INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS

DRAFT TECHNOLOGY SUBMISSION WORKING DOCUMENT

IEEE/WG P1520

PROGRAMMING INTERFACES FOR NETWORKS

IEEE/WG P1520

P1520/TS/ATM-017

New York, 28-29 March, 1999

Source: Xbind, Inc.

Title: ATM Switch Resource Abstractions

Authors: Constantin M. Adam, Aurel A. Lazar, Mahesan Nandikesan

Contact: Mahesan Nandikesan, Tel:+1-212-809-3302, Fax:+1-212-809-3305, e-mail: mahesan@xbind.com

TABLE OF CONTENTS

1. Overview

This document provides an IDL specification of the minimal capability set discussed in [1].

1.1 Scope

The level one interfaces are a collection of methods that allow to retrieve the state, to control and to manage the ATM switch resources they abstract.

1.2 Purpose

This is a specification of the level one interfaces mentioned in [1].

2. Resource Abstraction for ATM Switches

This section describes three IDL interfaces: the `ForwardingMap`, the `Multiplexer`, and the `SchedulableRegion`. They are all scoped under the module `P1520` (we have omitted the 'P1520::' prefix, however). Appendix A contains the definitions of the quality of service concepts used in these interfaces. Appendices B and C define the scheduling and buffer management policies and their parameters. Appendix D gives a representation for the schedulable region. Appendix E describes the schedulable region estimation policies. Finally, Appendix F contains a list of exceptions that can arise at this level.

2.1 Forwarding Map

2.1.1 IDL Specification

```
module P1520
{
  const ushort DONT_CARE  = 0;

  struct MapEntry
  {
    ushort in_port;
    ushort in_vpi;
    ushort in_vci;
    ushort out_port;
    ushort out_vpi;
    ushort out_vci;
    ulong buffer_index;
  };
  typedef sequence<MapEntry> MapEntrySeq;

  interface ATMForwardingMap
  {
    ushort getForwardingMapInfo(
      out ushort num_ports,
      out ulong capacity,
      out string mac_address)
    raises(Unknown);

    void getNamespace(
      in  ushort port,
      out ushort min_vpi,
      out ushort max_vpi,
      out ushort min_vci,
      out ushort max_vci)
    raises(InvalidArgument, NotFound, Unknown);

    ushort addChannel(
      in    ushort in_port,
      inout ushort in_vpi,
      inout ushort in_vci,
      in    ushort out_port,
      inout ushort out_vpi,
      inout ushort out_vci,
      in    ulong buffer_index)
    raises (InvalidArgument, UnsupportedMulticastOption, Unknown);
```

```
    ushort removeChannel(
      in ushort in_port,
      in ushort in_vpi,
      in ushort in_vci,
      in ushort out_port,
      in ushort out_vpi,
      in ushort out_vci)
    raises (InvalidArgument, NotFound, Unknown);

    ushort removeMulticast(
      in ushort in_port,
      in ushort in_vpi,
      in ushort in_vci)
    raises (InvalidArgument, NotFound, Unknown);

    MapEntrySeq getOutputEntries(
      in ushort in_port,
      in ushort in_vpi,
      in ushort in_vci)
    raises (InvalidArgument, NotFound, Unknown);

    MapEntry getInputEntry(
      in ushort out_port,
      in ushort out_vpi,
      in ushort out_vci)
    raises (InvalidArgument, NotFound, Unknown);

    MapEntrySeq getForwardingMap (
      in ushort port,
      in ushort vpi)
    raises (InvalidArgument, Unknown);
  };
};
```

2.1.2 Description

```
  struct MapEntry
  {
    ushort in_port;
    ushort in_vpi;
    ushort in_vci;
    ushort out_port;
    ushort out_vpi;
    ushort out_vci;
    ulong buffer_index;
  };
  typedef sequence<MapEntry> MapEntrySeq;
```

Represents one entry of the forwarding map. The input port, VPI and VCI are mapped to the output port, VPI and VCI, and each mapping is associated a buffer in the output port multiplexer. A multicast connection is modeled as a set of map entries with the same input Port, VPI and VCI. For a virtual path, the VCI fields should be set to `DONT_CARE`.

```
  ushort getForwardingMapInfo(
    out ushort num_ports,
    out ulong capacity,
    out string mac_address)
  raises(Unknown);
```

Returns information about the map: the number of ports of the switch the map capacity and the physical address.

```
void getNamespace(
  in  ushort port,
  out ushort min_vpi,
  out ushort max_vpi,
  out ushort min_vci,
  out ushort max_vci)
raises(InvalidArgument, NotFound, Unknown);
```

Returns the virtual path range and the virtual channel range for a given port of the switch.

```
ushort addChannel(
  in ushort in_port,
  inout ushort in_vpi,
  inout ushort in_vci,
  in ushort in_port,
  inout ushort in_vpi,
  inout ushort in_vci,
  in ulong buffer_index)
raises (InvalidArgument, UnsupportedMulticastOption, Unknown);
```

Adds a unicast connection or a branch to a multicast tree. Creates a new map entry that associates the input Port, VPI, VCI to the output Port, VPI, VCI and the output port multiplexer buffer. If the input Port, VPI, VCI are already used, then the method will add a branch to a multicast tree. If both the input and the output Port, VPI, VCI are used, the request is rejected.

```
ushort removeChannel(
  in ushort in_port,
  in ushort in_vpi,
  in ushort in_vci,
  in ushort in_port,
  in ushort in_vpi,
  in ushort in_vci)
raises (InvalidArgument, NotFound, Unknown);
```

Removes a unicast channel or a multicast tree branch from the forwarding map.

```
ushort removeMulticast(
  in ushort in_port,
  in ushort in_vpi,
  in ushort in_vci)
raises (InvalidArgument, NotFound, Unknown);
```

Removes from the forwarding map all the branches of the multicast tree rooted in the given Port, VPI, VCI.

```
MapEntrySeq getOutputEntries(
  in ushort in_port,
  in ushort in_vpi,
  in ushort in_vci)
raises (InvalidArgument, NotFound, Unknown);
```

Retrieves the map entries that have the specified input port, VPI and VCI. If the port, VPI and VCI represent the root of a multicast tree, the method will return a sequence of map entries (an entry per branch).

```
MapEntry getInputEntry(
  in ushort out_port,
  in ushort out_vpi,
  in ushort out_vci)
raises (InvalidArgument, NotFound, Unknown);
```

Retrieves the map entry that contains the specified output port, VPI and VCI.

```
MapEntrySeq getForwardingMap (
  in ushort port,
  in ushort vpi)
raises (InvalidArgument, Unknown);
```

2.2 Multiplexer

2.2.1 IDL Specification

```
module P1520
{

  interface Multiplexer
  {

  // Scheduling policies

    const ushort PROPRIETARY_SP      = 0; // Proprietary (to the switch)
    const ushort FIFO_SP             = 1; // FIFO
    const ushort STATIC_PRIORITY_SP  = 2; // Static priority
    const ushort WRR_SP              = 3;       // Weighted round robin
    const ushort MARS_SP             = 4; // MARS (reference [1])

    struct SchedulingPolicy
    {
      ushort   policy;      // given above
      OctetSeq parameters;  // appendix B

    };

    // Buffer management policies

    const ushort PROPRIETARY_BMP    = 0;  // Proprietary
    const ushort SIMPLE_BMP         = 1;  // See appendix C

    // Parameters for buffer management policies
    struct BufferPolicy
    {
      ushort   policy;      // given above
      OctetSeq parameters;  // appendix C
    };

    ulong getCellRate()
    raises(Unknown);

    void setCellRate(
      in ulong speed)
    raises(MaxLimitExceeded, UnsupportedOption, Unknown);

    void setSchedulingPolicy(
      in SchedulingPolicy policy)
    raises (InvalidArgument, PolicySpecific, UnsupportedOption,
            Unknown);

    SchedulingPolicy getSchedulingPolicy()
    raises (Unknown);

    void setNumberOfBuffers(
      in ushort buffers,
      in UshortSeq buffer_sizes)
    raises (ZeroCardinality, MaxLimitExceeded, UnsupportedOption,
            Unknown);
```

```
    ushort getNumberOfBuffers(
      out UshortSeq buffer_sizes)
    raises (Unknown);

    void getTotalBufferSize(
      out ulong memory_size,
      out ulong max_buffers)
    raises (UnsupportedOption, Unknown);

    ulong addBuffer (
      in ushort buffer_size,
      in ulong bandwidth)
    raises (MaxLimitExceeded, UnsupportedOption, Unknown);

    void removeBuffer (
      in ulong buffer_id)
    raises (NotFound, UnsupportedOption, Unknown);

    void setBufferMgmtPolicy(
      in BufferPolicy policy)
    raises (InvalidArgument, PolicySpecific, UnsupportedOption,
            Unknown);

    BufferPolicy getBufferMgmtPolicy()
    raises (Unknown);

    ushort getNumberOfCellsInBuffer (
      in ulong buffer_id)
    raises (InvalidArgument, UnsupportedOption, Unknown);

    ulong getBufferStatistics (
      in short reset,
      in ulong buffer_id)
    raises (InvalidArgument, UnsupportedOption, Unknown);
  };
};
```

2.2.2 Description

```
  ulong getCellRate()
  raises(Unknown);
```

Returns the cell rate of the multiplexer.

```
  void setCellRate (
    in ulong speed)
  raises(MaxLimitExceeded, UnsupportedOption, Unknown);
```

Sets the cell rate of the multiplexer.

```
  void setSchedulingPolicy(
    in SchedulingPolicy policy)
  raises (InvalidArgument, PolicySpecific, UnsupportedOption, Unknown);
```

Sets the scheduling policy specified by the argument `policy`. The scheduling policies are described in appendix B.

```
  SchedulingPolicy getSchedulingPolicy()
```

```
  raises (Unknown);
```

Returns the scheduling policy setup on the multiplexer.

```
  void setNumberOfBuffers(
    in ushort buffers)
  raises (ZeroCardinality, MaxLimitExceeded, UnsupportedOption,
          Unknown);
```

Sets the number of buffers on the multiplexer and the size of each buffer.

```
  ushort getNumberOfBuffers(out UshortSeq buffer_sizes)
  raises (Unknown);
```

Returns the number of buffers on the multiplexer and the size of each buffer.

```
  ulong addBuffer (
    in ushort buffer_size,
    in ulong bandwidth)
  raises (MaxLimitExceeded, UnsupportedOption, Unknown);
```

Creates a new buffer, specifying the buffer size and the bandwidth associated with it. Returns a buffer id.

```
  void removeBuffer (
    in ulong buffer_id)
  raises (NotFound, UnsupportedOption, Unknown);
```

Removes the buffer identified by `buffer_id`.

```
  void getTotalBufferSize(
    out ulong memory_size,
    out ulong max_buffers)
  raises (UnsupportedOption, Unknown);
```

Returns the amount of memory available to allocate for buffers and the maximum number of output buffers that can be created on the multiplexer.

```
void setBufferMgmtPolicy(
    in BufferPolicy policy)
  raises (InvalidArgument, PolicySpecific, UnsupportedOption, Unknown);
```

Sets the buffer management policy specified by the argument `policy`. The buffer management policies are described in appendix C.

```
  BufferPolicy getBufferMgmtPolicy()
  raises (Unknown);
```

Returns the buffer management policy setup on the multiplexer.

```
  ushort getNumberOfCellsInBuffer (
    in ulong buffer_id)
  raises (InvalidArgument, UnsupportedOption, Unknown);
```

Returns the number of cells currently present in a buffer queue, denoted by `buffer_id`.

```
  ulong getBufferStatistics (
    in short reset,
    in ulong buffer_id)
```

```
  raises (InvalidArgument, UnsupportedOption, Unknown);
```

Returns the number of cells that were sent through the buffer queue denoted by `buffer_id`. If the reset flag is true, then, after returning the present measurement, the measurement statistics are reset to zero.

2.3 Schedulable Region

2.3.1 IDL Specification

```
module P1520
{
  interface SchedulableRegion
  {
    typedef sequence<long>          Hyperplane;
    typedef sequence<Hyperplane>    HyperplaneSeq;

    // Schedulable region estimators
    const ushort PROPRIETARY_SRE = 0;
    const ushort PEAKRATE_SRE    = 1;

    struct EstimatorDescription
    {
      ushort type;         // Estimator type (PROPRIETARY_SRE, etc)
      OctetSeq parameters; // Estimator-dependent parameters.
    };

    struct ClassToBuffer{
      ushort traffic_class;
      ulong buffer_id;
    };
    typedef sequence<ClassToBuffer> ClassToBufferSeq;

    void setTrafficClasses(
      in TrafficClassSeq traffic_class_seq)
    raises (InvalidArgument, InvalidSize, UnsupportedOption, Unknown);

    TrafficClassSeq getTrafficClasses()
    raises (Unknown);

    void setClassToBufferMap (in ClassToBufferSeq cls_to_buffer)
    raises (Unknown);

    void getClassToBufferMap (out ClassToBufferSeq cls_to_buffer)
    raises (Unknown);

    void setEstimator(
      in EstimatorDescription estimator)
    raises (InvalidArgument, InvalidSize, UnsupportedOption,
            PolicySpecific, Unknown);

    EstimatorDescription getEstimator()
    raises (Unknown);

    HyperplaneSeq getSchedulableRegion()
    raises (UnsupportedOption, Unknown);

  };
};
```

2.3.2 Description

The data type `HyperplaneSeq` describes a schedulable region as a sequence of hyper planes as described in the appendix D.

The estimator policies are described in appendix E.

```
void setTrafficClasses(
  in P1520::TrafficClassSeq traffic_class_seq)
raises (InvalidArgument, InvalidSize, UnsupportedOption, Unknown);
```

Sets the specification for traffic classes according to the sequence of traffic class descriptors provided as input. This traffic class descriptor structure has been specified in the previous chapter.

```
P1520::TrafficClassSeq getTrafficClasses()
raises (Unknown);
```

Returns the specification of the traffic classes defined on the switch.

```
void setClassToBufferMap (
  in ClassToBufferSeq cls_to_buffer)
raises (Unknown);
```

Sets the mapping of the traffic classes to output multiplexer buffers.

```
void getClassToBufferMap (
  out ClassToBufferSeq cls_to_buffer)
raises (Unknown);
```

Returns the mapping of the traffic classes to output multiplexer buffers.

```
void setEstimator(
  in EstimatorDescription estimator)
raises (InvalidArgument, InvalidSize, UnsupportedOption,
        PolicySpecific, Unknown);
```

Sets the schedulable region estimation algorithm specified by the argument `estimator`.

```
EstimatorDescription getEstimator()
raises (Unknown);
```

Returns the description of the algorithm that estimates the schedulable region.

```
HyperplaneSeq getSchedulableRegion()
raises (UnsupportedOption, Unknown);
```

Returns the schedulable region as a sequence of hyper planes.

References

[1] C. M. Adam, A. A. Lazar, M. Nandikesan, "Model for designing open interfaces for networks," IEEE Working Group P1520, March 1999.

[2] J. M. Hyman, A. A. Lazar, G. Pacifici, "Real-time scheduling with quality of service constraints," IEEE Journal on Selected Areas of Communications, September 1991.

[3] C.M. Adam, J.-F. Huard, A.A. Lazar, K.-S. Lim, M. Nandikesan, E. Shim, “Proposal for standardization of ATM Binding Interface Base 2.1”, IEEE Working Group P1520, January 18, 1999.

Appendices

A. Traffic Classes and Quality of Service

A traffic class is a statistical model for an information stream. It is described by a set of quantitative and qualitative parameters. The former consists of parameters such as the peak rate, average rate and maximum burst size of the stream. The latter consists of a descriptor such as video, voice, audio or data to describe the traffic characteristics qualitatively. A special descriptor called premium is used to describe traffic that does not fit into any of the other categories.

With each traffic class, a set of quality of service constraints is associated.

Traffic Description – Quantitative

Peak rate Reciprocal of the shortest intercell duration.

Average rate Average bit-rate

Maximum burst size

Traffic Description – Qualitative

Premium Arbitrary statistics

Video Real-time video statistics

Voice Real-time voice statistics

Audio Real-time audio statistics

Data Data statistics

Quality of Service

Maximum Cell Delay Maximum queueing delay of a cell at a multiplexer

Average Cell Delay Average queueing delay of a cell at a multiplexer over a measurement era

Cell Loss Ratio Fraction of cells lost at a multiplexer over a measurement era.

Average Gap Loss Average number of consecutively lost cells at a multiplexer over a measurement era.

Measurement era The duration over which averages are computed

IDL Specification

```
module P1520
{
  const ulong PREMIUM_NTD      = 0;
  const ulong VIDEO_NTD        = 1;
  const ulong VOICE_NTD        = 2;
```

```
  const ulong AUDIO_NTD        = 3;
  const ulong DATA_NTD         = 4;

  struct TrafficDescription
  {
    ulong  peak_rate;       // Peak rate (kb/s)
    ulong  ave_rate;        // Average rate (kb/s)
    ulong  max_burst_size;
    ulong  qualitative;     // PREMIUM_NTD, VIDEO_NTD, etc from above
  };

  struct QOSDescription
  {
    ulong  max_delay;      // Maximum cell delay (usec)
    ulong  ave_delay;      // Average cell delay (usec)
    float  loss;           // Cell loss ratio
    float ave_gap_loss;    // Average gap loss
    ulong  meas_era;       // Measurement era (msec)
  };

  struct TrafficClass
  {
    TrafficDescription traffic;
    QOSDescription     qos;
  };

  typedef sequence<TrafficClass> TrafficClassSeq;
};
```

B. Scheduling Policies

The parameters for the scheduling policies are as follows:

Policy	Parameters
PROPRIETARY_SP	-
FIFO_SP	-
STATIC_PRIORITY_SP	- (The priority is given by the order of the buffers in the structure SchedulingPolicyNode.)
WRR_SP	Weights, specified as a UshortSeq, in the order of the buffers in the structure SchedulingPolicyNode.
MARS_SP	H, {h_i; 1<= i <=n}, specified as a UshortSeq, where n is the number of classes. The order of the h_i's is the same as that of the buffers in the structure SchedulingPolicyNode.

C. Buffer Management Policies

Simple Policy - cells are admitted if and only if the queue length of the is below the specified threshold.

The parameters for the buffer management policies are as follows:

Policy	Parameters
PROPRIETARY_BMP	-
SIMPLE_BMP	Buffer thresholds (two bytes each) starting with buffer 1.

D. Schedulable Region Representation

The schedulable region of a multiplexer is the set of possible combinations of calls of each traffic class that the multiplexer can handle while supporting quality of service [2].

An *n*-dimensional schedulable region *S* will be represented using an *m* x *n* non-negative matrix $A = (a_{ij})$, where m is a positive integer:

$$S = \cup_{i=1}^{m} H_i$$

where

$$H_i = \{x \in N_n \mid \sum_{j=1}^{n} x_j/a_{ij} \leq 1\}$$

The convention 0/0 = 0 and $x/0$ = infinity for $x > 0$ shall be adopted.

The rows of the matrix *A* represent *m* (*n*-1)-dimensional hyperplanes; H_i is the set bounded by the *i*-th hyperplane and the coordinate hyperplanes.

E. Schedulable Region Estimation Policies

The PEAKRATE_SRE schedulable region estimator estimates the schedulable region by the hyper plane described by the following sequence: $(C/p_1, C/p_2, \ldots, C/p_n)$ where C is the line rate and p_i is the peak rate of class i.

The PROPRIETARY_SRE and PEAKRATE_SRE schedulable region estimators take no parameters.

F. Data Types and Exceptions

```
module P1520
{
  typedef unsigned short ushort;
  typedef unsigned long  ulong;
  typedef sequence<octet> OctetSeq;
  typedef sequence<ushort> UshortSeq;

  exception InvalidArgument{};
  exception InvalidSize{};
  exception ZeroCardinality{};
  exception MaxLimitExceeded{};
  exception UnsupportedOption{};
  exception IncompatibleValueSet{};
  exception NotFound{};
  exception PolicySpecific{};
  exception Unknown{};
  exception UnsupportedMulticastOption{};

};
```