaryState.
	secondarystate	The secondary circuit-pack state. Secondary state is of type SecondaryState.
	regenmode	The regeneration mode for the circuit-pack. This is provisionable for XPDR cards. Valid values are: q DISABLED q ENABLED q ENABLED-OPS q ENABLED-OTN
	portcontrol	The port-control mode for the circuit-pack. This is provisionable for XDPR cards. Port control mode is of type PortControl. [NORMAL, OVERRIDE]
	channelID	The channel associated with the XCVR, XPDR, or EWM circuit-pack. This is provisionable only for EWM circuit-packs.
	wavelength	The wavelength of the channel associated with the XCVR, XPDR, or EWM circuit-pack. This is provisionable only for EWM circuit-packs. wavelength is a character string with format: xxxx.xx nm.

C.4.4 TL1 autonomous messages

REPT ALM	Report Alarm	
Description	This is an autonomously generated alarm. It reports the onset or clearing of a condition that requires immediate attention. Trouble events occurring in the Network Element (NE) are classified as alarmed or non-alarmed events. In general, an alarmed event causes a standing condition and has immediate or potential impact on the operation or performance of the entity. Some form of maintenance effort is required to restore normal operation or performance of the entity after the event has occurred. The string "rr" is used to designate a number of possible options, outlined below.	
Input format	None	
Output format	REPT ALM <rr>: Report Alarm <rr>	
Output parameters	aid	[COMPONENT] aid is the AID Component.
	ntfcncde	The notification code for the message. ntfcncde is of type NotificationCode.
	conditiontype	CONDITIONTYPE conditiontype is of type ConditionType.
	srveff	The effect on service caused by the standing or alarm condition. It can be either SA or NSA. *srveff* is of type ServiceEffect.
	ocrdat	The location associated with a particular command. *locn* is of type Location. *locn* is optional.
	dirn	DIRN dirn is a string. dirn is optional.
	conddescr	\"CONDDESCR\" *conddescr* is a string. The condition description is of format description of ConditionType for a raise of alarm and description of ConditionType Cleared for a alarm clear.

REPT EVT	Report Event	
Description	This is an autonomously generated message. It reports a non-alarmed event. Trouble events occuring in the Network Element (NE) are classified as alarmed or non-alarmed events. The event being reported may change the status or occurrence of an irregularity, which by itself is not severe enough to warrant an alarm notification. One example of this is a performance threshold crossing. This message may also be used to report the recovery from off-normal or trouble conditions that were reported initially via REPT^EVT. This is done using the <condtype> sent by the original event report and using the value CL for <condeff>. Condeff is not supported in 5.0 release. The string "rr" is used to designate a number of possible options, outlined below.	
Input format	None	
Output format	REPT EVT rr: Report Event rr	
Output parameters	aid	[COMPONENT] aid is the AID Component.
	eventtype	EVENTTYPE *eventtype* is a string.

	srveff	SRVEFF *srveff* is a string.
	ocrdat	Date when the specific event or violation occurred. *ocrdat* is a string.
	ocrtm	Time when the specific event or violation occurred. *ocrtm* is a string.
	locn	LOCN *locn* is a string. *locn* is optional.
	dirn	DIRN dirn is a string. dirn is optional.
	conddescr	\"CONDDESCR\" *conddescr* is a string.

C.4.5 Error codes

ConditionType	**Description**
ADMIN	Alarm Administration Status
AGENT-FAIL	Agent Failure
AIS	Alarm Indication Signal
AIS-L	Alarm Indication Signal - Line
AIS-P	Alarm Indication Signal Present on Path Layer
AIS-S	Alarm Indication Signal Present on Section Layer
AISSYNCPRI	Primary Sync AIS
AISSYNCSEC	Secondary Sync AIS
APR-ADJ-FAIL	APR Adjust Fail
APR-ACT	APR Is Active
APS	Automatic Protection Switch In Effect
ASE-TBL-FAIL	Build ASE Calibration Table Failure
AUTOPROV	Shelf Lost Database and In Auto Provisioning Mode
BACKREFLECTION	Back Reflection Error
BATT	Battery Failure
BAYBATT_A	Bay Battery A Failed
BAYBATT_B	Bay Battery B Failed
BAYBRKER	Bay Breaker Triggered
BAYBRKER_A	Bay Breaker A Tripped
BAYBRKER_B	Bay Breaker B Tripped
BDI-P	Backward Defect Indication on Path
BDI-S	Backward Defect Indication on Section
BEI-P	Backward Error Indication on Path
BEI-S	Backward Error Indication on Section
BRKER-A	Breaker A Tripped
BRKER-B	Breaker B Tripped
CLKFAIL0	Clock Failure from Plane 0
CLKFAIL1	Clock Failure from Plane 1
CLKPROTFAIL	Clock Protection Failure
COMMLINK-0	Communications Link on Plane 0 down to SP
COMMLINK-1	Communications Link on Plane 1 down to SP
COMM-LOA	COMM/Loss of Association
COMMLINK	Communications Link Failure to SP
CP-FLT	Circuit Pack Fault
CP-INT-MISM	Circuit Pack Int. Mismatch

CP-MISM	Circuit Pack Mismatch
CP-UNEQ	Circuit Pack Unequipped
CVS	Coding Violations-Section
DATALOL	Data Loss Of Light
DSK-LOW	Disk Low
DUAL-SECURITY	Dual Security
DUP-ID	Duplicate NE ID
DUP-NAME	Duplicate NE Name
EDFA-GAIN-HI	EDFA Gain High
EDFA-GAIN-LOW	EDFA Gain Low
EDFA-INP-B-HI	Second Stage Input High
EDFA-INP-B-LO	Second Stage Input Low
EDFA-INP-HI	EDFA Input Power High
EDFA-INP-LO	EDFA Input Power Low
EDFA-LP-HI	EDFA Laser Power High
EDFA-LP-LO	EDFA Laser Power Low
EDFA-LP-A-HI	A/EDFA Laser Power High
EDFA-LP-A-LO	A/EDFA Laser Power Low
EDFA-LP-B-HI	B/EDFA Laser Power High
EDFA-LP-B-LO	B/EDFA Laser Power Low
EDFA-MIDST-HI	EDFA Mid-Stage High
EDFA-MIDST-LO	EDFA Mid-Stage Low
EDFA-OP-A-HI	A/EDFA Optical Power High
EDFA-OP-A-LO	A/EDFA Optical Power Low
EDFA-OUT-HI	EDFA Output Power High
EDFA-OUT-LO	EDFA Output Power Low
EDFAPOWERLO	EDFA Power Low
ESS	Error Seconds-Section
EVS	Encoding Violations-Section
EXOSCSW	Excessive OSC Switching
FANFAIL1	Fan Unit 1 not Operating
FANFAIL2	Fan Unit 2 not Operating
FANFAIL3	Fan Unit 3 not Operating
FANFAIL4	Fan Unit 4 not Operating
FANFAIL5	Fan Unit 5 not Operating
FANFAIL6	Fan Unit 6 not Operating
FANCOMMFAIL	Fan Communication Failure
FLASHFAIL	Write To Flash Failed
FPGAMISM	FPGA Mismatch
FRCD-0	Forced Switch to Plane 0
FRCD-1	Forced Switch to Plane 1
FRCDSWTOPRI	Forced Sync Reference Switch To Primary
FRCDSWTOSEC	Forced Sync Reference Switch To Secondary
FRNGSYNCCG	In Freerun Timing Mode
GAIN-TILT-FAIL	EDFA Gain Tilt Fail
GAINHI	Gain High
GAINLO	Gain Low
HITEMP	Shelf High Temperature
HIVOLT-A	Rectifier A High Voltage
HIVOLT-B	Rectifier B High Voltage

HKIN	Housekeeping Input
HKIN1	Housekeeping Input 1
HKIN2	Housekeeping Input 2
HKIN3	Housekeeping Input 3
HKIN4	Housekeeping Input 4
HKOUT	Housekeeping Output
HLDOVRSYNC	In Holdover Timing Mode
IAE-S	Incoming Alignment Error on Section
IDPFAIL	IDProm Cannot be Read
IPVIOL	IP Violation
LBCHI	High Laser Bias Current
LBC-A-HI	A/Laser Bias Current High
LBC-A-LO	A/Laser Bias Current Low
LBC-B-HI	B/Laser Bias Current High
LBC-B-LO	B/Laser Bias Current Low
LBC-DATA-HI	Data/Laser Bias Current High
LBC-DATA-LO	Data/Laser Bias Current Low
LBC-OSC-HI	OSC/Laser Bias Current High
LBC-OSC-LO	OSC/Laser Bias Current Low
LBCLO	Laser Bias Current Low
LCK-P	Lock Signal Received on Path
LINEUP-UNKNOWN	SWDL Lineup Unknown
LINKFAIL	Link Synchronization Failure
LKOUT	Protection Switch Lockout
LKOUT-0	Protection Switch Lockout on Plane 0
LKOUT-1	Protection Switch Lockout on Plane 1
LMC-HI	Laser Modulation Current High
LMC-LO	Laser Modulation Current Low
LOA	Loss Of Association
LOCKOUTOFREF	Lockout Of Reference
LOF	Loss Of Frame
LOF-0	Loss Of Frame on Plane 0
LOF-1	Loss Of Frame on Plane 1
LOFREQ	Loss Of Frequency
LOFSYNCPRI	LOF Sync Primary
LOFSYNCSEC	LOF Sync Secondary
LOI	Loss Of Input
LOI-0	Loss Of Input Failure – 0
LOI-1	Loss Of Input Failure - 1
LOL	Loss of Light
LOM	Loss of Multiframe
LOP	Loss of Pointer
LOS	Loss Of Signal
LOS-0	Loss Of Signal on Plane 0
LOS-1	Loss Of Signal on Plane 1
LOSCHARSYNC	Loss of Character Sync
LOSSYNCPRI	Primary Sync Loss of Signal
LOSSYNCSEC	Secondary Sync Loss of Signal
LOTEMP	Shelf Low Temperature
LOWVOLT-A	Rectifier A Low Voltage

LOWVOLT-B	Rectifier B Low Voltage
LPAHI	A/Laser Power High
LPALO	A/Laser Power Low
LPBHI	B/Laser Power High
LPBLO	B/Laser Power Low
LPBK-F	Facility Loopback Initiated (Far-End)
LPBK-T	Terminal Loopback Initiated (Near-End)
LPHI	Laser Power High
LPLO	Laser Power Low
LSROVR	Laser Safety Override
LTAHI	A/Laser Temperature High
LTALO	A/Laser Temperature Low
LTBHI	B/Laser Temperature High
LTBLO	B/Laser Temperature Low
LTHI	Laser Temperature High
LTLO	Laser Temperature Low
MANSWTOPRI	Manual Sync Reference Switch To Primary
MANSWTOSEC	Manual Sync Reference Switch To Secondary
MANUAL-0	Manual Switch to Plane 0
MANUAL-1	Manual Switch to Plane 1
MEM-LOW	Memory Low
MISCON	Equipment Misconfiguration
MSA-OPR-HI	A/EDFA Mid-Stage RX Optical Power High
MSA-OPR-LO	A/EDFA Mid-Stage RX Optical Power Low
NO-ASE-TBL	No ASE Calibration Table
NO-GAIN-CALIB	No Calibration Gain
OCI-P	Open Connection Indication on Path
OIF	Optical Input Failure
OLP-HI	Optical Line Power High
OLP-LO	Optical Line Power Low
OPR-DATA-HI	Data/Optical RX Power High
OPR-DATA-LO	Data/Optical RX Power Low
OPR-OSC-HI	OSC/Optical RX Power High
OPR-OSC-LO	OSC/Optical RX Power Low
OPRLO	Low Optical Power Received
OPRHI	High Optical Power Received
OPT-DATA-HI	Data/Optical TX Power High
OPT-DATA-LO	Data/Optical TX Power Low
OPT-OADM-HI	TX Optical Power To OADM High
OPT-OADM-LO	TX Optical Power To OADM Low
OPT-OSC-HI	OSC/Optical TX Power High
OPT-OSC-LO	OSC/Optical TX Power Low
OPT-SFLMT	EDFA Output Power Exceeds Safety Limit
OPTLO	Low Optical Power Transmitted
OPTHI	High Optical Power Transmitted
OSCCGF	OSC Configuration Unsupported
OSCFLT	Communication Failure on OSC Channel
OSCLFR	OSC Loss of Frame
OSCLOA	OSC Loss Of Association
OSCLOL	OSC Loss Of Light

OSCTXDISABLED	OCS TX Power Disabled
OSCWS	OSC Wiring Suspect
OVERBW	Overbandwidth
PEFAIL-0	Power Equalization Failure At Plane 0
PEFAIL-1	Power Equalization Failure At Plane 1
PLM-P	Payload Mismatch on Path
PRBS-LINE	PRBS Line Initiated
PRILOCKOUTREF	Primary Lockout Reference
PT-FLT	SFP/XFP Port is in a Fault State
PT-MISM	SFP/XFP Port Does Not Match
PT-UNEQ	SFP/XFP Port is Removed/Missing
PT-UNKNOWN	SFP/XFP Port is Unknown
PUMP-SHUTDOWN	Pump Shutdown
RFI-L	Remote Failure Indication - Line
RINV	Rate is Invalid
RLOS	Remote Loss Of Signal
RMPROTCMD	Remote Command Fail
ROOR	Rate Out of Range
RX-ERR	Receive Error
SD	Signal Degrade
SD-0	Signal Degrade Plane 0
SD-1	Signal Degrade Plane 1
SD-ODU	ODU Signal Degrade
SEC-VOIL	Invalid IP or ICMP Packets Received
SECLOCKOUTREF	Secondary Lockout Reference
SEFS	Severely Errored Frame-Section
SEFSS	Severely Errored Frame Seconds - Section
SER-UNKN	Shelf Serial Number Invalid
SESS	Severely Errored Seconds-Section
SETPRIREFFAIL	Set Primary Clock Fail
SETSECREFFAIL	Set Secondary Clock Fail
SF-0	Signal Failure on Plane 0
SF-1	Signal Failure on Plane 1
SH-UNEQ	Shelf Unequipped
SHBATT-A	Shelf Battery A Failed
SHBATT-B	Shelf Battery B Failed
SPPROTFAIL	SP Protection Is Not Available
SSF	Severe Signal Failure
SW-ACTV	Unsuccessful Activation of Software
SW-MISM	Incorrect Software Load
SW-TRNF	Unsuccessful Transfer of Software
SWCOMPL	Automatic Protection Switching Complete
SWDL-FAIL	Unsuccessful Download of Software
SWFAIL-0	Protection Switch Unsuccessful to Plane 0
SWFAIL-1	Protection Switch Unsuccessful to Plane 1
TIM-P	Trail Trace Identifier Mismatch on the Path Layer
TIM-S	Trail Trace Identifier Mismatch on Section
TLTLO	Low Transmit Laser Temperature
TLTHI	High Transmit Laser Temperature
VOA-ATT-HI	VOA Attenuation High

VOA-ATT-LO	VOA Attenuation Low
WTR	Wait To Restore

Appendix D TNRC Specific Part for Calient DiamondWave FiberConnect

In this section we provide the specifications for the TNRC_SP software design for the Calient which was developed and implemented by developers at PSNC and UEssex. The software provides various functionalities and capabilities that allows the Calient Optical Cross connect (TN equipment) located at UEssex to be dynamically and remotely controlled. We also provide an analysis the software design and architecture of the software looking at the uses case and state diagrams used during the development. This document also provides the supported TNRC_SP APIs in accordance the earlier released specification document, providing the supported data structures and variables. Finally we then provide an appendix decribing all possible error codes.

D.1 Calient TNRC_SP Software Design

This section describes the data structures and API available for communication between the TNRC_AP and TNRC_SP.

D.1.1 Data structures

```
typedef unsigned int tnrcsp_handle_t;

typedef enum {
       TNRCSP_RESULT_NOERROR = 0,
       TNRCSP_RESULT_EQPTLINKDOWN,
       TNRCSP_RESULT_PARMERROR,
       TNRCSP_RESULT_NOTCAPABLE,
       TNRCSP_RESULT_BUSYRESOURCES,
       TNRCSP_RESULT_INTERNALERROR,
       TNRCSP_RESULT_GENERICERROR
} tnrcsp_result_t;
```

```
typedef void (*tnrcsp_response_cb_t)(tnrcsp_handle_t *handle, tnrcsp_result_t result,
void *cxt);

typedef void (*tnrcsp_notification_cb_t)(tnrcsp_handle_t *handle,
tnrcsp_fsc_resource_id_t **failed_resource_listp, void *cxt);

typedef unsigned int tnrcsp_fsc_evmask_t; /* values TBD, depending on the hw */

typedef struct {
       SLIST_HDR                 event_list;

       tnrc_portid_t                    portid;
       tnrc_operstate_t                 oper_state;
       tnrc_adminstate_t         admin_state;
       tnrcsp_fsc_evmask_t       events;
} tnrcsp_fsc_event_t;

typedef struct {
       SLIST_HDR                 resource_list;

       tnrc_portid_t                    portid;
} tnrcsp_fsc_resource_id_t;

typedef struct {
       tnrc_portid_t                    portid;
       tnrc_operstate_t                 oper_state;
       tnrc_adminstate_t         admin_state;
       tnrcsp_fsc_evmask_t       last_event;
       /* other values TBD */
} tnrcsp_fsc_resource_detail_t;

typedef enum {
       TNRCSP_LISTTYPE_UNSPECIFIED,
       TNRCSP_LISTTYPE_RESOURCES
} tnrcsp_list_type_t;
```

D.1.2 Detailed specification of TNRC_SP FSC API functions

The following functions should be included in the API:

XC creation	tnrcsp_result_t **tnrcsp_fsc_calient_make_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t activate, tnrcsp_response_cb_t response_cb, void *response_cxt, tnrcsp_notification_cb_t async_cb, void *async_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
activate	In	turn a couple of reserved ports into a XC

response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
async_cb	In	asynchronous notification function provided by the TNRC AP, to be called whenever something asyn occurs on the XC or some of its elements
async_cxt	In	asynchronous context provided by the TNRC AP, to be returned in the async notification callback
Description		
This function will create the XC, with the following behaviour: ▪ It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device ▪ Later, when the XC has been completed or failed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation, ▪ XC creation is composed from few TL1 commands sequence, ▪ If XC creation failed, all resources are released, and device should be in the same state as before XC creation, ▪ Correctness of XC creation is checked at the end of action, ▪ XC activation (activate=True) will success only if there was XC reservation called before, ▪ Any future event related to the XC or one of its components (e.g. ports) will be reported to the TNRC AP with the asynchronous callback. Calient uses TL1 autonomous messages to inform about events and alarms.		
Used TL1 commands		
ACT-CRS	Reserve XC	Normal situation
ENT-CRS	Activate XC	Normal situation if XC activation
RTRV-CRS	Retrieve XC	Normal situation
CANC-CRS	Unreserve XC	Exceptional situation if XC activation
DLT-CRS	Delete XC	Exceptional situation
Synchronous function results		
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid	
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in	
TNRCSP_RESULT_PARMERROR	Not valid arguments	
Pseudo-synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	
TNRCSP_RESULT_PARMERROR	Wrong argument value	
TNRCSP_RESULT_BUSYRESOURCES	Resources not available	
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure	
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid and resources are available)	

XC removal	tnrcsp_result_t

	tnrcsp_fsc_calient_destroy_xc(tnrcsp_handle_t handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_boolean_t virtual, tnrc_boolean_t deactivate, tnrcsp_response_cb_t response_cb, void *response_cxt)	
Parameters		
handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
deactivate	In	turn an XC into a couple of reserved ports
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
Description		
This function will destroy the XC, with the following behaviour: ▪ It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device, ▪ Later, when the XC removal has been completed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation, ▪ XC deletion is composed from few TL1 commands sequence, ▪ In case of any unsuccessful processing of command the release of resources is continued, ▪ XC deactivation (deactivate=True) will success only if there was active XC, ▪ Correctness of XC deletion is checked at the end of action.		
Used TL1 commands		
CANC-CRS	Unreserve XC	Normal situation
DLT-CRS	Delete XC	Normal situation
RTRV-CRS	Retrieve XC	Normal situation
Synchronous function results		
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid	
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in	
TNRCSP_RESULT_PARMERROR	Not valid arguments	
Pseudo-synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	
TNRCSP_RESULT_PARMERROR	Wrong argument value	
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure	
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid)	

XC reservation	tnrcsp_result_t **tnrcsp_fsc_calient_reserve_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t virtual, tnrcsp_response_cb_t response_cb, void *response_cxt)
Parameters	

handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id
portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback

Description

This function will reserve the XC, with the following behaviour:

- It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device,
- Later, when the XC reservation has been completed or failed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation,
- XC reservation is composed from few TL1 commands sequence,
- If XC reservation failed, all resources are released, and device should be in the same state as before XC creation,
- Correctness of XC reservation is checked at the end of action.

Used TL1 commands

ACT-CRS	Reserve XC	Normal situation
RTRV-CRS	Retrieve XC	Normal situation
DLT-CRS	Delete XC	Exceptional situation

Synchronous function results

TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in
TNRCSP_RESULT_PARMERROR	Not valid arguments

Pseudo-synchronous function results

TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value
TNRCSP_RESULT_BUSYRESOURCES	Resources not available
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid and resources are available)

XC unreservation	tnrcsp_result_t **tnrcsp_fsc_calient_unreserve_xc**(tnrcsp_handle_t *handlep, tnrc_portid_t portid_in, tnrc_portid_t portid_out, tnrc_xcdirection_t direction, tnrc_boolean_t virtual, tnrcsp_response_cb_t response_cb, void *response_cxt)

Parameters

handlep	Out	generic handler, generated by the TNRC SP and kept by the TNRC AP
portid_in	In	ingress port id

portid_out	In	egress port id
direction	In	directionality of the XC (unidir and bidir)
response_cb	In	pseudo-synchronous callback function provided by the TNRC AP, to be called when the operation has been completed
response_cxt	In	response context provided by the TNRC AP, to be returned in the response callback
Description		

This function will unreserve the XC, with the following behaviour:

- It returns soon after the preliminary checks have been carried out (parameters are in valid range and there is connection to device) and send first TL1 command to device,
- Later, when the XC removal has been completed, the TNRC SP will come back to the TNRC AP using the response callback (if any) and context, and delivering the result of the operation,
- XC unreservation is composed from few TL1 commands sequence,
- In case of any unsuccessful processing of command the release of resources is continued,
- XC unreservation will success only if XC is not active,
- Correctness of XC unreservation is checked at the end of action.

Used TL1 commands		
DLT-CRS	Delete XC	Normal situation
RTRV-CRS	Retrieve XC	Normal situation

Synchronous function results	
TNRCSP_RESULT_NOERROR	Connected, logged in and arguments are valid
TNRCSP_RESULT_EQPTLINKDOWN	No TCP session to device or not logged in
TNRCSP_RESULT_PARMERROR	Not valid arguments
Pseudo-synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure
TNRCSP_RESULT_GENERICERROR	Device denies to process an action (but arguments are valid)

Register events notification	tnrcsp_result_t **tnrcsp_fsc_calient_register_async_cb**(tnrcsp_fsc_event_t *events)	
Parameters		
events	In	List of events to be notified to the TNRC AP; each event item focuses on a port and reports about states (operational, administrative) and occurred events (using a bitmask)
Num	In	number of events
Description		

This function will register events to be notified to TNRC AP; Notification mechanism is invoked asynchronously by the TNRC SP when:

- TL1 autonomous alarm notification appear,
- operation state occur,

• administration state occur. The administrative and operational status are periodically polled and states are compared with registered values. This function doesn't use any TL1 command.	
Synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_PARMERROR	Wrong argument value
TNRCSP_RESULT_INTERNALERROR	Unrecognized action failure

Fetching of resources list	tnrcsp_result_t **tnrcsp_fsc_calient_get_resource_list**(tnrcsp_fsc_resource_id_t **resource_listp)	
Parameters		
resource_listp	Out	to be returned as pointer to the list of resource ids
num	Out	number of returned resource ids
Description		
This function allows to fetch the list of underlying resources. Each resource will be assigned an id by the TNRC_SP. The resource identifier is composed from Access Identifier code (AID) of card. For example: AID = 1.1.8 ("*bay-shelve-slot*") then port id = 118. This transformation generates unique ids and its is easily reversible. This function doesn't send any TL1 command. It used gathered information by periodically sending a loop of RTRV-PORT for each port continually.		
Synchronous function results		
TNRCSP_RESULT_NOERROR	Action processed successfully	
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost	

Fetching of details about a specific resource	tnrcsp_result_t **tnrcsp_fsc_calient_get_resource_detail**(tnrcsp_fsc_resource_id_t resource_id, tnrcsp_fsc_resource_detail_t *resource_detailp)	
Parameters		
resource_id	In	identifier of the resource whose details are fetched
resource_detailp	Out	to be returned as pointer to the structure of resource details
Description		
This function allows to fetch the details of a specific resource. The details contains information about: o current administrative state (disabled, enabled), o current operational state (disabled, enabled), o last event. There is also second condition for crossconnection possibility – labels (wavelengths) for both resources must be the same. Administrative state depends on PrimaryState returned by device:		

Administrative State	PrimaryState	Description
TNRC_ADMINSTATE_ENABLED	IS	In-service
	IS-ANR	In-service, abnormal

	IS-ANRST	In-service, abnormal and restricted
	IS-NR	In-service, normal
	IS-RST	In-service, restricted
	UMA	Under Management
TNRC_ADMINSTATE_DISABLED	OSS	Out-of-service
	OSS-AU	Out-of-service, autonomous
	OOS-AUMA	Out-of-service, autonomous and management
	OOS-AURST	Out-of-service, autonomous and restricted
	OOS-MA	Out-of-service, management
	OOS-MAANR	Out-of-service, management and abnormal

Operational state depends on SecondaryState returned by device:

Administrative State	SecondaryState	Description
TNRC_OPERSTATE_ENABLED	ACT	Active
TNRC_OPERSTATE_DISABLED	ASWDL	Automatic Software Download
	DGN	Diagnostic
	DSBLD	Data Sync
	FLT	Fault
	LPBK-FAC	Loopback Facility
	LPBK-TERM	Loopback Terminal
	MISM	Mismatched
	NALM	No Alarm
	PRBS	PRBS test
	SGEO	Supporting entity outage
	STBY	Supporting entity outage
	SWDL	Software download
	TCAI	TCA Inhibited
	TUNE	Indicates laser is in the process of turning on
	UAS	Unassigned
	UEQ	Unequipped

Last event present last non-alarm or alarm condition. Alarm values are presented in the error table section of annex. Non-alarm events are not listed yet (lack in documentation).

This function doesn't send any TL1 command. It used gathered information by periodically sending a loop of RTRV-PORT for each port continually

Synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value

Fetching of crossconnections list	tnrcsp_result_t **tnrcsp_fsc_calient_save_xc_list**(tnrcsp_resource_id_t resource_id, unsigned int* num)	
Parameters		
resource_id	In	identifier of the resource whose labels are fetched
label_listp	Out	to be returned as pointer to the list of labels
num	Out	number of returned resource ids

Description	
This function doesn't send any TL1 command. It used gathered information by periodically send RTRV-CRS command.	
Synchronous function results	
TNRCSP_RESULT_NOERROR	Action processed successfully
TNRCSP_RESULT_EQPTLINKDOWN	TCP session to device lost
TNRCSP_RESULT_PARMERROR	Wrong argument value

Flushing of lists	tnrcsp_result_t **tnrcsp_fsc_calient_flush_list**(tnrcsp_list_type_t list_type, void *list)	
Parameters		
list_type	In	Type of the list to be flushed (might be left unspecified)
list	In	Pointer
Description		
This function allows to free a generic simple list of elements previously returned by the TNRC_SP. If the freeing is simple (i.e. no nested pointers), then the list type could be unspecified.		

D.2 Calient TNRC_SP Software Implementation

D.2.1 TNRC_SP use-case scenarios

In order to fully develop the TNRC_SP, various considerations and assumptions were made based on the specification documents TNRC specification documents. These assumptions describe the functionality and characteristics as follows:

a) implementing the specific actions on the hardware, by means of any available and suitable management interface (e.g. TL1, SNMP, CLI).

b) decoupling the mechanisms of the lower management from the upper layers (i.e. TNRC_AP):

   i. decoupling of blocking/unblocking sync/async communication,

   ii. decoupling of objects or sessions identifiers,

c) perform any final translation from the semantics and object identifiers passed by the TNRC_AP into those needed to communicate with the hardware.

d) hide away from TNRC_AP some unneeded peculiarities of the underlying transport network equipment; e.g. the port in an FSC switch might be organized in rack, shelf, and port, and the port unique ids on the device could be made of these 3 identifiers. The G^{2}MPLS, and the TNRC_AP on behalf of it, are not interested in these details, and just need to use a unique port id (built as the TNRC_SP likes). Of course, some exceptions to this rule might exist and they need to be taken into account, and this will be discussed and addressed case by case.

Based on these requirements a use case diagram has been developed to explain the various commands that are supported on the Calient OXC. This diagram is shown in

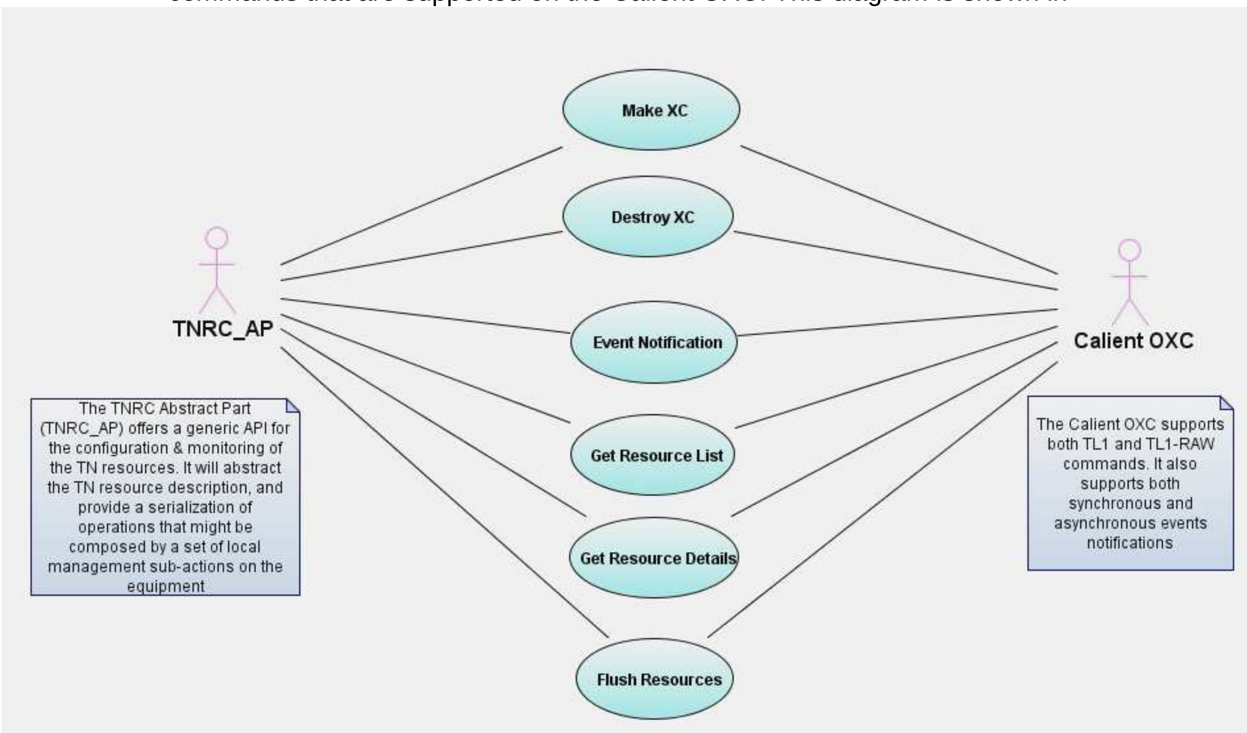

Figure 16-11 and it explains in human readable terms what is expected from both the TNRC_AP and the Calient OXC. The TNRC_SP will basically offer the upper part (TNRC_AP) an API specific to the equipment considered, in this case the Calient OXC. It will name resources based on the underlying TN technology and SwCap (Switching Capability) which in the Calient is Fibre switching. The core part of the TNRC_SP is highly dependent on the Calient OXC's controlling agent in which TL1, TL1-RAW and SNMP are supported to configure, manage and monitor the OXC. The Calient OXC can also receive the required function (commands) using TL1 commands language via Telnet and Serial interfaces.

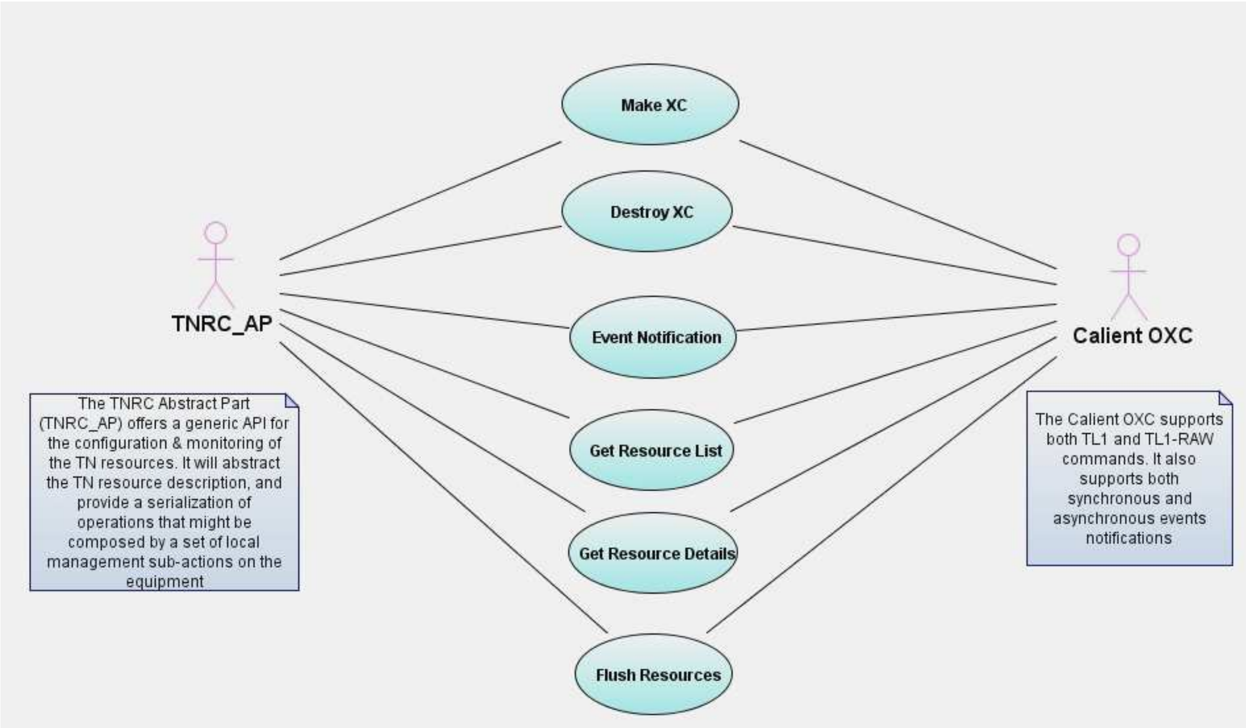

Figure 16-11: Uses Case Diagram for the TNRC_SP.

We have chosen to use the TL1 command language in conjunction with Telnet interface because of speed, respective modularity, flexibility and ease of integration. Further details on the integration of the TLI agent, Telnet interface and the TNRC_SP will be provided later in the document. The main functions to be implemented are:

- Make XC
- Destroy XC
- Reserve XC
- Unreserve XC
- Register Asynchronous Call Back
- Get Resource list
- Get Resource Details
- Flush Resources.

To further explain the functions described in the use case, individual functions are described based on the command that will be sent, the response expected and the actions to be executed between the TNRC_AP, TNRC_SP and Calient OXC.

D.2.1.1 *Make XC*

This command allows cross-connections to be performed. Although the Calient OXC supports the reservation of XC in order to activate it at a later time, our implementation of the TNRC_SP doesn not support this function. This can be easily integrated to the TNRC_SP if needed in the future.

ENT-CRS	**Entering Connection**	
Description	This command creates a new connection between two ports.	
Input format	ENT-CRS:[TID]:<srcPort>,<dstPort>:[CTAG]:: [<groupName>],[<connType>],[<connName>],[<waveband>], [<force>]; example : agent> ent-crs::0.12b.8,0.12b.5:::calient,1way,sf_la; calient 02-11-01 13:36:29 M 0 COMPLD /* ENT-CRS OK. 0.12b.8>0.12b.5 */ ;	
Input parameters	srcPort	This parameter specifies the port used for the connection. *srcPort* must be specified.
	dstPort	This parameter specifies the port used for the destination. *dstPort* must be specified.
	groupName	This parameter specifies the name of the group who is serviced by a connection. The group name consists between 1 to 35 alphanumeric characters, including special characters such as periods (.) and underscores (_). *groupName* is optional.
	connType	This parameter specifies the direction of a connection. Options are: - 1way for unidirectional - 2way for bidirectional connType is optional.
	connName	This parameter specifies the connection name, consists of up to 35 alphanumeric characters. The conn_name must be unique within a customer group, and you cannot use duplicate connection names for the same customer. *connName* is optional.
	wavebandwaveband	This parameter specifies the waveband constraint when making a connection. Options are: - WBand (default) for wavelengths between 1260 and 1625 nm - CBand for wavelength between 1530–1565 nm - LBand for wavelength between 1565–1610 nm

		- XLBand for wavelength between 1610–1625 nm - OBand for wavelength between 1260–1360 nm *waveband* is optional.
	force	This parameter specifies if all specified parameters can be forced onto member ports. Options are: - N (default) indicates only the applicable parameters (those which have not been previously overridden) are provisioned onto member ports. - Y indicates all parameters are forced onto the member ports of a port group. *force* is optional.
Output parameters	*respCode*	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
Error Codes	ENEQ	No hardware present for that connection
	IDNV	1. Invalid Data. Invalid (hardware not present) or unsupported equipment. 2. Invalid Port descriptor. Card might have been deleted.
	SRCN	Port already in connection. Connection already exists.

Details:

The cross-connections are completed in less than one second and the responses are displayed back to the TLI agent prompt immediately.

D.2.1.2 *Destroy XC*

This command permanently deletes cross-connections on the Calient OXC. Once the XC has been deleted the resources involved are no longer reserved and they become available immediately. Although the Calient supports a functionality to only deactivate the XC but not delete it, we do not currently support this function in the TNRC_SP implementation.

DLT-CRS	Deleting Connection	
Description	This command deletes an existing connection, removing it from the DiamondWave equipment database.	
Input format	DLT-CRS:[TID]:[<srcPort>,<dstPort>]:[CTAG]::<circuitId>; example : agent> dlt-crs:::::0.3a.4-0.4a.3; calient 01-07-25 17:39:08 M 0 COMPLD ;	
Input parameters	srcPort	This parameter specifies the port used for the connection. *srcPort* must be specified.
	dstPort	This parameter specifies the port used for the destination. *dstPort* must be specified.
	circuitId	This parameter specifies the connection ID to delete. *circuitId* must be specified.
Output parameters	*respCode*	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
Error Codes	IDNV	Customer Name or Circuit does not exist
	IIFM	1. Invalid format of customer_name or circuitId string 2. Invalid format of eqptId
	RCIN	Requested CircuitID does not exist

Details:

The deletion of a cross-connection is also completed in less than one second and the responses are displayed back to the TLI agent prompt immediately. It is also important to know that at least a pair of the circuit id, groupName or connName must be used in the TL1 command

D.2.1.3 *Reserve XC*

This command is used to reserve ports for cross connections which could be activated sometime in the future

ACT-CRS	Activating Connections
Description	This command reactivates a cross connection that had previously been deactivated by the CANC-CRS command. This command moves the connection from an under management state (AS_UMA) to an in-service state (AS_IS).
Input format	ACT-CRS:[TID]:[<srcPort>,<dstPort>]:[CTAG]::<circuitId>;

	example : agent>act-crs:::::0.3a.1>0.4a.1; calient 01-08-01 10:00:44 M 0 COMPLD ;	
Input parameters	srcPort	This parameter specifies the port used for the connection. *srcPort* must be specified.
	dstPort	This parameter specifies the port used for the destination. *dstPort* must be specified.
	circuitId	This parameter specifies the connection ID to delete. *circuitId* must be specified.
Output parameters	*respCode*	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
Error Codes	IDNV	Customer Name or Circuit does not exist
	IIFM	Invalid format of customer_name or circuitId string
	RCIN	Requested Circuit ID does not exist

Details:

The reservation of a cross-connection is also completed in less than one second and the responses are displayed back to the TLI agent prompt immediately. It is also important to know that at least a pair of the circuit id, groupName or connName must be used in the TL1 command

D.2.1.4 *Unreserve XC*

This command is used to unreserve ports that has been previous reserved or to deactivate and existing cross connection. Although the existing cross connection is deactivated, it is not deleted from the system.

CANC-CRS:	Cancelling Connections
Description	This command cancels (deactivates) a previously active connection, moving the connection state from in-service (AS_IS) to under management (AS_UMA). While in an AS_UMA state, the connection still functions normally; however, alarms are not logged. When a connection is deactivated, all outstanding alarms associated with the connection are cleared.

Input format	CANC-CRS:[TID]:[<srcPort>,<dstPort>]:[CTAG]::<circuitId>; example : agent>canc-crs:::::0.3a.4>0.4a.3; calient 01-08-23 13:02:56 M 0 COMPLD ;	
Input parameters	srcPort	This parameter specifies the port used for the connection. *srcPort* must be specified.
	dstPort	This parameter specifies the port used for the destination. *dstPort* must be specified.
	circuitId	This parameter specifies the connection ID to delete. *circuitId* must be specified.
Output parameters	*respCode*	CMPLD – Completed successfully, DENY – Action denied, DELAY – Successful delayed action activation, PRTL – Partially successful response, RTRV – multiple parts successful response (last part with CMPLD).
	IDNV	Customer Name or Circuit does not exist
	IIFM	Invalid format of customer_name or circuitId string
	RCIN	Requested CircuitID does not exist

Details:

The deactivation of a cross-connection is also completed in less than one second and the responses are displayed back to the TLI agent prompt immediately. It is also important to know that at least a pair of the circuit id, groupName or connName must be used in the TL1 command

D.2.1.5 *Event Notification*

The event notification is handled by parsing the autonomous messages that are received from the switch . These Autonomous messages are used to report alarms, configuration changes, or condition changes. Many of these messages, such as those relating to alarm conditions, are spontaneously triggered by the NE itself without intervention.

REPT	Report Alarm

ALM:	
Description	This message cannot be issued from the console. It is displayed on the console after receipt from the TL1 agent.
Output format	Example: agent 02-12-04 13:01:35 ** 15 REPT ALM ENV "0.11b:MJ,T-ADC,NSA,02-12-04,13-01-35,,:\"AlarmId=42: Description=ADC Bus Errors detected\"" ;;

REPT DBCHG:	Report Database Change
Description	This message cannot be issued from the console. It is displayed on the console after receipt from the TL1 Agent. This autonomous message reports immediately to the operational service DiamondWave database changes that have occurred as a result of commands to change - equipment provisioning or configuration - the value of the TID or SID - the value of the keywords defined in the common block or specific block
Output format	Example: agent> set-sid:::::calient; calient 02-12-04 13:01:14 A 11 REPT DBCHG "DATE=02-12-04,TIME=13-01-14,SOURCE=0, USERID=admin,DBCHGSEQ=7:SET-SID:calient" ;

	Report Event Messages
Description	There are two types of event messages: **- REPT EVT SECU: Report Event Security** **- REPT EVT COM: Report Event Commons** These messages cannot be issued from the console. They are displayed on the console after receipt from the TL1 agent. These autonomous messages result in a display of a DiamondWave event on the console. For example,
Output format	Examples: calient 02-11-25 11:18:25 A 350 REPT EVT SECU "admin:SEC-LOGON,TC,02-11-25,11-18-25,,,,:\"User login\"" ; calient 02-12-04 13:01:35

	A 14 REPT EVT COM "0.11b:MON-MJ,TC,02-12-04,13-01-35,,,,:\"Monitor major threshold crossed\"" ;

D.2.1.6 *Get Resource List*

This command probes for the available resources on the Calient OXC. Although the Calient uses single commands for the probing of lists such as XCs and alarms, there is no single command that probes for the amount of available resources and their current states. To support this functionality we use a loop to retrieve individual ports and the cumulative result of the loop is presented to the TNRC_AP.

RTRV-PORT	**Retrieving Port Information**	
Description	This command retrieves port information.	
Input and Output format	*Input:* RTRV-PORT:[TID]:<eqptId>:[CTAG]::[<owner>],[<portcategory>]; *Output:* SID DATE TIME M CTAG COMPLD "<AID>:<portType>,<inOwner>,<outOwner>: [INOPTDEGR=<inoptdegr>], [INOPTCRIT=<inoptcrit>], [OUTOPTDEGR=<outoptdegr>], [OUTOPTCRIT=<outoptcrit>], [INOPTHI=<inopthi>], [ATTNMODE=<attnmode>], [OUTPOWER=<outpower>],[VARIANT=<variant>],[ALIAS=<alias>], [IN_AS=<inAS>],[IN_OS=<inOS>],[IN_OC=<inOC>], [OUT_AS=<outAS>], [OUT_OS=<outOS>], [OUT_OC=<outOC>]" ; example : agent> rtrv-port::0.18.1; TL1AGENT 04-12-08 00:31:05 M 0 COMPLD "0.18.1:OAONR,TRANSIT, NONE:INOPTDEGR=-15.00, INOPTCRIT=-18.00,OUTOPTDEGR=-23.00, OUTOPTCRIT=-26.00,INOPTHI=13.00,, ALIAS=TEST,POWERMODE=CONSTOUTPUT,OUTPOWER=0.00,VARIANT=0.50,INAS=IS,INOS=RDY,INOC=OK,OUTAS=OOS-NP, OUTOS=OOS, OUTOC=OK, PORTID=1205761" ;	
Input parameters	eqptId	This parameter specifies the port ID to modify. For example, 0.13a.
	owner	This parameter specifies the ownership of connection. Options are: - trib (tributary) indicates the port is used in an optical network connection - none indicates the port is used in a local node cross connection
	portcategory	This parameter specifies the port category. Options are: - all displays all ports - free displays the ports that are not used in any

		connection
Output parameters	PORTTYPE	This parameter specifies the type of the port (card type).
	INOPTDEGR	This parameter specifies the input optical power threshold.
	INOPTCRIT	This parameter specifies the input optical power monitor critical threshold. The range is –35 dBm to 15 dBm. Default is –18.0 dBM.
	OUTOPTDEGR	This parameter specifies the output optical power monitor degraded threshold for the light through the active switch matrix. The range is –35 dBm to 20 dBm. Default is –23.0 dBM.
	OUTOPTCRIT	This parameter specifies the output optical power monitor critical threshold for the light through the active switch matrix. The range is –40 dBm to 15 dBm. Default is –26.0 dBM.
	INOPTHI	This parameter specifies the input optical power monitor high threshold for the light through the active switch matrix. The range is –10 dBm to 20 dBm. Default is 20.0 dBM.
	ATTNMODE	This parameter specifies the attenuation mode for the VOA application.
	OUTPOWER	For IO cards with a power gain feature, this parameter specifies the target output power for the port. This parameter is applicable only to a port configured as constant output. Setting is in increments of +dBm based on the granularity setting of the variant. The range is -30dBm to 15dBm. For example, the setting 1.5 increases optical power gain 1.5dBm. *outpower* is optional.
	VARIANT	This parameter specifies the attenuation threshold ranging between 0–15 dB. The default is 0.5 dB. The range is 0.5dB to 10dB. *variant* is optional.
	ALIAS	This parameter specifies an assumed name created for the port.
	INAS	This parameter specifies the input administrative state.
	INOS	This parameter specifies the input operational state.
	INOC	This parameter specifies the input operational capability.
	OUTAS	This parameter specifies the output administrative state.
	OUTOS	This parameter specifies the output operational state.
	OUTOC	This parameter specifies the output operational capability.

D.2.1.7 *Get Resource Details*

The command retrieves the properties of a particular resource. The same command for the get resource detail as explained earlier is used but without the loop.

D.2.1.8 *Flush Resources*

At the moment, this command is not quite clear. We are not sure, if it is to clear all alarms in the system or it is to stop the notifications of events. This is been flagged to be discussed within the group.

D.2.2 TNRC_SP_Calient Generic Descriptions

The diagram below provides an overview of the processes and threads that were implemented in the Calient TNRC_SP software. The software is made up two major threads in which one is used for listening and the other for interacting with the switch.

The diagram in Figure 16-12. also shows the structure and the integration of the functions recommended in the TNRC_SP specification document. As explained earlier on we are using the TL1 command language integrated with the Telnet communication interface. The TNRC_SP architecture is divided into four broad categories described below.

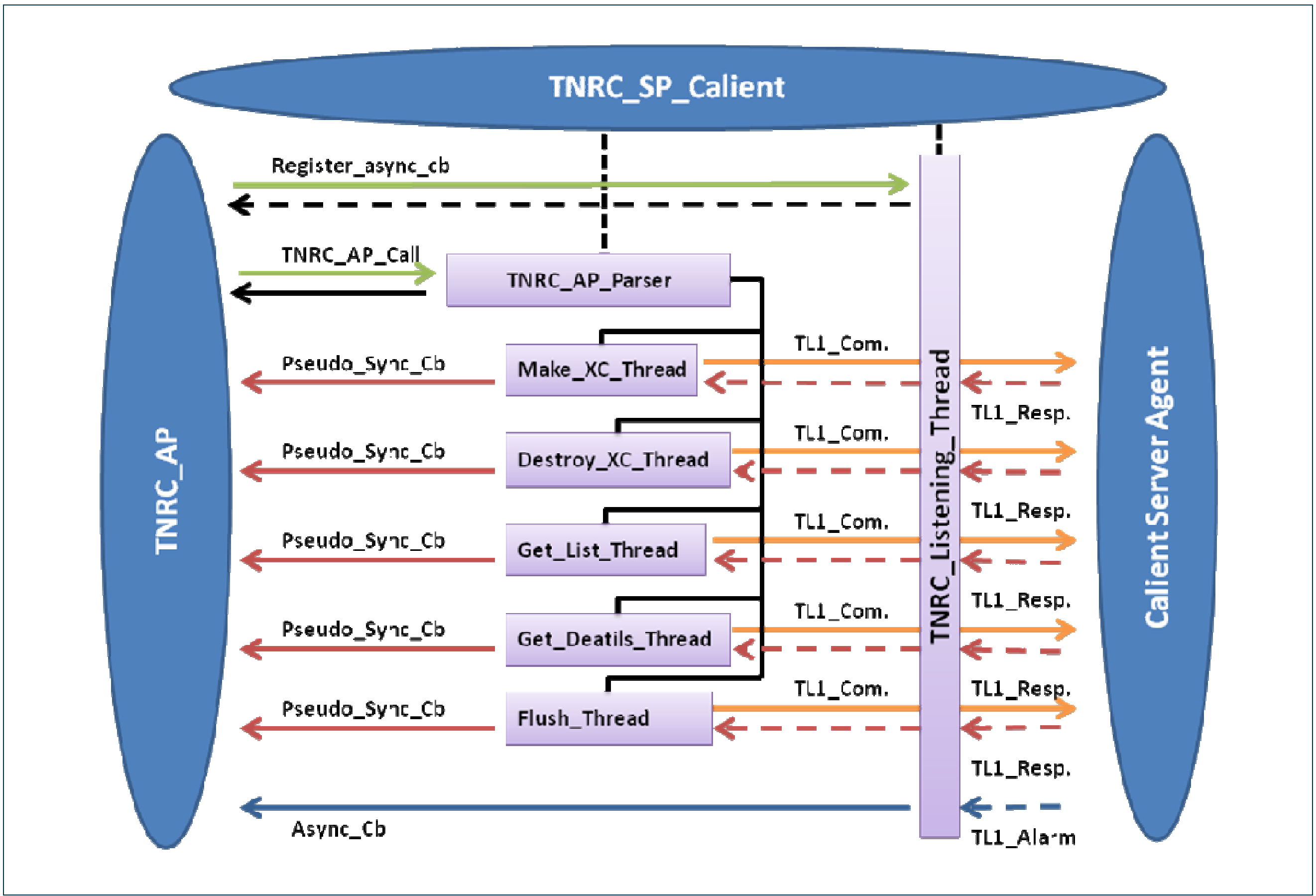

Figure 16-12: Process and threads sequential diagram.

- **On start**: TNRC_AP creates an instance of TNRC_SP_Calient
  - Each instance :
    - — Establishes a Telnet client session for life time of the instance.
    - — Implements a Telnet client listener thread (TNRC_Listening_Thread) for life time of the instance.
    - — Implements TNRC_AP_Parser method.
    - — To parse TNRC_AP commands.
  - Up to 8 concurrent instances can be created. This is because the Calient Telnet server is limited to only 8 parallel sessions.
- **On process**: each TNRC_AP_Call (i.e. Make_XC,Destroy _XC, Reserve XC, Unreserve XC) calls TNRC_Parser method:
  - Sends Ack to TRNC_AP.
  - Creates an independent thread for each call.
  - Each thread :
    - — Sends associated Telnet command with a unique tag.
    - — Starts a “no response” timer.

— Waits for Ack from TNRC_Listening_Thread.
— Implements pseudo-synchronous notification method:
(1) On “no response” timer expiry notify TNRC_AP.
(2) On Ack from TNRC_Listening_Thread notify TNRC_AP.
(3) On Nack from TNRC_Listening_Thread notify TNRC_AP.
— Thread dies after notification or timeout.

The state diagrams for the commands are shown in Figure 16-13, Figure 16-14, Figure 16-15 and Figure 16-16.

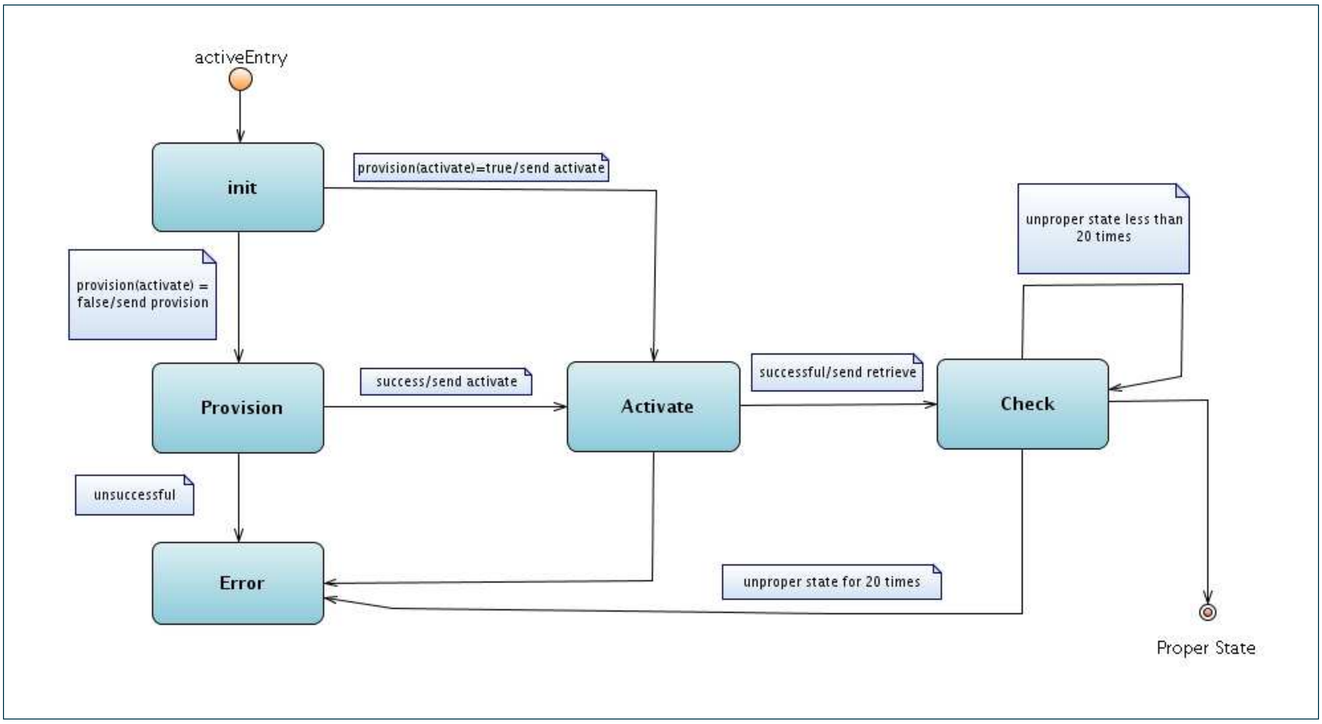

Figure 16-13: State Diagram for Make XC.

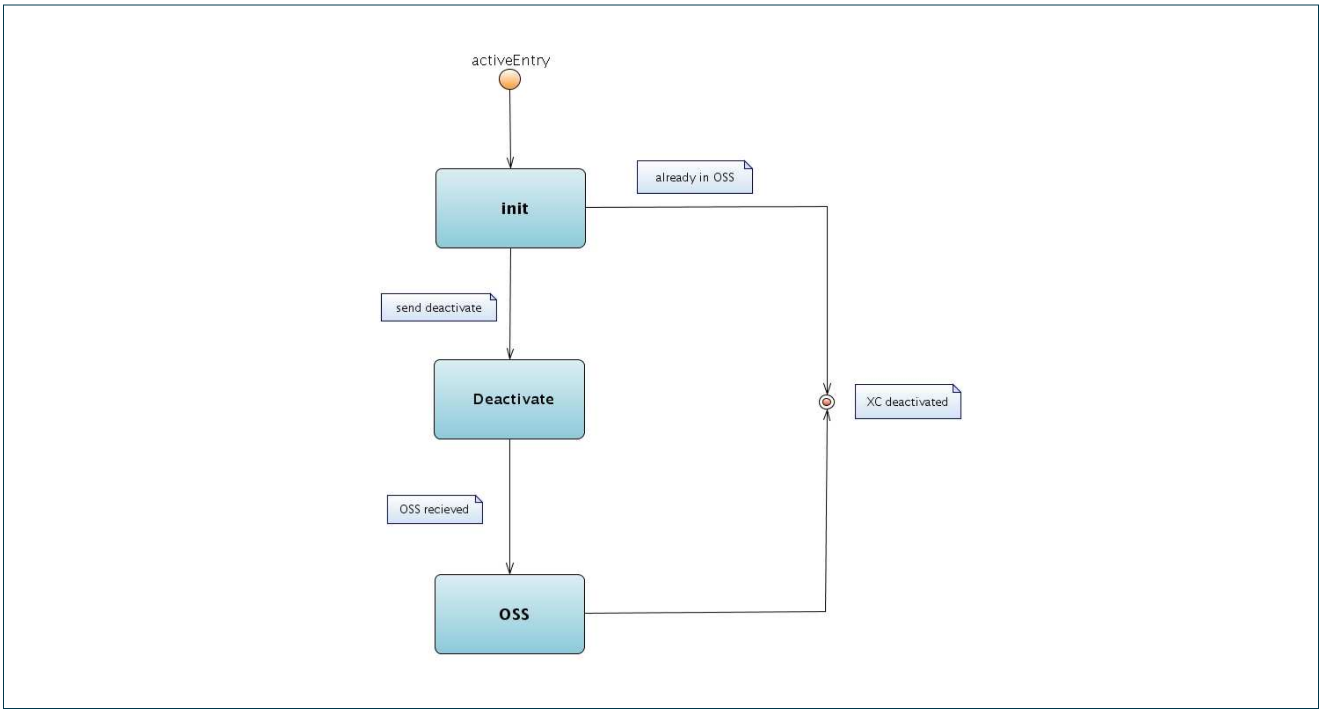

Figure 16-14: State Diagram for Destroy XC.

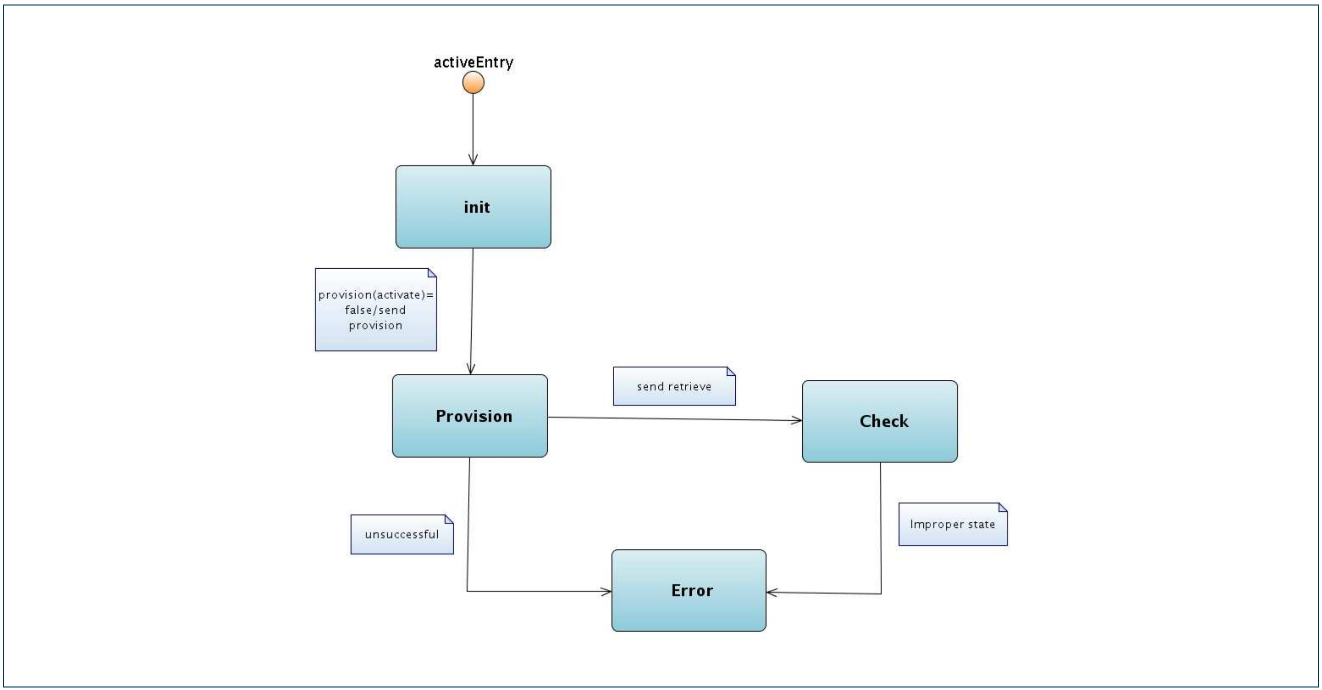

Figure 16-15: State Diagram for Reserve Resources.

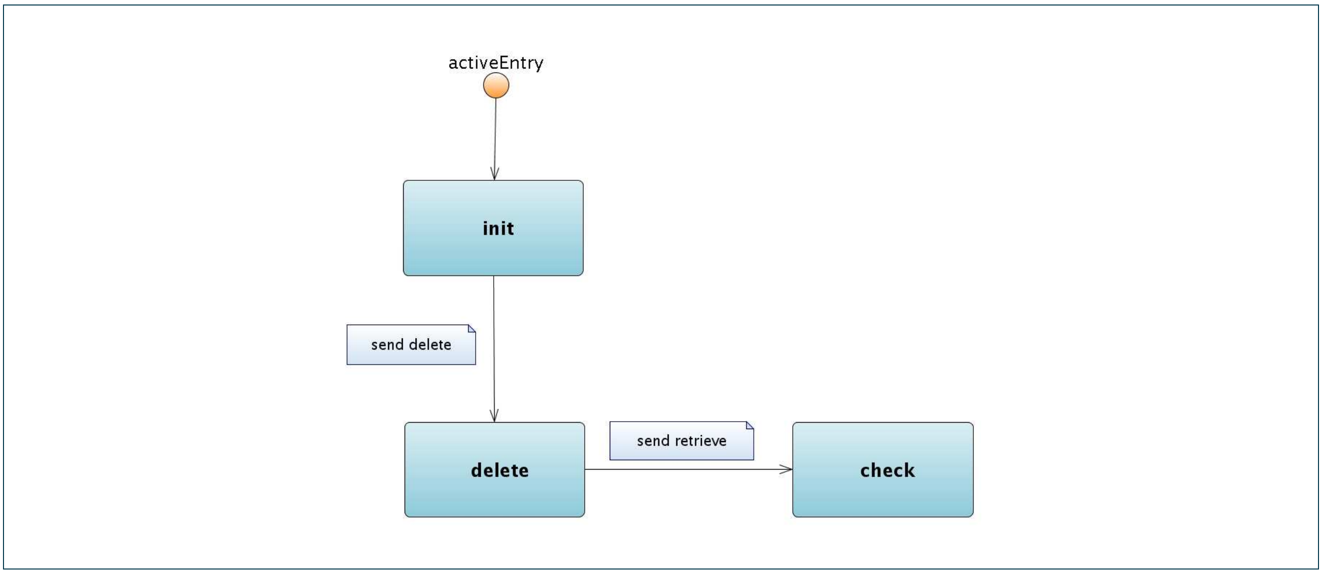

Figure 16-16: State Diagram for Un-reserve Resources.

- **On process**: register_async_cb initiates the event notification registration:
  - Registers the associated events.
  - Sends Ack to TRNC_AP.

- **On Process**: Telnet client listener thread (TNRC_Listening_Thread) monitors (listens) Telnet client socket:
  - TNRC_Listening_Thread:
    - — Listens to message broadcasted by Telnet server agent in Calient.
    - — Implements a message parsing method:
    - (1) Lookup for registered events/alarms.
    - (2) Notify TNRC_AP with the registered events (register_async_cb).
    - (3) Send call each response to its associated (tag) source thread.

<END-OF-DOCUMENT>