.

Information Technology Laboratory
ITL Advanced Network Technologies Division
NIST National Institute of Standards and Technology

RSVP – TE Modeling Specification

Message and Data Structure

Chul Kim
Guest Researcher ANTD
E-mail : goldfe@nownuri.net
Alternative: chulkim@antd.nist.gov
Created Date: 2002/4/12
Last Update: 2002/6/21
Current State: Draft

Blank

Contents

1. RSVP Message Definition 5
2. RSVP Object Modeling 6
   - 2.1 SSFNET Protocol Message 6
   - 2.2 RSVP Objects 6
     - 2.2.1 Payload 7
     - 2.2.2 Object Format 7
     - 2.2.3 Session Object 8
     - 2.2.4 RSVP_HOP Object 9
     - 2.2.5 TIME_VALUES Object 9
     - 2.2.6 STYLE Object 10
     - 2.2.7 TSpec Object 10
     - 2.2.8 Sender Object 10
     - 2.2.9 ERROR_SPEC Object 11
     - 2.2.10 SCOPE Object 11
     - 2.2.11 RESV_CONFIRM Object 12
     - 2.2.12 Flow_Descriptor Object 12
     - 2.2.13 Sender_Descriptor Object 12
   - 2.3 RSVP-TE Extensions to RSVP for LSP Tunnels 12
     - 2.3.1 Label Object 13
     - 2.3.2 Label Request Object 13
     - 2.3.3 Explicit Route Object / Record Route Object 14
     - 2.3.4 LSP_TUNNEL_IPv4/6 Session, Sender Template, Filter Spec Object 15
   - 2.4 RSVP - GMPLS TE Extension 16
     - 2.4.1 Generalized Label Request Object 16
     - 2.4.2 Generalized Label Object 17
     - 2.4.3 Waveband Switching Object 17
     - 2.4.4 Label Set Object 17
     - 2.4.5 Notify Request Object 18
     - 2.4.6 ERO / RRO Sub-object 18
3. Generic Data Structure 19
   - 3.1 Path State Block 19
   - 3.2 Reservation State Block 20
   - 3.3 Traffic Control State Block 20
   - 3.4 Blockade State Block 21

1. RSVP Message Definition

Table 1 shows the RSVP messages and message format. By now we does not support the [RFC 2961] RSVP Refresh Overhead Reduction Extension.

Table 1 RSVP Message and Message Format

Message Type	Message Format	
Path	<Common Header> <Session> <RSVP_HOP> <Time Value> [<Explicit Route>] <Label Request> [<Session Attribute>] [<Notify Request>] <sender descriptor>	<sender descriptor> ::= un-diirection <Sender Template> <Sender TSpec> Bi-directional <Sender Template> <Sender Tspec> <Upstream Label>
Resv	<Common Header> <Session> <RSVP_HOP> <Time Value> [<Resv Confirm>] [<Scope>] [<Notify Request>] <Style> <flow descriptor list>	<flow descriptor list> ::= <FF flow desc list><SE flow desc> <FF flow desc list> ::= <Flow spec><filter pec><label> [<Record Route>] <SE flow desc > ::= <flow spec><filter spec><label> [<Record Route>]
PathTear	<Common Header> <Session> <RSVP_HOP> <sender descriptor>	
ResvTear	<Common Header> <Session> <RSVP HOP> [<Scope>] <Style> <flow descriptor list>	
PathErr	<Common header> <Session> <Error Spec> <sender descriptor>	
ResvErr	<Common Header> <Session> <RSVP_HOP> <Error Spec> [<Scope>] <Style> <error flow descriptor>	
ResvConf	<Common Header> <Session> <Error Spec> <Resv Confirm> <Style> <flow descriptor list>	
Hello	<Common Header> <Hello> [<Restart Cap>]	
Notify	<Common Header> <Error Spec> <Notify Session List>	

2. RSVP Object Modeling

2.1 SSFNET Protocol Message

In SSFNet, all protocol messages are modeled as **ProtocolMessage** class. More information can be found in SSFNet site. But in this document we reverse engineers the ProtocolMessage to show the variables and functions of this class. Fig. 1 shows the class diagram of the ProtocolMessage Class.

```
ProtocolMessage
(from OS)

◆ProtocolMessage( )
◆size( ) : float
◆version( ) : String
◆fromVersion(V : String) : ProtocolMessage
◆dropPayload( ) : void
◆dropHeader( ) : void
◆copy( ) : ProtocolMessage
◆carryPayload(payload : ProtocolMessage) : void
◆payload( ) : ProtocolMessage
◆previous( ) : ProtocolMessage
◆bytecount( ) : int
◆header_bytecount( ) : int
◆tobytes(buf : byte[], offset : int) : void
◆frombytes(buf : byte[], offset : int) : void
```

Fig. 1 ProtocolMessage Class

2.2 RSVP Objects

In order to model RSVP message, it is necessary to know the RSVP format. RSVP message consists of a common header, followed by a body consisting of a variable number of variable-length, typed **Object.** Fig. 2 shows the common header format of RSVP Message.

```
        0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Flags|  Msg Type   |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|  Send_TTL   | (Reserved)  |        RSVP Length        |
+-------------+-------------+-------------+-------------+
```

Fig. 2 Common Header

The **Message** class models a common header. Fig. 3 shows the Class diagram that contains the member function and variables. This class contains the RSVP message generation function that creates the messages. To call these functions freely we model this function **static** function. So, without instantiation we can call these functions to generate message.

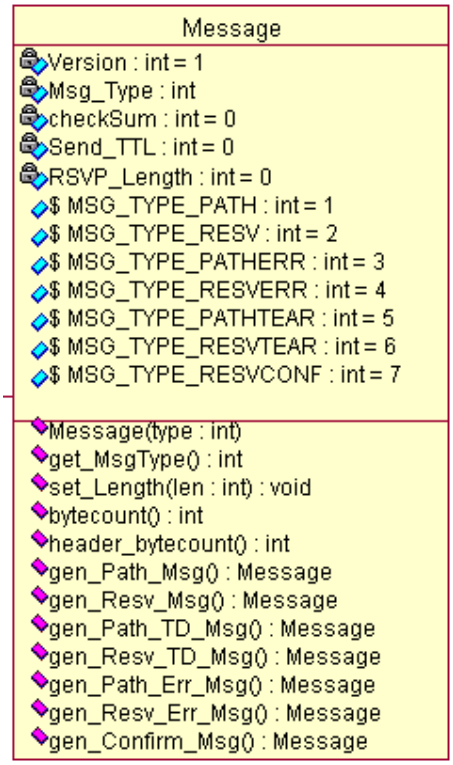

Fig. 3 Message Class

2.2.1 Payload

Basically, Payload consists of variable number of object. To support such capability we introduce the Payload class that contains the variable number of objects. To accommodate the variable number of objects we use the Vector class. This class will be carried by the Message class.

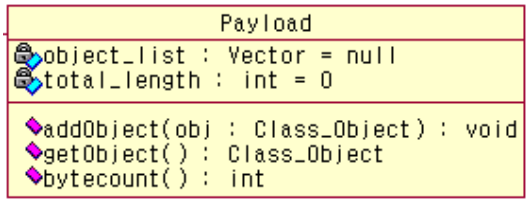

Fig. 4 Payload class

2.2.2 Object Format

Every RSVP message consists of variable number of variable-length, typed **objects**. Every object consists of one or more 32-bit words with a one-word header, with the following format.

```
        0             1             2             3
+-------------+-------------+-------------+-------------+
|       Length (bytes)      |  Class-Num  |   C-Type    |
+-------------+-------------+-------------+-------------+
|                                                       |
//                  (Object contents)                  //
|                                                       |
+-------------+-------------+-------------+-------------+
```

Fig. 5 Object format

To model the object format, we introduce the abstract class **Class_Object** that is parent class of all object classes. This class contains the length, class-num, and c-type as member variables and it also has some utility functions such as print_object(). This class also defines the constant variables that indicate class name and class type.

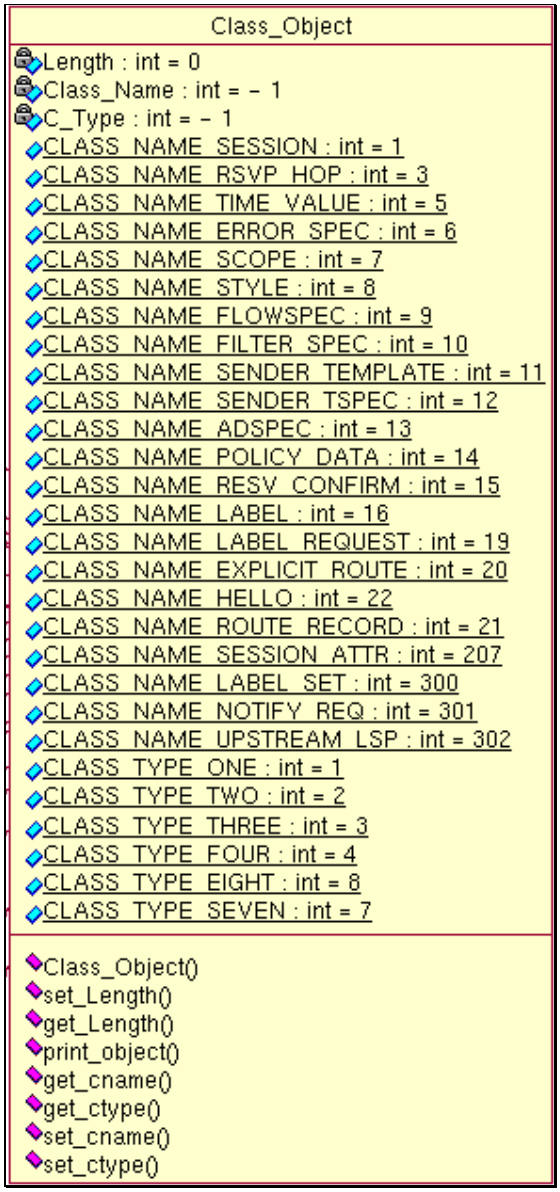

Fig. 6 Class_Object Class

2.2.3 Session Object

Session Object contains IP destination address of destination node, IP protocol id, and generalized destination port to define a specific session for the other object that flow. This will be shown in every RSVP message. The Session Object is modeled as **Session_Object** class. This is also extended to accommodate the LSP tunnel extension that is used to LSP setup with RSVP signaling.

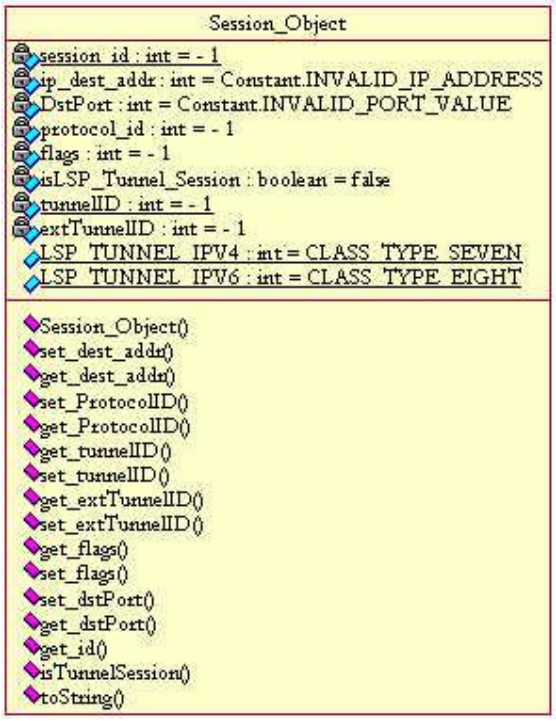

Fig. 7 Session_Object Class

2.2.4 RSVP_HOP Object

RSVP_HOP object carries the IP address of the RSVP-capable node that sent this message and a local outgoing interface handle (LIH). This object is referred as a PHOP(previous hop) object for downstream message or as a NHOP(next hop) object for upstream message.

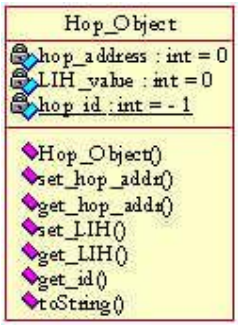

Fig. 8 Hop_Object class

2.2.5 TIME_VALUES Object

This object contains the value for the refresh period R used by the creator of the message. This information required in every Path and Resv message.

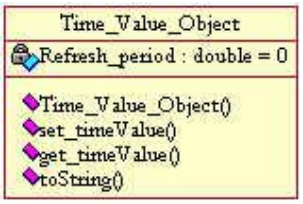

Fig. 9 Time_Value_Object Class

2.2.6 STYLE Object

This object defines the reservation style plus style-specific information that is in FLOWSPEC or FILTER_SPEC object and it is required in every Resv message. The RSVP defines four reservation styles, but only three styles, WF, FF, and SE are defined for signaling purpose.

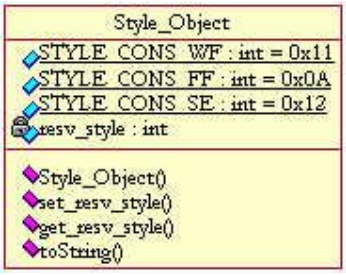

Fig. 10 Sytle_Object Class

2.2.7 TSpec Object

Because FLOWSPEC and SENDER_TSPEC object has the same information we models it with TSpec Object. Two objects define the QoS parameters delivered to the sender such as token bucket size and token bucket rate, etc.

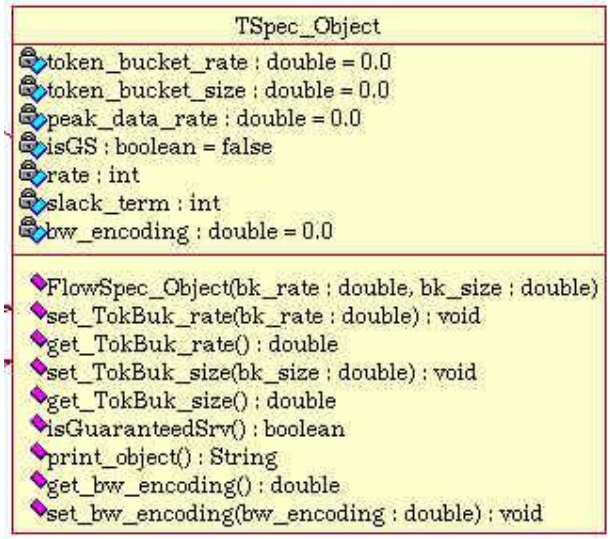

Fig. 11 TSpec_Object class

2.2.8 Sender Object

Since two objects, Filter Specification and Sender Template, contain similar parameters we models these object as one class, Sender Object.. This object presents a subset of session data packets that should receive the desired QoS (specified by a FLOWSPEC object), in a Resv message and contains a sender IP address.

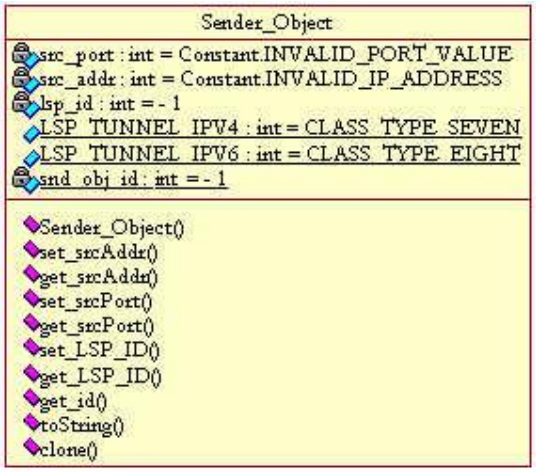

Fig. 12 Sender_Object Class

2.2.9 ERROR_SPEC Object

This object specifies an error in a PathErr, ResvErr, or a confirmation in a ResvConf message.

Fig. 13 Error_Spec_Object class

2.2.10 SCOPE Object

This object carries an explicit list of sender hosts towards which the information in the message is to be forwarded.

This may appear in a Resv, ResvErr, or ResvTear message.

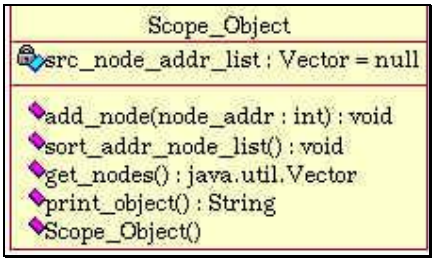

Fig. 14 Scope_Object class

2.2.11 RESV_CONFIRM Object

This carries the IP address of a receiver that requested a confirmation. This may appear in a Resv or ResvConf message.

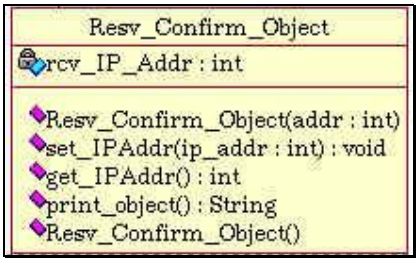

Fig. 15 Resv_Confirm_Object Class

2.2.12 Flow_Descriptor Object

A flowspec together with a filter spec are called a flow descriptor. This pair is an essential element in many RSVP messages and other general data structures. So, we model it **Flow_Descriptor** class. This class has **Flow_Desc_Object** class as a member variable and several helper functions that handle the member variables in efficient way. The Flow_Desc_Object consists of the following parameters: Label_Object, Sender_Object, ER_RR_Object, and TSpec_Object.

2.2.13 Sender_Descriptor Object

This object is not regular object of RSVP. This class represents the sender descriptor object that consists of Sender_Object and TSpec_Object. This class is shown in the many RSVP message frequently. This class has sender template and sender tspec object as a member variable and several helper function that handle the two objects.

2.3 RSVP-TE Extensions to RSVP for LSP Tunnels

In order to support LSP Tunnel to RSVP it is required TE extension to RSVP. The TE extension for LSP tunnels contains newly defined message object and re-defined RSVP object. In following we describe the newly defined or re-defined message object. The re-defined message objects are preferred when implementation.

2.3.1 Label Object

Label may be carried in Resv Message. For the FF and SF styles, a label is associated with each sender. The label for a sender must immediately follow the FILTER_SPEC for the sender in the Resv message. LABEL object has 16 for Class Name, 1 for Class Type.

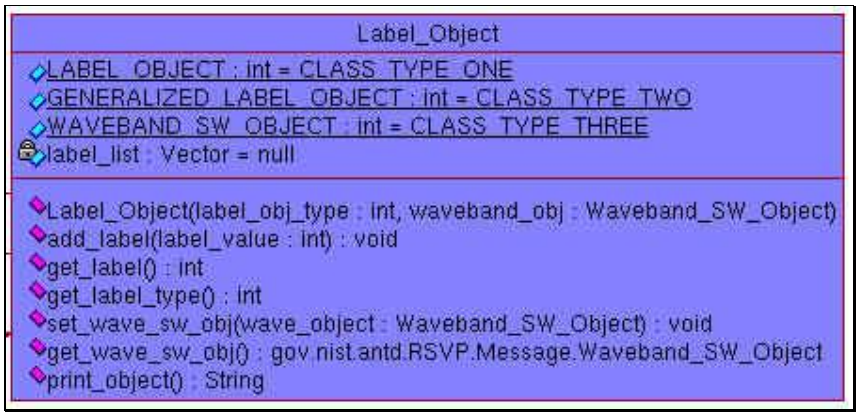

Fig. 16 Label Object

Fig. 16 shows the Label Object. The modeled class also contains the GMPLS-TE extension. The additional part of GMPLS-TE extension is discussed in RSVP-GMPL TE extension part.

2.3.2 Label Request Object

The Label Request class name is 19. Currently there are three possible C_Type. Type 1 is a Label Request without label range. Type 2 is a label request with an ATM label range. Type 3 is a label request with a Frame Relay label range.

Although current SSFNet does not support ATM or FR model we model this object for the future implementation. RFC3209 define three Label request object: Without Label Range, with ATM Label Range, and with FR Label Range. In order to model these object with OOP concept, we model Label Request Object, parent class, and two objects are inherited from the parent class. Fig. 17 shows the parent class, Label Request Object. Two child objects are Label_Req_Obj_ATM and Label_Req_Obj_FR class. These classes are shown in Fig. 18,Fig. 19 respectively.

Fig. 17 Label Request Object

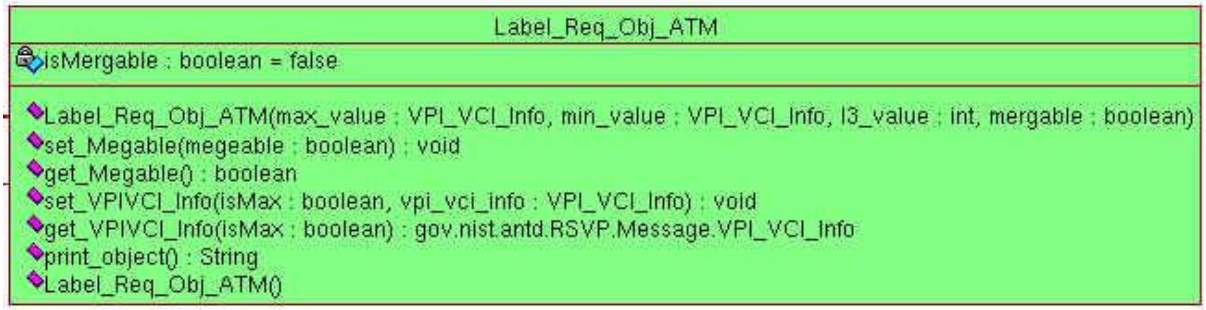

Fig. 18 Label Request Object with ATM Label Range

Fig. 19 Label Request Object with FR Label Range

2.3.3 Explicit Route Object / Record Route Object

Because the content of ERO and RRO is very similar with each other, we model it with one Class. Explicit routes are specified via the EXPLICIT_ROUTE Object (ERO). It has class Name 20 and one C-type is defined. While the contents of an EXPLICIT_ROUTE object are a series of variable-length data items called subobjects. Record Route object enumerates the node with Stack. Fig. 20 shows the ER_RR_Object. It contains the Vector and Stack as member variables to accommodate the Sub-objects. Sub-objects have common fields and have different fields to support IPv4, IPv6 and Label information. We model it with Route_Object class, shown in Fig. 21.

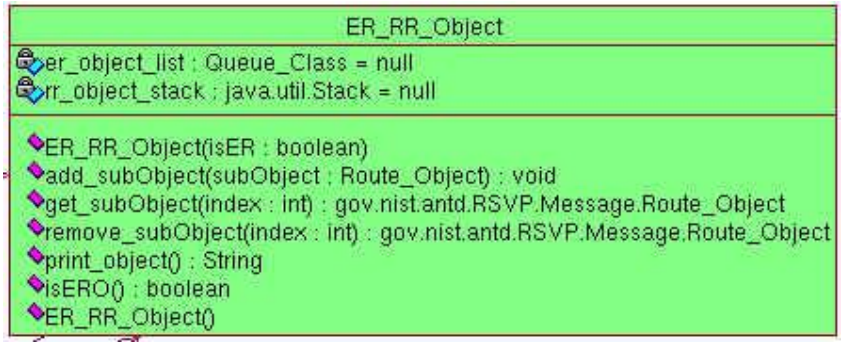

Fig. 20 Explicit Route / Record Route Object class

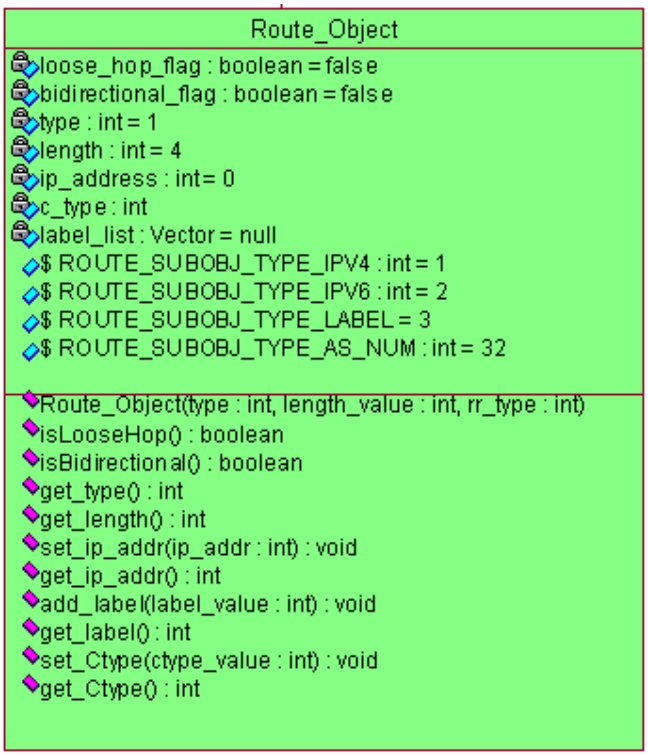

Fig. 21 Route Object Class

2.3.4 LSP_TUNNEL_IPv4/6 Session, Sender Template, Filter Spec Object

In order to support LSP Tunnel to RSVP and meet the requirements we have to extend Session, Sender Template, and Filter spec object. Because the format of filter spec object is identical to the sender template object we only show the sender template object.

```
Session_Object
ip_dest_addr : int = 0
protocol_id : int = 0
flags : int = 0
DstPort : int = 0
session_id : int = -1
isLSP_Tunnel_Session : boolean = false
tunnelID : int = -1
extTunnelID : int = -1
$ LSP_TUNNEL_IPV4 : int = CLASS_TYPE_SEVEN
$ LSP_TUNNEL_IPV6 : int = CLASS_TYPE_EIGHT

Session_Object(ip_addr : int, pro_id : int, dst_port : int)
set_dest_addr(dstAddr : int) : void
get_dest_addr() : int
set_ProtocolID(pro_id : int) : void
get_ProtocolID() : int
set_flags(flag : int) : void
get_flags() : int
set_dstPort(dst_port : int) : void
get_dstPort() : int
set_SessionID(Id : int) : void
get_SessionID() : int
print_object() : String
set_isTunnelSession(isTunnel : boolean) : void
get_isTunnelSession() : boolean
set_tunnelID(tunnel_id : int) : void
get_tunnelID() : int
set_extTunnelID(ext_tunnel_id : int) : void
get_extTunnelID() : int
```

Fig. 22 Session Object – Extended to support LSP Tunnel

```
Sender_Template_Object
src_addr : int = 0
src_port : int = 0
lsp_id : int = -1
isTunnelSendTemplate : boolean = false
$ LSP_TUNNEL_IPV4 : int = CLASS_TYPE_SEVEN
$ LSP_TUNNEL_IPV6 : int = CLASS_TYPE_EIGHT

Sender_Template_Object(isTunnel : boolean, addr : int, port : int)
set_srcAddr(srcAddr : int) : void
get_srcAddr() : int
set_srcPort(srcPort : int) : void
get_srcPort() : int
print_object() : String
is_TunnelLSP() : boolean
set_LSP_ID(lsp_id : int) : void
get_LSP_ID() : int
```

Fig. 23 Sender Template Object – Extended to support LSP Tunnel

2.4 RSVP - GMPLS TE Extension

To support Generalized MPLS RSVP-TE signaling is required to be extended. Because we already define RSVP, RSVP-TE signaling above, we design only GMPLS extension.

2.4.1 Generalized Label Request Object

Instead of creating new Generalized Label Request Object class we extend current Label Request Object. To extend Label Request Object we create General_Lable_Req_Info class that represents the General Label Request

Information..

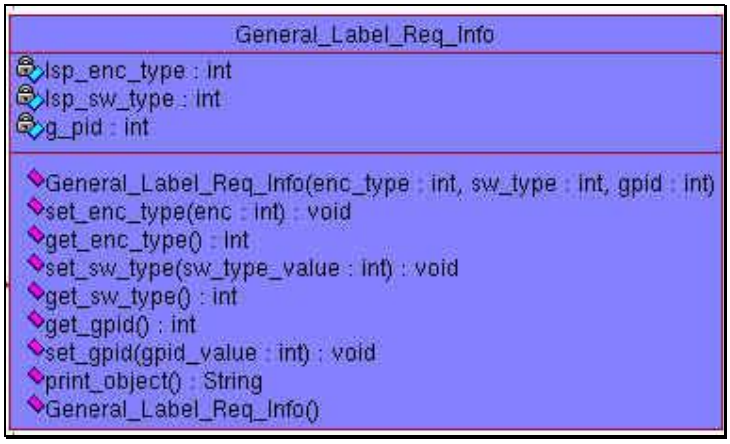

Fig. 24 General_Label_Req_Object Class model

2.4.2 Generalized Label Object

Generalized Label Object is identical to Label Object in 2.3.1. We show the modeled class diagram.

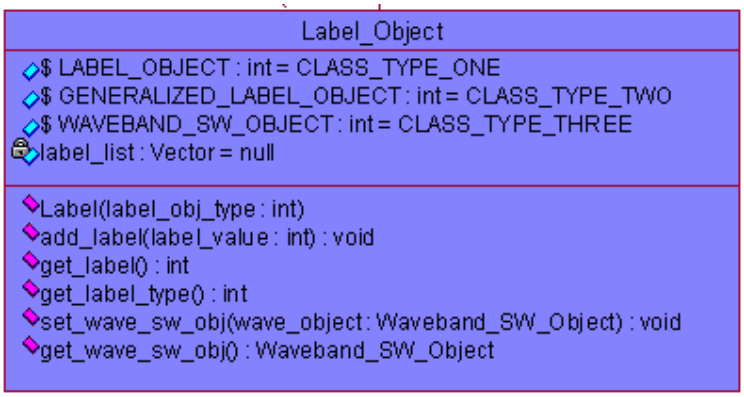

Fig. 25 Generalized Label Object

2.4.3 Waveband Switching Object

Waveband switching uses the same format as the generalized label. So we models the different part and make it class as shown in Fig. 26.

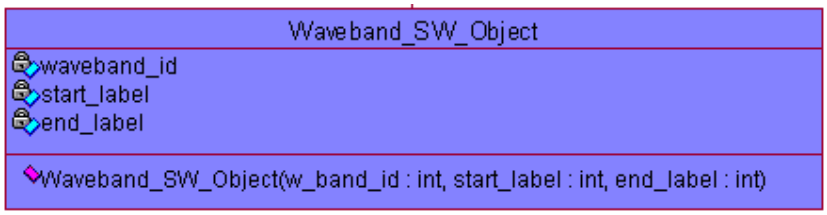

Fig. 26 Waveband Switching Object

2.4.4 Label Set Object

A Label Set object is defined as shown Fig. 27. It contains subchannel/Labels.

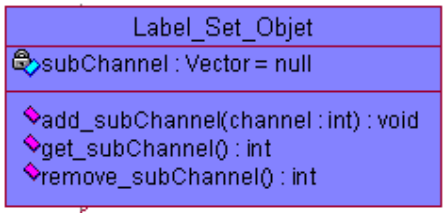

Fig. 27 Label Set Object

2.4.5 Notify Request Object

Notifications may be sent via the Notify message. The Notify Request object is used to request the generation of notifications. Notifications, i.e., the sending of Notify message, may be requested in both the upstream and downstream direction. The Notify Request Object may be carried in Path or Resv Message.

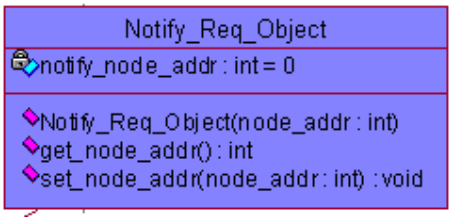

Fig. 28 Notify Req Object

2.4.6 ERO / RRO Sub-object

Refer section 2.3.3.

3. Generic Data Structure

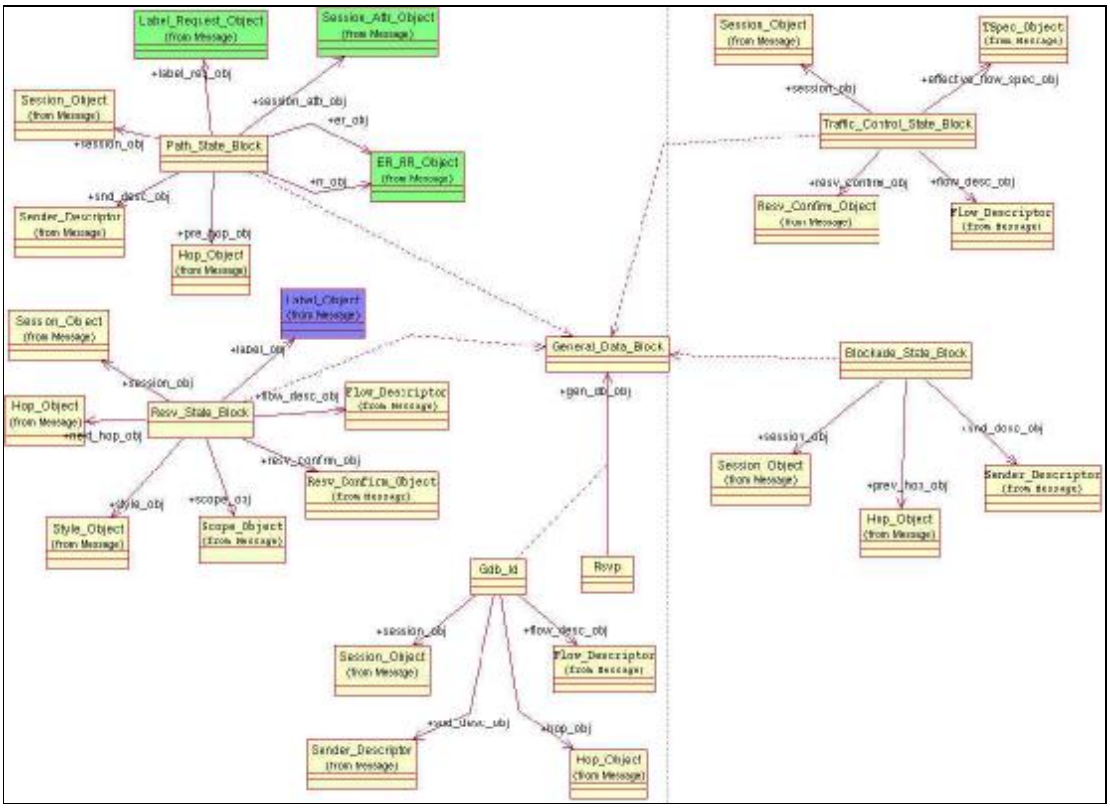

Fig. 29 Generic Data Structure Class Diagram

3.1 Path State Block

Each PSB holds path state for a particular (session, sender) pair, defined by SESSION and SENDER_TEMPLATE objects, respectively, received in a PATH message. Fig. 30 shows the modeled Path State Block.

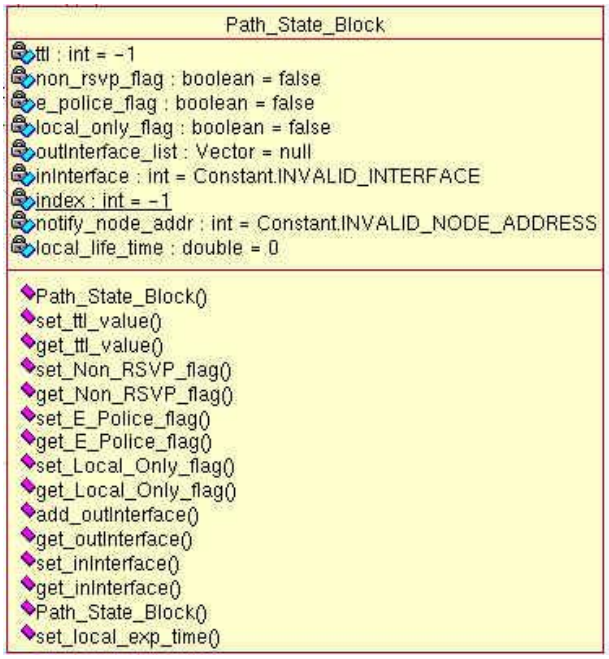

Fig. 30 Path State Block Class

3.2 Reservation State Block

Each RSB holds a reservation request hat arrived in a particular RESV message, corresponding to the triple: (session, next hop, Filter_Spec_list). Fig. 31 shows the Reservation State block class.

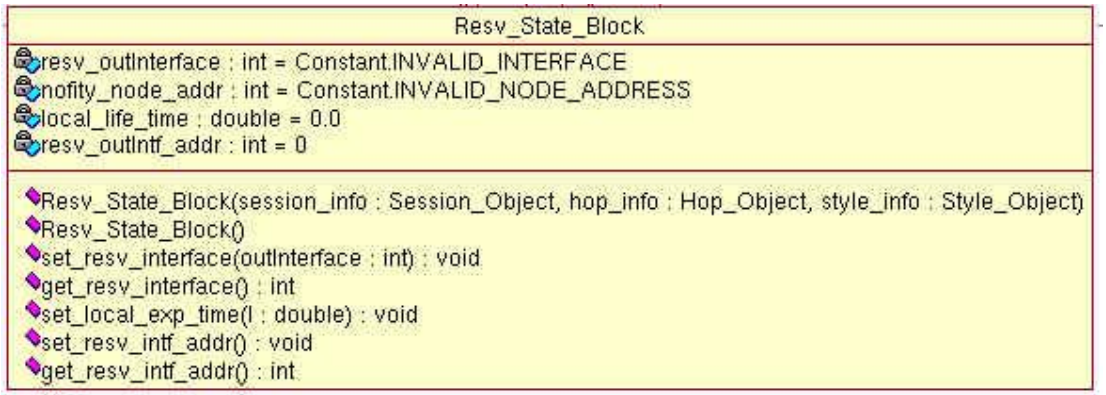

Fig. 31 Resv State Block Class

3.3 Traffic Control State Block

Each TCSB holds the reservation specification that has been handled to traffic control for a specific outgoing interface. In general, TcSB information is derived from RSB's for the same outgoing interface. Each TCSB defines a single reservation for a particular triple: (Session, Outinterface, Filter_spec_list).

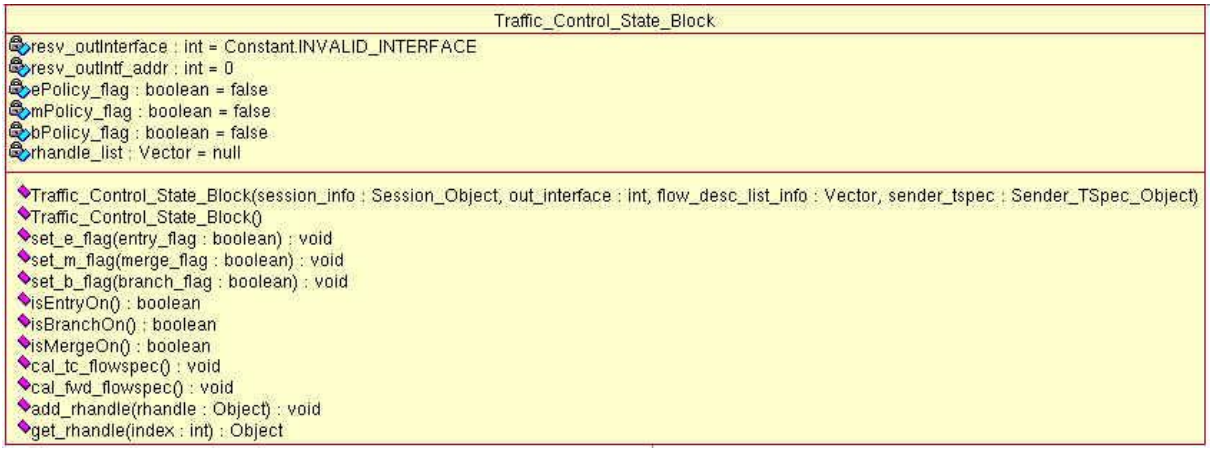

Fig. 32 Traffic Control State Block Class

3.4 Blockade State Block

Each BSB contains an element of blockade state. Depending upon the reservation style in use, the BSB's may be per (session, sender_template) pair or per (session, PHOP) pair.

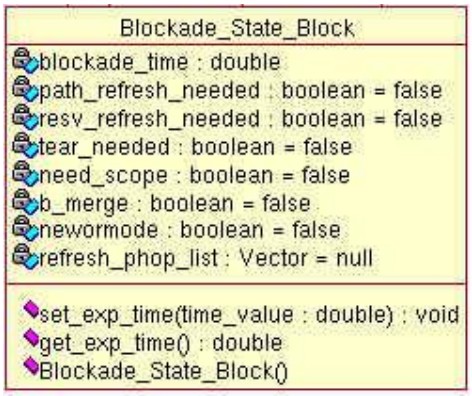

Fig. 33 Blockade State Block Class