INTEGRATED COMMUNICATIONS MANAGEMENT OF BROADBAND NETWORKS

Crete University Press, Heraklio, Greece
ISBN 960 524 006 8

Edited by David Griffin

Copyright © The ICM consortium, Crete University Press 1996

First published in 1996 by
CRETE UNIVERSITY PRESS
Foundation for Research and Technology
P.O. Box 1527, Heraklio, Crete, Greece 711 10
Tel: +30 81 394235, Fax: +30 81 394236
email: pek@iesl.forth.gr

Copyright © The ICM consortium, CUP 1996

The ICM consortium consists of the following companies:

Alcatel ISR, France
Alpha SAI, Greece
Ascom Monetel, France
Ascom Tech, Switzerland
Centro de Estudos de Telecommunicações, Portugal
Cray Communications Ltd., United Kingdom (Prime contractor)
Danish Electronics, Light & Acoustics, Denmark
De Nouvelles Architectures pour les Communications, France
Foundation for Research and Technology - Hellas, Institute of Computer Science, Greece
GN Nettest AS, Denmark
National Technical University of Athens, Greece
Nokia Corporation, Finland
Queen Mary and Westfield College, United Kingdom
Unipro Ltd., United Kingdom
University College London, United Kingdom
University of Durham, United Kingdom
VTT - Technical Research Centre of Finland

Chapter 9

Network adaptation issues

Editor: Michel Besson
Authors: Michel Besson, Peter Baxedale

In this chapter the most important aspects of the components adapting ATM Network Elements (NE) to the Q_3 interface required by the VPC and Routing management system are discussed.

The RACE II project R2061 EXPLOIT developed an ATM testbed, located in Basel, is organised around a kernel of several ATM switches interconnected through high speed links. Various classes of traffic can be generated within this network through different types of terminal equipment connected directly or indirectly via appropriate adaptors or interworking units.

9.1 Architecture of the Network Elements

Each Network Element (ATM switch) is connected to a subsystem controller (SSC - a UNIX work station) through a high speed link. The SSCs contain the software blocks responsible for controlling routing, connection acceptance, traffic policing and updating the connection tables.

For the purpose of managing the ATM switches, an agent implementing a Q-Adaptor Function (QAF) was developed and installed on each of the SSCs (see Figure 9.1). The agent is responsible for interacting with the software resources in the SSCs and representing these resources as managed objects through a TMN compliant interface.

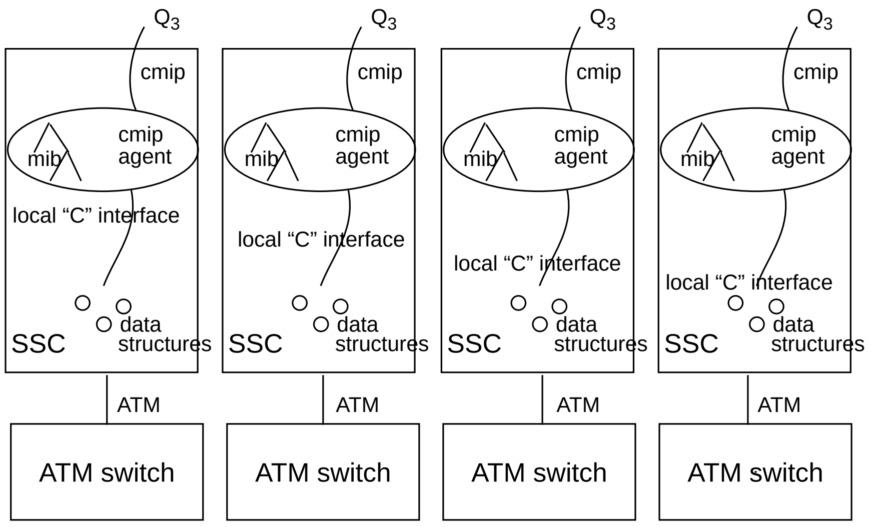

Figure 9.1 Functional architecture of the CMIP agent

Two approaches were considered for defining the MIBs to be presented to the upper, decision making levels of the TMN:

- *NE level*: This approach involved designing the NE MIB without considering the network-wide view. The key criteria were performance and ease of use. Careful attention was been paid to the use of GDMO facilities to structure information. This MIB is not based on the generic M.3100 [9.1] classes.
- *NEM level*: In this approach, the NE level is not modelled in isolation but with a wider view of both the NE and Network layers, with a specific attention to the configuration aspects of both NEs and network wide concepts (such as VPCs). The goal of this approach was to achieve an overall solution, parts of which would be mainly applicable to the Network layer and other parts to the Network Element layer, but embodying uniform and consistent modelling concepts. Consideration of the M.3100 [9.1] classes was the main criteria for this MIB.

The remainder of this chapter discusses these two MIBs without going into the details of their specification but rather by presenting the strong and weak points of both approaches. The *NEM level* MIB is presented first.

9.2 The NEM level MIB

The general approach was:

1. To use existing M.3100 classes when appropriate.
2. If not adequate, try to specialise them by deriving new classes.
3. If this cannot be done, propose changes to the standard classes to allow the required specialisation.
4. If this cannot be done, propose new classes.

9.2.1 Derived classes

In most cases, M.3100 classes could be used directly. In some cases it was necessary to derive new classes based on those from M.3100 but containing additional technology-specific attributes. An example is *linkTTPSource*. This represents a source port connected to a physical link. The bandwidth of the link and the number of VPs for which the port is configured (maximum number of VPs) is required, so the derived class *linkTTPSource* contains these additional attributes.

9.2.2 New classes

9.2.2.1 Routing tables

A Virtual Path/Virtual Channel ATM network has routing tables which identify paths to be taken (along particular VPCs) to reach a destination node. These tables are accessed and modified by the management system for route design and congestion control, and therefore they need to be represented in the MIB.

No standard classes could be found for modelling routing tables, so new classes were defined. These are *routeSelectionTable*, a container for routing table entries, and *routeSelectionEntry* for specifying a particular route. The *routeSelectionEntry* class was not designed to be generic, having technology specific attributes such as CoS (Class of Service). It could be redesigned to be more generic, possibly having a conditional package for some of the attributes, or deriving another more specialised class from a generic one. Attributes such as destination node, source pointer (showing which termination point source object to route to, in our case a *vpcTTPSource* object) and possibly route priority are probably fairly generic in nature.

9.2.2.2 CAC attributes

Attributes relating to the Connection Admission Control (CAC) - target cell loss probability and VPC allocated bandwidth - are required. These are technology (ATM) specific and therefore are not contained in M.3100. Rather than add these attributes to the derived classes representing VPC termination points, a new class *cacConfiguration* was defined. The use of a separate class allowed more flexibility in the design. The *cacConfiguration* class can be used to represent anything to related to CAC and can be instantiated wherever knowledge of the CAC is required. In our design we use different name bindings at the Network and NE levels. At the Network level, the CAC information is associated with a VPC and should therefore be contained by the *vpcTrail* object. But due to the design of the ETB network, CAC information is also required at each intermediate node, so the CAC data needs to be available at the start of each VPL. This can be achieved by instantiating a *cacConfiguration* object at each intermediate node along a VPC, contained by the relevant *vpcCTPSource* object (representing the VPL source).

A similar result could have been achieved by defining a *cacConfiguration* package, which could then be conditionally included in the *vpcCTPSource* and *vpcTrail* classes. We felt that this would have been a less elegant and less flexible approach.

9.2.2.3 Cross connections

Cross connections are used for connecting VPLs to build VPCs. A *vpcCTPSink* may be connected to an outgoing *vpcCTPSource*. Although M.3100 defines the *fabric* and *crossConnection* classes for dealing with these, they are not sufficiently generic and require attributes and actions not appropriate for ATM. New classes *atmFrabric* and *atmCrossConnection* are therefore defined which are very similar to the M.3100 classes and retain the basic principles but are more appropriate for ATM.

9.2.2.4 Statistical objects

New classes have also been defined for various objects used in the gathering of network statistics. These are technology specific and would not be expected to be found in the generic standards. The approach has been taken to use separate classes specifically for holding statistical data. Instances of these can then be created or deleted as required by monitoring applications and positioned appropriately in the containment tree.

9.2.3 Problems

The classes outlined above have been implemented without serious difficulties. At the Network layer, some problems were experienced in the dynamic creation and deletion of trails. There is no simple way to create an object such as a VPC from a remote manager, as is required by ICM. To construct a VPC one needs to create not only the trail object, which has pointers to the required connection objects (VPLs in this case), but also the connection objects themselves and all the trail and connection end points. This would be a very lengthy and involved process to carry out remotely. In fact, the only external information required is the name of the required VPC together with a list of the names of the physical links it is to traverse and the rest can be carried out internally. This suggests the need for a higher level action such as *createTrail* associated with the network class. An alternative approach, adopted by ICM, is to define a new class with a higher level of abstraction. This class we called *vpcTopology.* It contains a list of the links traversed by the VPC and hides the underlying detail.

Another problem experienced at the Network level is in remote access to the MIB to acquire certain information, again mainly relating to trails. If it is required, for example, to find the links over which a VPC passes, this information is not readily available. The trail object only contains pointers to the connection objects, and it is these objects which can be used to identify the physical links. This implies an extra level of indirection and extra remote accesses to get the information (first get the trail object, then get the connection objects it points to). To overcome this problem, in this particular example the *vpcTopology* class mentioned above can be used to provide the information directly. Alternatively, it may be possible to extract the information by careful use of scoping and filtering.

9.3 The NE level MIB

As mentioned before, the criteria used for the design of this MIB has been more the performance and “ease to access” aspects than the conformance to the M.3100 classes.

9.3.1 Usage of M.3100 “terminationPoint”

The main attribute used in the four *terminationPoint* classes from M.3100 is the *connectivityPointer* - a valid question is how consistent this is with the ATM technology. (Some other attributes are very specific to SDH technology, e.g. *supportableClientList*, which belongs to a conditional package, or are very generic such as the *operationalState* and as such they do not represent added value).

Each of these four classes contains an attribute, named specifically for each class, to handle the information about stream connectivity. They are illustrated Figure 9.2.

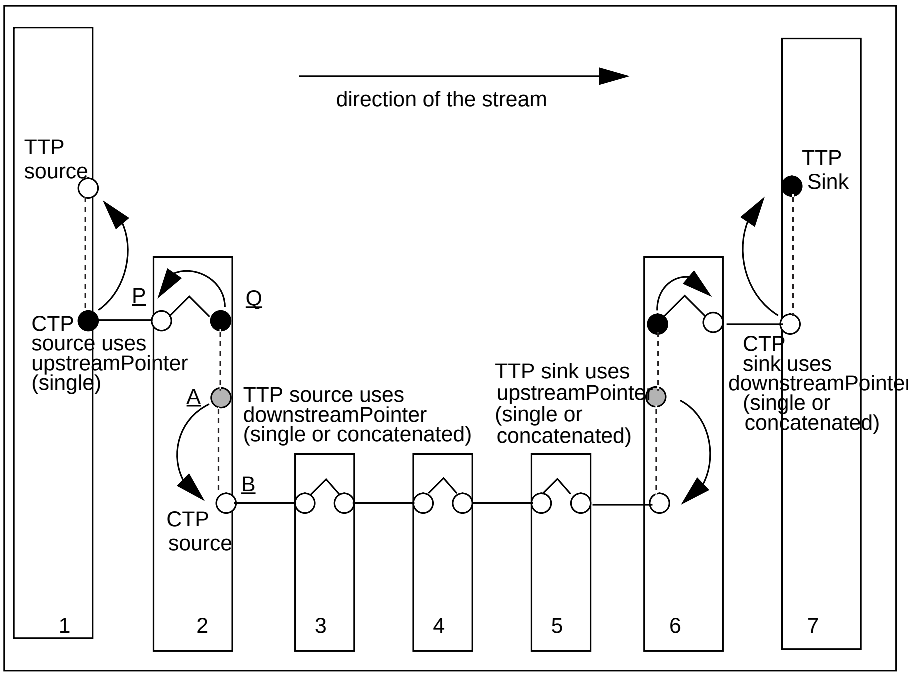
direction of the stream
TTP source
CTP source uses upstreamPointer (single)
P
Q
A
TTP source uses downstreamPointer (single or concatenated)
B
CTP source
TTP sink uses upstreamPointer (single or concatenated)
TTP Sink
CTP sink uses downstreamPointer (single or concatenated)
1
2
3
4
5
6
7

Figure 9.2 Stream connectivity

A “connectivityPointer” has the following ASN1 syntax

```
ConnectivityPointer ::= CHOICE { noneNULL,
                                 singleObjectInstance,
                                 concatSEQUENCE OF
                                 ObjectInstance}
```

where an “ObjectInstance” [9.3] is used to identify a specific object either through a DN or a RDN or an “ad hoc” octet string.

There are several remarks to be made here:

- Pointers are indirections to other objects; this means that it is not possible to read the attributes associated to a sibling TP in a single “GET” request. A first “GET” request is needed to recover the DN of the “pointed to” objects, and a second request must be used to recover the actual information.
- If we consider this model applied to ATM, where from top to bottom we have VCC, VCL, VPC, VPL we realise that these pointers are of no use. What is of interest for an ATM switch is to link between the layer of VCs and the layer of VPs: given a VP (VPC or VPL) one would like to know what are the VCs (VCC or VCL) going through it, and given a VC one would like to know what VP it belongs to. As shown on the figure, the pointers do not cross the layers.
- Having two different objects to designate a *TTPSource* and its associated *CTP-Source* (same for the sink) is pure overhead. Indeed, at a VPC source (A) it makes sense to have additional information compared to the corresponding VPL source (B), but since these two points are very closely related it seems much more natural to consider that a VPC is a specialisation of a VPL: this could be elegantly modelled using two classes: *VPCSource* inheriting from class *VPLSource*.

9.3.2 Naming port, VPs and VCs

Making *TTPSource* and *CTPSource* a single object does not solve the problem we identified earlier of being able to “point” between the VPs layer and the VCs layer (where a VCC is switched from one VPC to another one: point P to Q). In addition it seems natural also to keep the information associated to the sibling point: P pointing to Q and Q pointing to P. At first sight, it seems that in this case we need two pointers.

- First, instead of using pointers, we will simply use the integer values used in the ATM protocol and called VPI and VCI.
- Second, instead of using two attributes to convey this information in the “inheritance” tree, we will much more naturally use the containment tree, and consider that a *VCLSource* is “contained” within the subtree whose root is the corresponding *VPCSource*. A VPCSource is identified through its VPI and any specific VCLSource can be identified within this subtree through its VCI.

One advantage in using CMIP with respect to SNMP is that it is possible to have entries in tables that are tables themselves. Neither ETSI [9.6] nor ATMF M4 [9.4] MIB take advantage of this feature to model the ports, VPs and VCs, using the quite natural structuring facility offered by the containment schema shown in Figure 9.3.

In these two “standard” MIBs, these entities are all contained in the *atmManagedElement* instance, and therefore they have to refer to their real container entity with a pointer or an Id. (For example, A VC has to point to the VP which is containing it.).

In addition, the fact that all VP and VC instances are contained in the same *atm-ManagedElement* instance means that a global numbering scheme has to be used for the VCs and the VPs which does not match the reality.

The alternative solution that we adopted take advantage of the “structuring” facility of the containment tree, as shown in Figure 9.4.

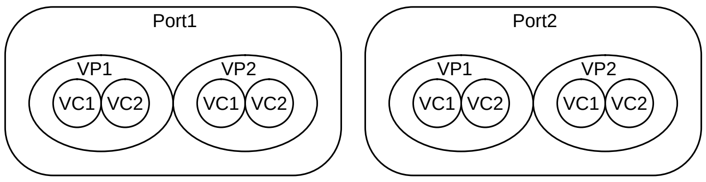

Figure 9.3

Understanding that in a port one can see the endpoint of either a VPC or a VPL, and in a VPC one can see the endpoint of either a VCC or a VCL.

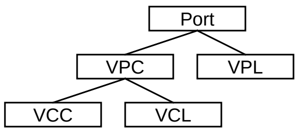

Figure 9.4

9.3.3 Management of the software

An ATM switch not only consists of the switch fabric itself but also of cell buffers, cross connections, and ports. The switch contains also a number of software layers responsible for signalling for connection establishment and release.

When an end terminal (A) wants to establish a connection with an other end terminal (B) it will first contact the call control mechanism on the nearest switch, and forward its request in terms of called number and quality of service, through the signalling protocol. The call control function will ask the routing algorithm about the ports and the VPCs to use to establish the connection, and the CAC algorithm about the availability of those resources. If there is capacity for the new call, the request is transferred to the next node, and the resources are reserved on the switch, otherwise it is rejected. Once B is reached, the connection is established, and the reservation of the resources are made effective.

All these events can be monitored, and provide information on the status of the network.

9.3.3.1 Call Control management

The call control function is mainly a general interface to the control software for establishing, releasing a connection. As such it may provide information on the following topics:

- on a call basis:
  - call acceptance
  - call rejection
- on a connection basis:

- connection acceptance
- connection release
- connection rejection

The information available on these may be:

- calling party number
- called party number
- quality of service required
- VPC and link used
- time of event

Using this basic information, one can use it for a certain number of purposes:

- Compute statistics (average and peak) on interconnection time and holding time of connections to use the appropriate traffic model (bursty or constant) for each end station (source or destination) and for each service type.
- Compute the number of rejected connection requests and established connections, for a certain time period. This could give information on the availability (emptiness and saturation) of certain resources: virtual paths, links between two nodes and providing a certain quality of service. Using this information may lead to the modification of the network.

Typical actions resulting from these measures may result in the modifications (removal, adding, change of allocated bandwidth) of the virtual paths provided by the network.

9.3.3.2 CAC management

The CAC mechanism manages a small database of links, VPCs, and bandwidths which are available or used. The table contains the following fields:

- Link number
- VPC number
- allocated bandwidth
- mean used bandwidth
- peak used bandwidth

The CAC algorithm uses this table to determine whether new connections may be accepted on the VPC(s) it is responsible for. The calculations and measurements are done by the CAC itself. The bandwidth is allocated to a VPC at creation time, and it may be modified with appropriate care by the management system: the allocated bandwidth of a VPC can not be less than the used bandwidth, and the sum of the allocated bandwidths on a link can not exceed the overall bandwidth available on that link.

Monitoring this table offers information on the usage of the links. This table may be modified dynamically in order to allocate more bandwidth to a VPC or add new VPCs

9.3.3.3 Routing management

The routing algorithm uses the route selection table to find a path when an incoming call requests it. This table contains the following entities:

- route identifier
- outgoing link
- outgoing VPC

- route priority

The route identifier must match the first few digits of a called party number in case the numbering scheme is hierarchical, as in common telephone numbering, where the first few digits give a geographical clue on the position of the called party.

The algorithm selects the route according to its priority. Modifying the priorities may lead to redirect new connections to a different path.

In order to manage the routing of the connections (VCCs) through the network, one must be able to modify this table, by adding new routes and deleting old ones.

The GDMO description we have used for this part is as follows:

```
routeSelTable                  MANAGED OBJECT CLASS
        DERIVED FROM           top;
        CHARACTERIZED BY       routeSelTablePkg;
       REGISTERED AS         { icmEtbQafManagedObjectClass 24 };

routeSelTable-node                      NAME BINDING
        SUBORDINATE OBJECT CLASS        routeSelTable;
        NAMED BY SUPERIOR OBJECT CLASS  node AND SUBCLASSES;
        WITH ATTRIBUTE                  routeSelTableId;
        CREATE                          ;
       REGISTERED AS                  { icmEtbQafNameBinding 24
};

routeSelTablePkg               PACKAGE
        ATTRIBUTES             routeSelTableId        GET;
        REGISTERED AS          { icmEtbQafPackage 22 };

routeSelTableId  ATTRIBUTE
        WITH ATTRIBUTE SYNTAX  ICM.Integer;
        MATCHES FOR            EQUALITY;
        REGISTERED AS          { icmEtbQafAttribute 64 };

routeSelEntry                  MANAGED OBJECT CLASS
        DERIVED FROM           top;
        CHARACTERIZED BY       routeSelEntryPkg;
       REGISTERED AS         { icmEtbQafManagedObjectClass 25 };

routeSelEntry-routeSelTable             NAME BINDING
        SUBORDINATE OBJECT CLASS        routeSelEntry;
        NAMED BY SUPERIOR OBJECT CLASS  routeSelTable;
        WITH ATTRIBUTE                  routeSelEntryId;
        CREATE                          WITH-AUTOMATIC-INSTANCE-
NAMING;
        REGISTERED AS { icmEtbQafNameBinding 25 };

routeSelEntryPkg               PACKAGE
        ATTRIBUTES             routeSelEntryId          GET,
                             nodeId                 GET-REPLACE,
                             cos                    GET-REPLACE,
                             portOut                GET-REPLACE,
                             vpiOut                 GET-REPLACE,
                             priority               GET-REPLACE;
        REGISTERED AS          { icmEtbQafPackage 23 };

routeSelEntryId                ATTRIBUTE
        WITH ATTRIBUTE SYNTAX  ICM.Integer;
        MATCHES FOR            EQUALITY;
        BEHAVIOUR              routeSelEntryIdBhvr;
        REGISTERED AS          { icmEtbQafAttribute 70 };

routeSelEntryIdBhvr            BEHAVIOUR
        DEFINED AS             «The EntryId is the index in the
table and
                               is provided by the upper layer»;

priority                       ATTRIBUTE
        WITH ATTRIBUTE SYNTAX  ICM.Integer;
        MATCHES FOR            EQUALITY;
        BEHAVIOUR              priorityBhvr;
        REGISTERED AS          { icmEtbQafAttribute 72 };

priorityBhvr                   BEHAVIOUR
       DEFINED AS             «The priority can have values 0-9.
If in
                             the selection table, two routes are
found
                             with the same routeId, then the one
with
                               the higher priority is chosen»;
```

9.3.4 Statistical information with a very low granularity

As mentioned before, for the connection management, a very large number of statistics may have to be computed. These computations may be done through metric objects [9.2]. However, the key problem is that metric objects work by polling the target values to get samples to do their computations. Since they do not have a very good granularity because they may poll only once every second, many connection events may be lost. Therefore these events must be logged, for example in EventLogRecord [9.2] objects.

This would mean that if the metric object has to compute the average number of connections over the last N seconds, it would have to scan all the log records which time is more recent than the current time-N. Unfortunately metric objects can not compute statistics on an undefined number of instances. Moreover, the instance that is monitored by a metric object has to exist *before* the metric object instance is created. This causes an additional problem if one wants to monitor a connection type that has not been initiated yet.

To circumvent these problems, we have implemented local statistical objects, designed for the specific purpose of computing the following items for each source/destination/quality-of-service over a certain time period:

- connection holding time (mean and average) defined as the duration of a connection
- interconnection time (mean and average) defined as the time between the end of a connection and the beginning of the next connection
- number of accepted connections (mean and average)
- number of rejected connections (mean and average)
- connection rejection rate or ratio representing the number of rejected connections compared to the accepted ones

The proposed approach came up with the containment schema as shown in Figure 9.5.

Where the *StatsTable* contains all the possible Classes of Service (CoS) which is an index on the Qualities of Service supported by the switch. These Instances are created at start-up once for all.

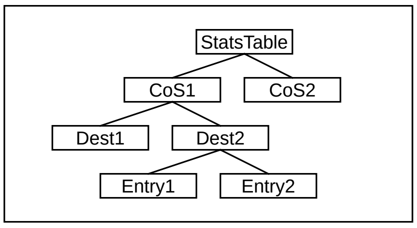

Figure 9.5

The destination instances represent the called destination. These instances are created when an upper layer wants to monitor certain type of connections, for certain destinations. The destinations are extracted from the called party number provided by the call initiator, and represent the node (switch) to which the called party is connected.

An entry is created every time an upper layer wants to monitor a certain connection type. The entry will compute the statistical information described above, based upon the local logs of the agent. This way, all the events will be taken into account, and there is no risk to miss one due to too rough granularity.

The time-period over which the averages are computed may be set using different approaches:

- One could set an attribute T in the entry class, signifying that the computations should be made over the period [t_0 - T to T], t_0 being the time at which a get request is answered.
- One could say the above period T stretches from t_0 to the time of the previous get request.
- The last possibility is to set the period T at creation time, and compute the values every T seconds, independently from the get requests. This solution has the disadvantage of requiring a synchronisation between the manager and the agent, in such a way that the agent polls the result with the same period T.

The full GDMO description is not presented in this document. Instead, the most interesting parts of the inheritance and containment trees are shown in Figure 9.6 and Figure 9.7.

9.3.5 Relationship with ITU Recommendation M.3100

M.3100 was designed for a general purpose and uses a generic approach: the goal being to capture the requirements which can be applicable independently from any specific technology. However the work has been based on the reference architecture for SDH as defined in [9.5].

M.3100 introduces a layered approach based on the abstract concept of trails and connections and their associated end points. The specific class ‘terminationPoint” is introduced. This class is further decomposed into 4subclasses called:

- connectionTerminationPointSource (CTPSource),
- connectionTerminationPointSink (CTPSink),
- trailTerminationPointSource (TTPSource),
- trailTerminationPointSink (TTPSink).

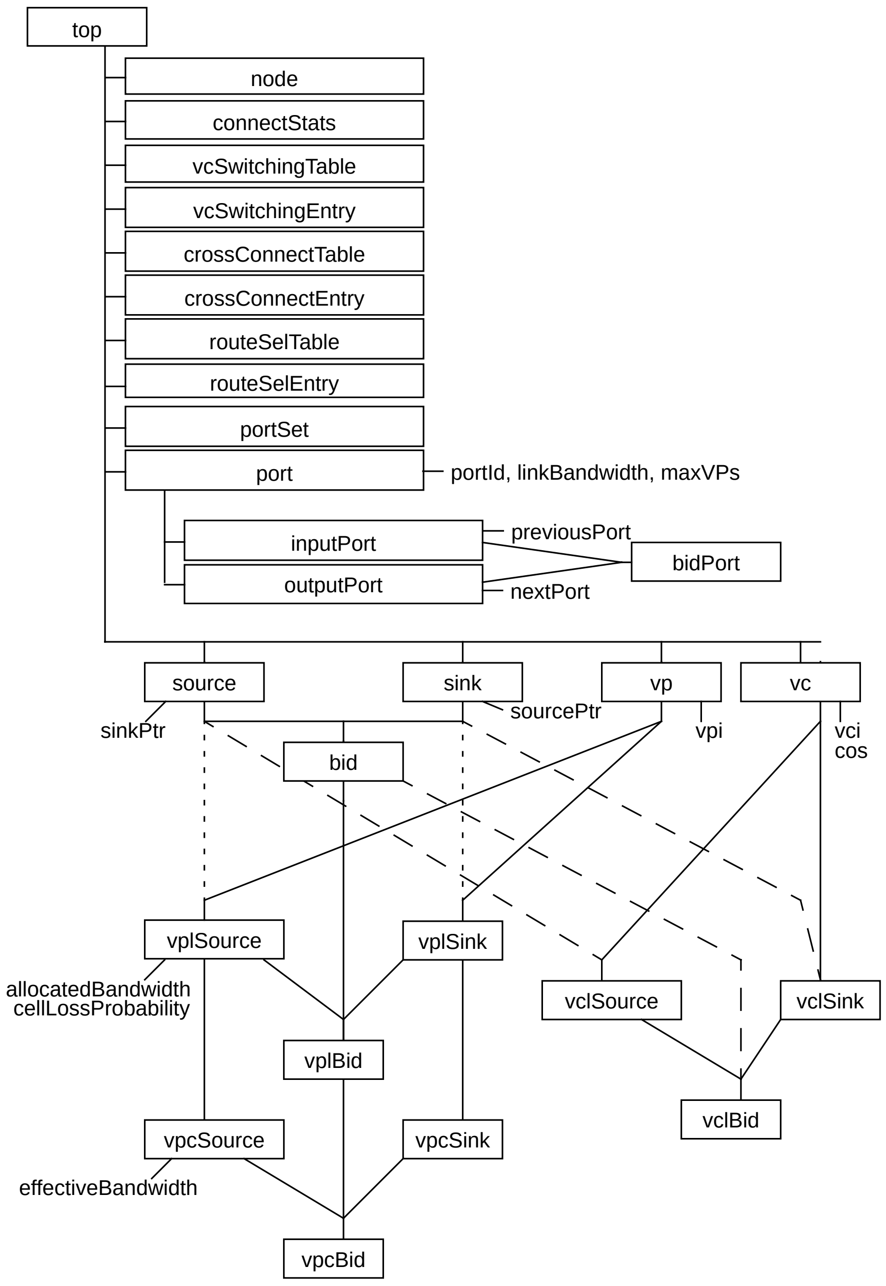

Figure 9.6 NE Level inheritance hierarchy

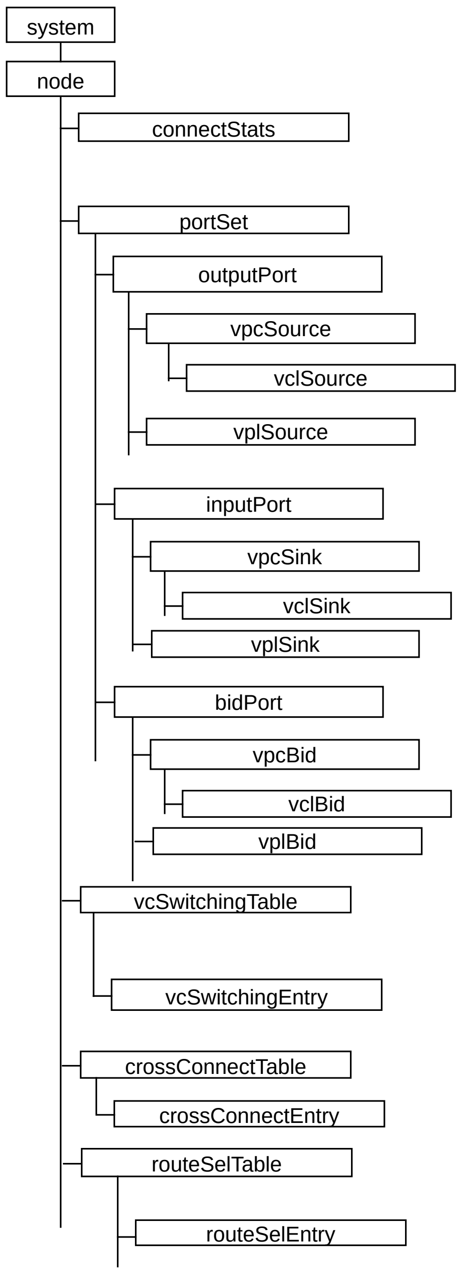

Figure 9.7 NE Level containment schema

A trail is made up of a set of connections. Each trail can act as a “server” for a connection at the next layer up. This layered approach fits quite well with the hierarchical layered network model for ATM, as described in I311. This mapping is illustrated in Figure 9.8.

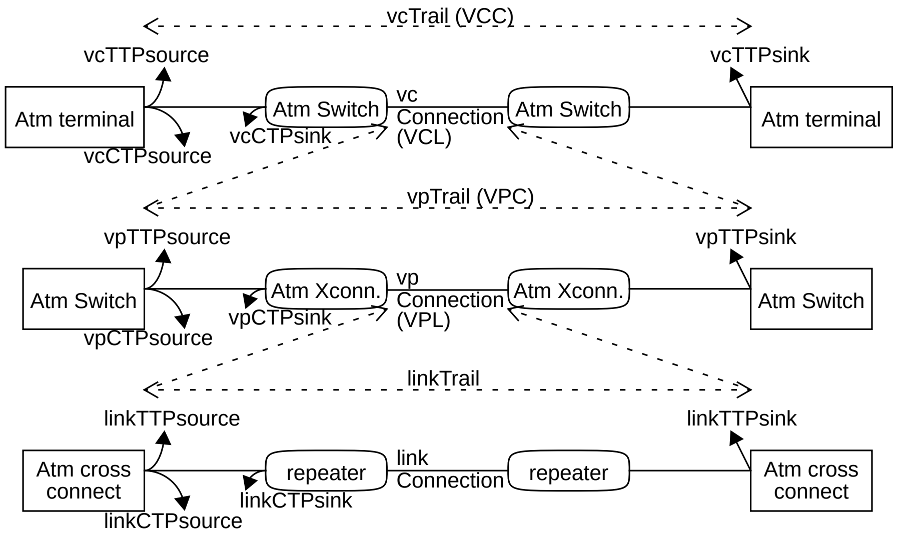

Figure 9.8

Following M.3100, a “Connection” represents the abstraction of a transport connection between two physical equipments within a given layer (n), while a “Trail” represents the abstraction of an end to end transport connection between equipment of the layer above (n+1). A trail is then made of one, or possibly more, connections between layer n equipment and crossconnections within this equipment.

Applying this model to ATM should read as follows: a VCC (Trail) extends through several VCLs (Connections), each VCL corresponding to a VPC (Trail) which in turn extends through a sequence of VPLs (Connections).

The M.3100 classes do not have any feature specific to VPC and VC. In order to model such objects, it is necessary to introduce specific classes:

- A VPC class containing few attributes of general purpose, such as the identifier (VPI) and the operational state. The bandwidth is handled differently along the VP: the maximum allocated bandwidth and the cell loss probability are of interest only at the source end, the effective bandwidth, representing the actual traffic through a given VP, is calculated by the CAC (Connection Acceptance Control) component only at the source of the VPC. Then, these attributes will not be present within the “abstract” class VPC but within specific subclasses.
- A VC class containing attributes to characterise the class of service (CoS) associated to the traffic they generate.

In addition to these “terminationPoint” M.3100 also introduces the “connectivity” class which is further specialised into “trail” and “connection” connectivity classes.

9.4 References

[9.1] ITU-T Recommendation M.3100 “Generic network information model,” September 1992.
[9.2] ITU-T X.721, Information Technology - Structure of Management Information: Definition of Management Information, February 1992.
[9.3] ITU-T X.711, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2, July 1991.
[9.4] M4 Interface Requirements and Logical MIB: ATM Network Element View, Version 1.0 (Draft), ATM Forum, 1994.
[9.5] ITU-T Recommendation G.803, March 1993.
[9.6] ETSI ETSI/NA5 WP BMA, Draft ETS, November 1993.