The ATM Forum
Technical Committee

# API Semantics for Native ATM Services Using UNI 4.0

AF-SAA-0108.000

February, 1999

## Acknowledgements

This document would not have been possible without the outstanding and continuing efforts of several individuals. These people have played a significant role in the development of this document via contributions, comments, editing, reviewing, knowledge and support. The editor wishes to thank these people:

Werner Almesberger
Francois Audet
Ken Brown
Tom Jepsen
Arata Koike
Eric Lampland
K.K. Ramakrishnan
Serge Sasyan
John Shaffer
Jong-Jin Sung

Editor: Steven A. Wright

Chairman, SAA API Ad Hoc: Jim Harford

# Contents

# TABLE OF FIGURES

# 1. Introduction

## 1.1. Purpose of Document

This document specifies the semantic definition of ATM-specific services that are available to software programs and hardware residing in devices on the user side of the ATM User-Network Interface. The ATM environment provides a wealth of new services to the application developer. This allows enhancements in performance and specification of network characteristics. Such ATM-specific services are denoted by the term "Native ATM Services".

"Semantic", means that this document will describe the services in a way that is independent of any programming language or operating system environment. Semantic specifications for generic services define various aspects of the interface to those underlying services, such as:

- request for the underlying service to perform some action
- notification that some event has occurred
- parameters of the requests and notifications
- response codes to the requests and notifications.

This document uses state machines, entity-relation diagrams, primitives, implementation-specific tips, and other informative text to accomplish this aim.

The semantic description presented in this document is not an Applications Programming Interface (API). An API is a set of libraries or interfaces that enable an application to use the language in which it is written to access the functionality of lower-level modules - such as operating systems, graphical user interfaces, and communications protocols. The ATM Forum's interest is in applications being able to access the functionality provided by Native ATM Services.

This document will advance the development of such APIs that allow access to Native ATM Services. The semantic description included herein is intended to influence the direction of these emerging APIs. This document addresses concerns with both interoperability aspects and the proper abstraction of ATM procedures and parameters. Thus, this semantic description provides a firm engineering foundation and logically precedes any API development. Note that API development is expected to occur within the ATM Forum, other industry groups, and ATM vendors.

Note also that the semantic description herein is not limited to development of traditional APIs. Rather, this interface can be applied to any software program or hardware that uses Native ATM Services. Some examples of this are:

1. operating system kernel interface between Native ATM Services and ATM LAN Emulation
2. operating system kernel interface between Native ATM Services and traditional networking protocols (e.g. IP, X.25)
3. operating system kernel interface to an ATM device driver, where the device driver contains the implementation of the Native ATM Services
4. inside a PBX, used for circuit emulation
5. inside an ATM switch, used for vendor applications providing value-added services.

## 1.2. Scope of Document

"Native ATM Services" include the following:

1. data transfer, including both reliable and unreliable data delivery, using the ATM layer and various ATM adaptation layers
2. provisions for setting up switched virtual circuits (SVC)
3. provisions for setting up permanent virtual circuits (PVC)

4. traffic management considerations, including traffic types and quality of service guarantees
5. distribution of connections and associated data to the correct application, or entity.
6. provisions for local participation in network management (i.e. ILMI and OAM protocols)

The next section presents a reference model showing the relationships between the various software components in a device at the user side of the ATM User-Network Interface. In the reference model, the dashed line labeled "Native ATM SAP" identifies the general scope and interest of this document.

This version of the specification supports version 4.0 of the ATM Forum's User-Network Interface (UNI) Specification. Future versions of this specification will support future versions of the UNI. Not all features of UNI 4.0 are supported by this document. In particular, several limitations are listed below.

- UNI 3.x,4.0 allows for two levels of ATM bearer service: virtual path (VP) and virtual channel (VC). This document supports only VC-level service. The support of VP-level service is for further study.
- UNI 3.x,4.0 allows for AAL type 1, AAL type 3/4, AAL type 5, and a user-defined AAL. This document supports only AAL Type 1, AAL type 5, and a user-defined AAL. The support for AAL1 and user-defined AAL includes the control plane (signaling), but the data plane is considered implementation-specific at this time. The support of the other AAL types (including AAL type 2) is for further study.
- This document supports only the message mode of AAL type 5. Support of AAL type 5 streaming mode is for further study.
- The procedures in this document do not support a single ATM device being multiple leaves on a same point-to-multipoint connection. This feature is for further study.
- Some features of UNI 3.0 were changed or obsoleted by UNI 3.1. Similarly some of the features of UNI 3.1 were obsoleted by UNI 4.0. Where such conflicts exist, this document is consistent with UNI 4.0.

Motivation for a software interface specification centers on portability of software and reduction of development expense. In addition, this document addresses end-to-end interoperability issues, such as:

- distribution of an incoming connection to the correct application inside the device on the user side of the UNI
- agreement of what parameters supported in the UNI may be expected by the application

## 1.3. Reference Model

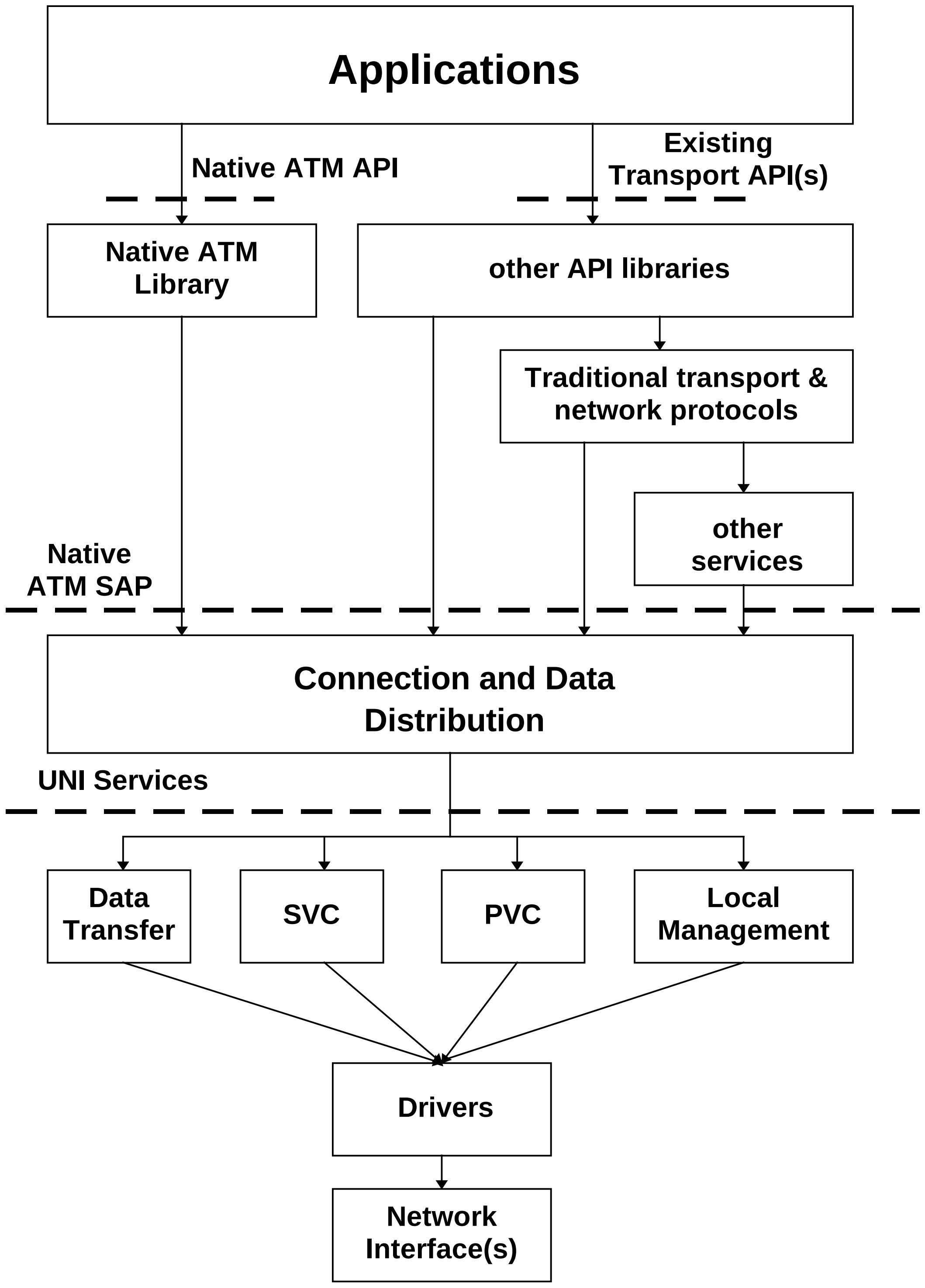

**Figure 1: Reference Model for Native ATM Services (Native ATM SAP)**

### 1.3.1. Notes for the Reference Model

The notes in this section are intended to help the reader understand the reference model.

1. “Native ATM API” — an API that is specifically tailored to support Native ATM Services.

2. “Existing Transport APIs” — an API that provides application access to an underlying transport layer. Note that in order to support applications that are aware of an underlying ATM network, the API must be extended to support Native ATM Services. Examples of existing transport APIs include sockets, XTI, Winsock, Netbios, etc.
3. “Native ATM Library” — software component that presents the Native ATM API to an application.
4. “other API libraries” — software components that present the transport APIs to the application.
5. “traditional transport & network protocols” — traditional data communications protocols such as X.25, TCP/IP, SPX/IPX, SNA, Netbui, Appletalk, etc. Note that these protocols may or may not be aware of the underlying ATM network.
6. “other services” — software components that provide an appearance of some network type other than ATM. Examples include ATM LAN Emulation, Circuit Emulation, etc.
7. “Connection and Data Distribution” — this component allows multiple applications (software programs and hardware) to simultaneously use the functions of ATM networks in a native fashion.
8. The directions of the arrows in the diagram are meant to loosely show the relationship between a service provider and a service user. For example, "Applications" uses the services of "Native ATM API".
9. The data transfer block can be SSCOP/AAL5

## 1.4. Normative Versus Informative Text

The main body of this specification, plus the annexes, are considered normative text. If a particular implementation supports a given service (e.g. SVC, PVC, AAL5, AAL1, SSCOP, ILMI, OAM), then access to that service shall be done in a manner consistent with this specification. The normative text can include hints for implementations.

The appendices of this specification are considered informative text.

# 2. API_CONNECTION State Machines

## 2.1. Motivation

This document describes behavior that is dynamic in nature. The valid actions across the interface to Native ATM Services change with time. For example, an application may not transfer data to another ATM device until an ATM call (between the two ATM devices) is setup across the network. In order to accomplish many actions, a certain sequence of requests and notifications must occur across the interface to Native ATM Services. To be precise, this document describes the dynamic behavior in terms of a finite state machine.

## 2.2. One-Shot State Machine

Most finite state machines contain cycles, or return paths, because the object being modeled repeats a similar sequence of actions. With these kinds of objects, a "dead end" in the state transition path implies a deadlocked state. The object becomes "stuck" in the deadlocked state and is unavailable for future use. For this reason, the last state in an operational sequence often returns to some initial state.

The state machines used for Native ATM Services are "one-shot" state machines. After the last state in the operational sequence, the state machine remains in the final state (e.g. A11) forever.

The difference in these two approaches result from differences in the objects being modeled. The first case is used when modeling an object that can be reused or recycled. Examples of this are communication channels, blood cells carrying oxygen, or a rental video tape. The second case can be used when modeling an object that has a finite *lifespan*. Examples of this are human age progression or consumption by fire.

By design, the state machines used for Native ATM Services models objects with a finite lifespan. The objects being modeled are:

- a connection between two or more parties across an ATM network, which occurs once in time,
- an LIJ request, each of which occurs once from the perspective of the leaf and root.

The alternative would have been to model a *capacity* provided by the ATM device's operating system to communicate across the ATM network. In real operating systems, this capacity is typically identified by something analogous to a UNIX file descriptor. The problem with this approach is that various operating systems and existing APIs attach different semantics to the capacity identifier. The differences might include ability to share the capacity identifier across process boundaries, generation of a new capacity identifier when an incoming call is accepted, and the recycling of a capacity identifier after a connection has been terminated. The differing semantics for capacity identifiers would have resulted in a state machine that was tied to a particular operating environment or existing API.

Native ATM Services abstracts away from these differences. Instead, the API_connection state machine models an object much more fundamental to ATM communication: an individual connection across the ATM network. The focus of the API_connection state machine is an abstract concept herein called an "API_connection" or "connection". Note that the definition for "connection" differs in this document from the use of that term found in the UNI specification. The precise definition and properties of an API_connection were created in order to describe the interface to Native ATM Services in terms of a finite state machine. The specific definition for an API_connection is found immediately below.

In addition to the API_connection state machine described above, other state machines are referenced in this specification in relation to the LIJ services. Theses state machines are the LIJ Leaf State Machine and the LIJ Root State Machine which apply at the respective leaf and root API_Endpoints of an LIJ ATM Multicast connection.

The number of API_connection State Machines, LIJ Leaf State Machines, and/or LIJ Root State Machines controlled by an API_endpoint is dependent on the actions taken by the applications. All potential LIJ and API_connection Finite State Machines (FSMs) are considered initialized when the API_endpoint is allocated through ATM_associate_endpoint. Primitives affect all relevant API_connection and LIJ FSMs controlled by a particular API_endpoint at the time the primitive is invoked. (See section 2.5.2 for details)

## 2.3. Terms Used

- “API_endpoint” or “endpoint” - an object associated with a set of attributes by which an application communicates with other ATM devices.
- “API_connection” or “connection” - a relationship between an API_endpoint and other API_endpoints, that has the following characteristics:
  1. Data communication may occur between the API_endpoint and the other API_endpoints participating in the API_connection.
  2. Each API_connection may occur over a duration of time only once; the same set of communicating API_Endpoints may form a new connection after a prior connection is released.
  3. The API_connection may be presently active (able to transfer data), or merely anticipated for the future.
  4. It is possible that more than one API_endpoint of an API_connection may reside within the same ATM device. In this case, the underlying ATM service provider may or may not detect the fact and perform an internal loopback.

## 2.4. States of API_connection

The states of the API_connection state machine are denoted by the terminology (Ax), where x is an Arabic integer. The states of the LIJ Leaf State Machine are denoted by the terminology (Lx), where x is an Arabic numeral. The states of the LIJ Root State Machine are denoted by the terminology (Rx), where x is an Arabic numeral.

### 2.4.1. Null (A0)

There is no association between the API_connection and the API_endpoint that may be used in the future to offer native ATM service to the application.

### 2.4.2. Initial (A1)

An association now exists between the API_connection and the API_endpoint that may be used to offer native ATM service to the application.

### 2.4.3. Outgoing Call Preparation (A2)

The application has indicated the intention of placing an outgoing call across the ATM network. Any data structures that are needed to specify characteristics of the outgoing call must exist while the API_connection is in state A2. Those data structures are initialized, to the extent possible, when the API_connection enters state A2. The application is allowed to examine and/or modify some subset of these data structures while in state A2.

### 2.4.4. Outgoing Call Requested (A3)

The application has requested that an outgoing call be placed across the ATM network. The characteristics of the call are those specified by data structures that were manipulated while the API_connection was in state A2.
State A3 is entered when the application issues a primitive requesting that the outgoing call be established. This state (A3) can be mapped to the following possible system implementations:

| | |
|---|---|
| **polling:** | The application polls to determine the status of the requested outgoing call. While polling (and before receiving a status indication), the API_connection remains in state A3. |

**blocking:** The operating system blocks the application's process thread, until the outgoing call attempt is either accepted or rejected by the network. While the application is blocked, the API_connection remains in state A3.

**messaging:** The application receives an asynchronous message to indicate either acceptance or rejection of the outgoing call attempt. While the application is waiting for that message, the API_connection remains in state A3.

### 2.4.5. Incoming Call Preparation (A4)

The application has indicated the intention of receiving an incoming call across the ATM network. Any data structures that are needed to queue incoming calls (for the purpose of presenting them to the application) must exist while the API_connection is in states A4, A5, A6, and A7.

### 2.4.6. Wait Incoming Call (A5)

The application has requested that incoming calls from the ATM network be queued for possible acceptance. The characteristics of the queue of potential calls are those specified by data structures that were manipulated while the API_connection was in state A4.

State A5 is entered when the application issues a primitive requesting that incoming calls with the registered SAP be queued and presented to this application. This state (A5) can be mapped to the following possible system implementations:

**polling:** The application polls to determine the presence of an incoming call that has not been presented to the application. While polling, the API_connection remains in state A5 until the application is notified of the new incoming call.

**blocking:** The operating system blocks the application's process thread, until a new incoming call is present. While the application is blocked, the API_connection remains in state A5.

**messaging:** The application receives an asynchronous message to indicate that a new incoming call is present. While the application is waiting for that message, the API_connection remains in state A5.

### 2.4.7. Incoming Call Present (A6)

An incoming call exists that has not been presented to an application; it is presented to the application for possible acceptance. While in this state (A6), the Native ATM Services makes available characteristics of the incoming call that help the application make an accept/reject decision. If parameter negotiation is present, the application can choose and/or modify the appropriate parameters.

### 2.4.8. Incoming Call Requested (A7)

The application has accepted the incoming call that is at the head of the incoming call queue. However, the call must be awarded by the network before the call is completely setup and ready for data transfer.

State A7 is entered when the application issues a primitive requesting that the incoming call at the head of the incoming call queue be accepted. This state (A7) can be mapped to the following possible system implementations:

**polling:** The application polls to determine the status of the accepted call. While polling (and before receiving a status indicating the call was either awarded or released), the API_connection remains in state A7.

**blocking:** The operating system blocks the application's process thread, until the accepted call is either awarded or released. While the application is blocked, the API_connection remains in state A7.

**messaging:** The application receives an asynchronous message to indicate that the call is either awarded or released. While the application is waiting for that message, the API_connection remains in state A7.

### 2.4.9. Point-to-Point Data Transfer (A8)

The point-to-point call (either outgoing or incoming) has been set up across the ATM network. The application may now send and receive data. If any additional states are required for flow control, they will be considered sub-states of state A8.

### 2.4.10.Point-to-Multipoint Root Data Transfer (A9)

The point-to-multipoint call has been set up across the ATM network, with the application being the root node. The application may now send data. If any additional states are required for flow control, they will be considered sub-states of state A9. If any additional states are required for the joining and releasing of additional leafs, they will be considered sub-states of state A9.

### 2.4.11.Point-to-Multipoint Leaf Data Transfer (A10)

The point-to-multipoint call has been set up across the ATM network, with the application being a leaf node. The application may now receive data. If any additional states are required for flow control, they will be considered sub-states of state A10.

### 2.4.12.Connection Terminated (A11)

The connection's lifespan is completed. This state could have been reached by either:

1. An active connection (API_connection in state A8, A9, or A10) was released.
2. A pending connection (API_connection in state A3, A6 or A7) was released by the network, remote ATM device, or the application.
3. A connection was aborted by the application.

There is no longer an association between the API_connection and the API_endpoint that was used to offer native ATM service to the application.

### 2.4.13.Leaf - Initial (L0)

The join request modeled by this state has not been requested by the application.

### 2.4.14.Leaf - LIJ Pending (L1)

The join request modeled by this state has been requested; the leaf application is now awaiting a response.

### 2.4.15.Leaf - LIJ Resolved (L2)

The join request that was previously considered pending (state L1) has been resolved, in one of the following ways:

- the leaf has been added to the point-to-multipoint connection
- the leaf's join request has been denied by either the network or the root of the point-to-multipoint connection
- NAS timed out waiting for a response to the leaf's request.

### 2.4.16.Root - Initial (R0)

The LIJ call identification and LIJ call parameters have not been specified for the API_endpoint.

### 2.4.17.Root - Bound (R1)

The LIJ call identification and LIJ call parameters have been specified for the API_endpoint.

### 2.4.18.Root - LIJ Pending (R2)

The join request modeled by this state has been requested; NAS is awaiting a response from the root application.

### 2.4.19.Root - LIJ Resolved (R3)

The join request that was previously considered pending (state R1) has not been resolved, in one of the following ways:

- the leaf has been added to the point-to-multipoint connection
- the leaf's join request has been denied by the root application

## 2.5. Entity-Relationship Diagrams

The following diagrams are entity-relationship diagrams for each state of API_connection. The prime purpose of the entity-relationship diagrams is to show, for each state:

- the data structures required to implement Native ATM Services
- the information available to the application, through the interface primitives.

In the diagrams, the following conventions are used:

- the rectangles are entities
- the circles are attributes of the attached entities
- the diamonds are relationships between the entities
- the numbers on either side of the diamond denote the cardinality of the relationship (e.g. one-to-one, many-to-one).

### 2.5.1. Client-Server modeling

Note the cardinality of the relationship between API_endpoint and API_connection. During the early states of the state transition path, the relationship is one-to-many. This means that many API_connections are "bundled" together with one API_endpoint. During the later states of the state transition path, the relationship is one-to-one. Thus, only one API_connection is associated with an API_endpoint.

This is done to model the manner in which client-server programs are written. Typically, a server program will listen for incoming calls on a single API_endpoint. When an incoming call is accepted, then the server program will create a child process that communicates with the client program, but communication will involve a new API_endpoint. Thus, an API_endpoint is created while the API_connection transitions into a data transfer state.

To handle this, the model allows API_connection to change the API_endpoint with which it is associated, during the life of API_connection. The future API_endpoint with which API_connection will be associated during the data transfer phase may be unknown during the early states of the state transition path.

As an example, consider that API_connections labeled X, Y, and Z are going to occur some time in the future, involving a server program listening for incoming calls on API_endpoint 22. The state for API_connections X, Y, and Z would be state A5; and all of them would be associated with API_endpoint 22. As incoming calls are accepted, the following state transitions occur:

1. API_connection X moves to state A8 and becomes associated with API_endpoint 23. Connections Y and Z remain in state A5 and are associated with API_endpoint 22.
2. API_connection Y moves to state A8 and becomes associated with API_endpoint 24. Connection Z remains in state A5 and is associated with API_endpoint 22.
3. API_connection Z moves to state A8 and becomes associated with API_endpoint 25. Any other connections that will be forked from the server program remain in state A5 are associated with API_endpoint 22.

### 2.5.2. Specific ER Diagrams for Each State

#### 2.5.2.1. Null (A0)

In this state, there is no association between the API_connection and the API_endpoint that may be used in the future to offer native ATM service to the application. Also, there is no relationship between the application and the API_endpoint. Thus, there is no entity-relationship diagram for this state.

#### 2.5.2.2. Initial (A1)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many as shown in Figure 1. An application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

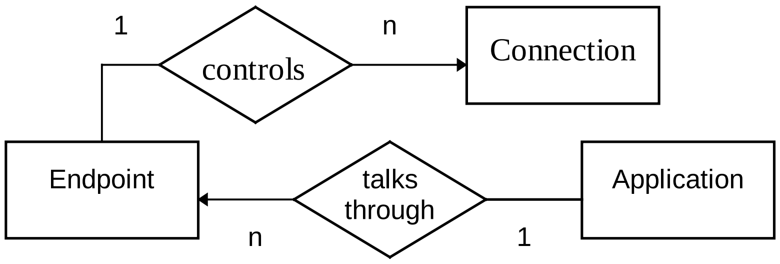

**Figure 1 E-R Diagram for A1 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

#### 2.5.2.3. Outgoing Call Preparation (A2)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

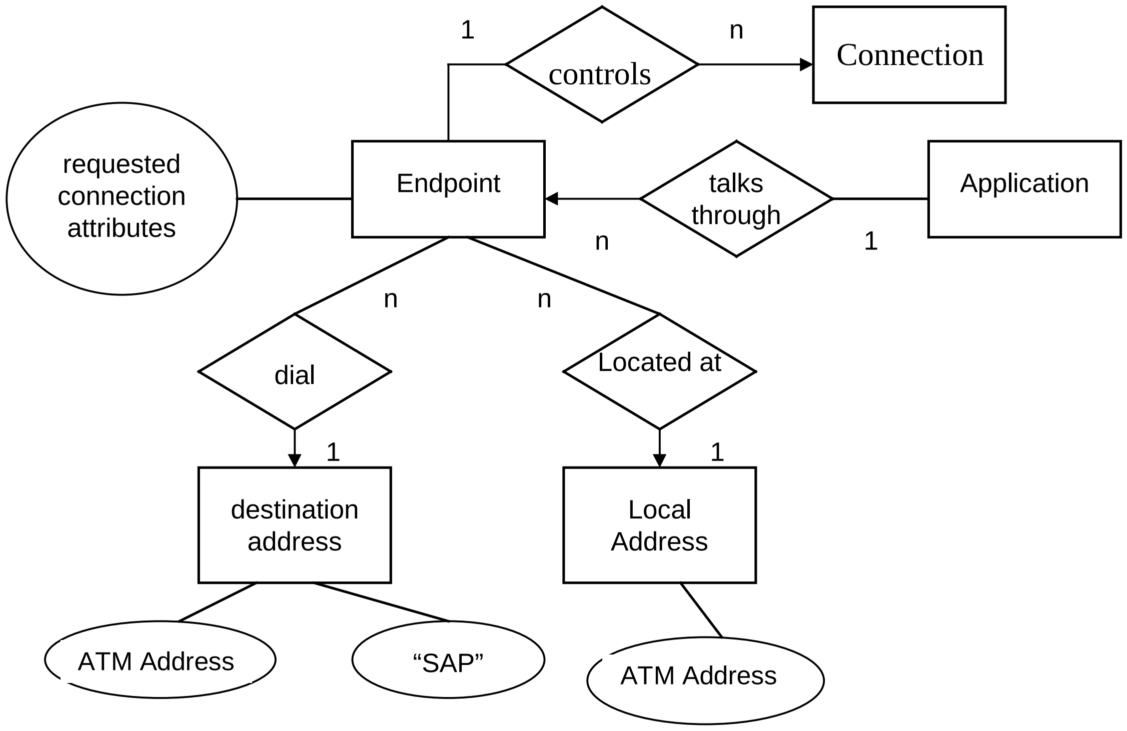

**Figure 2 E-R Diagram in A2 State**

There exist "requested connection attributes" that specify various characteristics of the call to be originated across the ATM network. These are modeled as an attribute of API_endpoint. The application issues primitives to query or set the attributes. Examples of these attributes would include AAL type, forward peak cell rate, quality of service class, and AAL1 clock frequency recovery method.

The API_endpoint is associated with a destination address (as shown in Figure 2), which is the complete specification for the software entity with which the application desires to communicate. The destination address includes both the ATM address of the target device, plus a "SAP" that allows the call notification to reach the correct software entity within that target device. See section 4.6 for more information on call distribution within an ATM device. Notice the many-to-one relationship implies that many API_endpoints can simultaneously be communicating with the same destination address.

NOTE: The destination address is supplied by the primitive that moves the connection from this state to A3.

In cases where the API_endpoint resides in a device known by multiple ATM addresses, there can only be one local ATM address associated with each API_endpoint. The ATM address referenced here will be reported in the "Calling Party Number" signaling information element. This ATM address might be known implicitly by default, or it could be explicitly specified by the application. Notice the many-to-one relationship implies that many API_endpoints can simultaneously be calling from the same ATM address.

### 2.5.2.4. Outgoing Call Requested (A3)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.
In this state, the relationship between an API_endpoint and an API_connection is one-to-one (as shown in Figure 3). Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

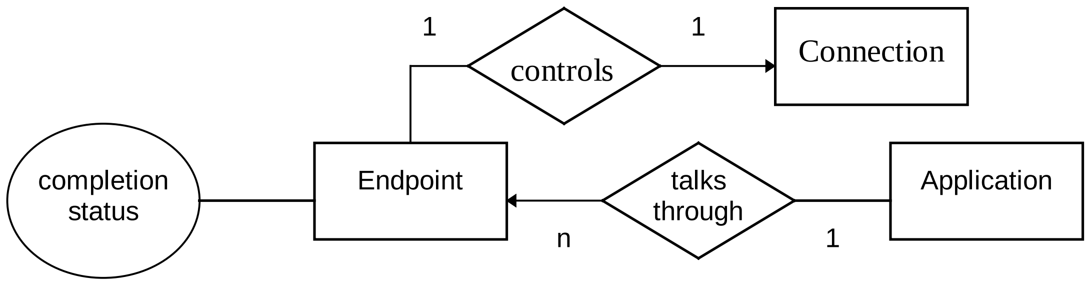

**Figure 3 E-R Diagram for A3 State**

The attribute of API_endpoint labeled "completion status" displays the status of the application's request to originate an outgoing call. This modeling may be useful for operating environments where polling is used to implement this state.

### 2.5.2.5. Incoming Call Preparation (A4)

In this state (as shown in Figure 4), the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

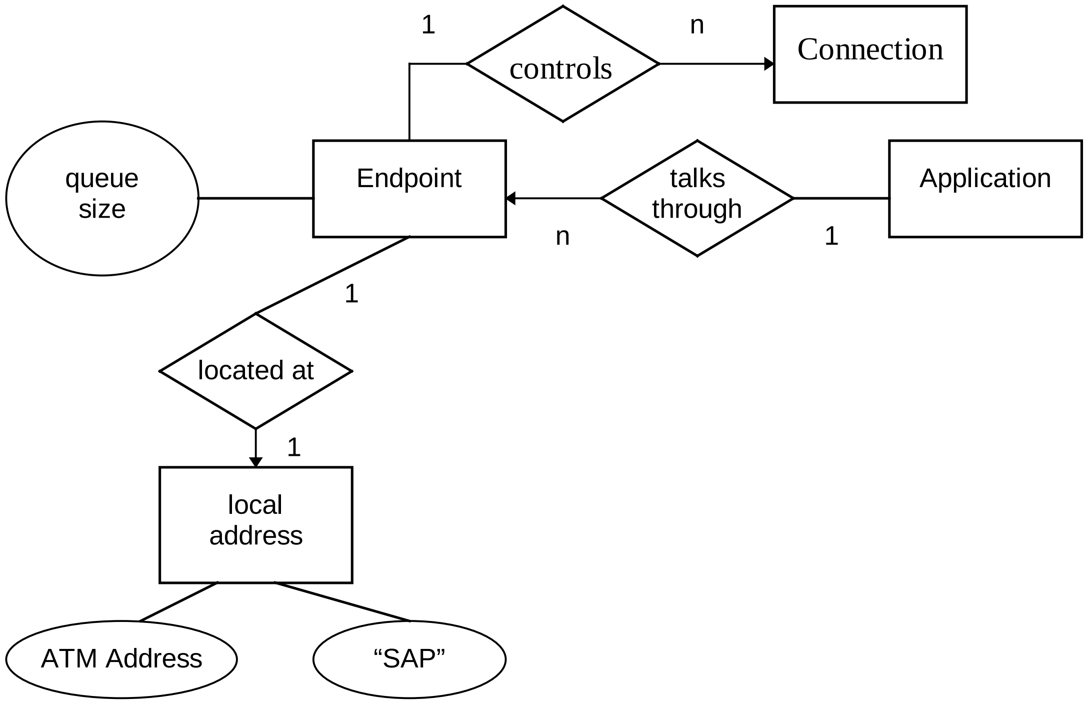

**Figure 4 E-R Diagram in A4 State**

The queue size specifies the number of incoming calls that may be held prior to the application's acceptance.

The API_endpoint is associated with a local address, which is the complete incoming call distribution specification for the application. The local address includes both the ATM address of the device housing the application, plus a "SAP" that allows the call notification to reach the correct application within the target device. See section 4.6 for more information on call distribution within an ATM device. Notice the one-to-one relationship ("located at") implies that only one API_endpoint can receive calls on any given local address.

#### 2.5.2.6. Wait Incoming Call (A5)

In this state, the application has access to an API_endpoint as shown in Figure 5. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

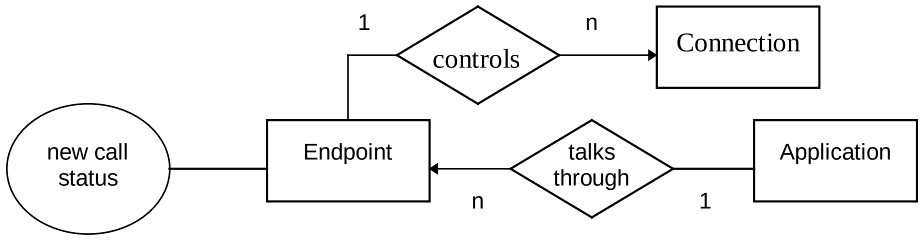

**Figure 5 E-R Diagram in A5 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).
The attribute of API_endpoint labeled "new call status" displays the existence of a new incoming call. This modeling may be useful for operating environments where polling is used to implement this state.

### 2.5.2.7. Incoming Call Present (A6)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

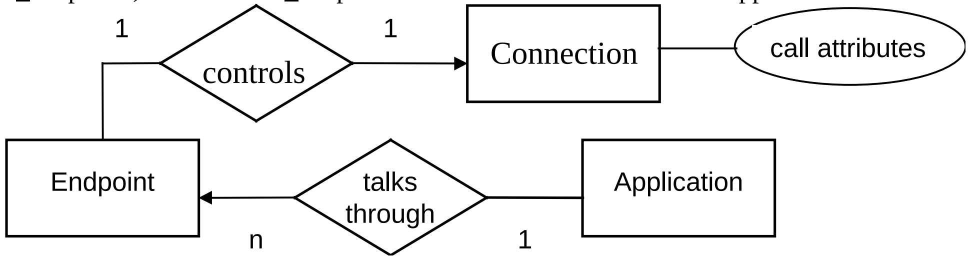

**Figure 6 E-R Diagram for A6 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-one as shown in Figure 6. Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

The queue that holds incoming calls for the application will have one and only one incoming call at the head of the queue. There exist "call attributes" that specify various characteristics of the call being received across the ATM network. These are modeled as an attribute of the entity "incoming call". The application issues primitives to query or set the attributes. Examples of these attributes would include AAL type, forward peak cell rate, quality of service class, and AAL1 clock frequency recovery method. Note that the number of attributes that may be set by the application are the set of parameters that are negotiated end-to-end: "Broadband low-layer information" is an example.

### 2.5.2.8. Incoming Call Requested (A7)

In this state, the application has access to an API_endpoint as shown in Figure 7. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

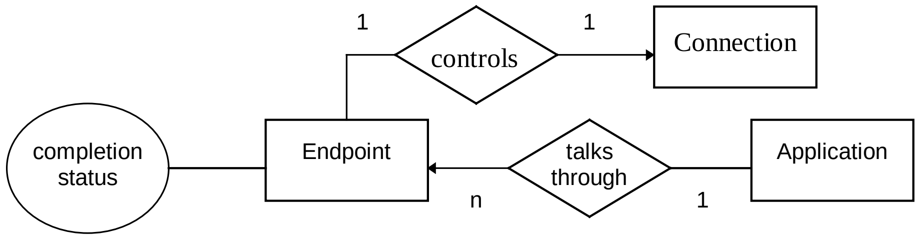

**Figure 7 E-R Diagram in A7 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-one. Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).
The attribute of API_endpoint labeled "completion status" displays the status of the application's request to accept an incoming call. This modeling may be useful for operating environments where polling is used to implement this state.

#### 2.5.2.9. Point-to-Point Data Transfer (A8)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application.

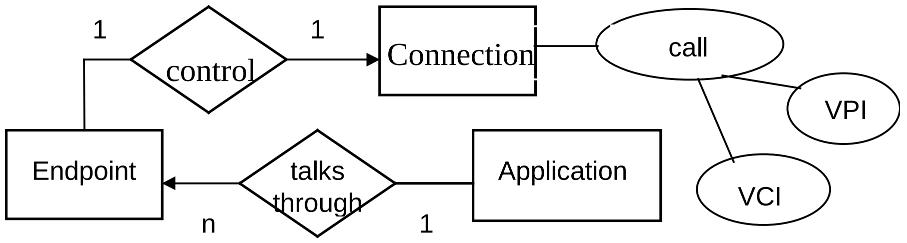

**Figure 8 E-R Diagram in A8 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-one as shown in Figure 8. Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

There exist "connection attributes" that specify various characteristics of the present call. These are modeled as an attribute of API_endpoint. The application issues primitives to query the attributes. Examples of these attributes would include AAL type, forward peak cell rate, quality of service class, and AAL1 clock frequency recovery method. Every API_endpoint in this state is associated with one and only one VCC that carries user data to/from the application. The VCC may be designated by a VPI and a VCI.

#### 2.5.2.10.Point-to-Multipoint Root Data Transfer (A9)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application as shown in Figure 9.

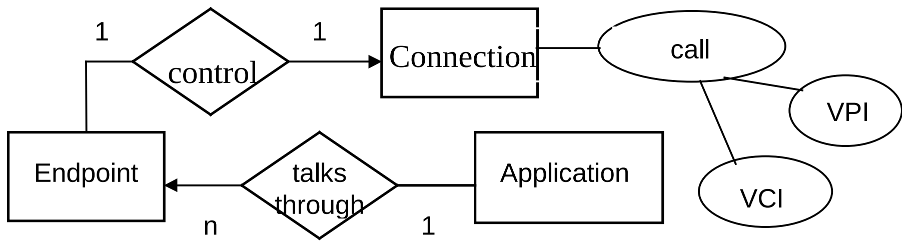

**Figure 9 E-R Diagram in A9 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-one. Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

There exist "connection attributes" that specify various characteristics of the present call. These are modeled as an attribute of API_endpoint. The application issues primitives to query the attributes. Examples of these attributes would include AAL type, forward peak cell rate, quality of service class, and AAL1 clock frequency recovery method.

Every API_endpoint in this state is associated with one and only one VCC that carries user data to/from the application. The VCC may be designated by a VPI and a VCI.

A "list of remote leafs" may be modeled as an attribute of API_endpoint. The remote leafs are other ATM devices that are leafs of a point-to-multipoint call, with the device housing API_endpoint as the root. The application uses primitives to modify this list, thereby adding and dropping parties of the point-to-multipoint call.

### 2.5.2.11.Point-to-Multipoint Leaf Data Transfer (A10)

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application as shown in Figure 10.

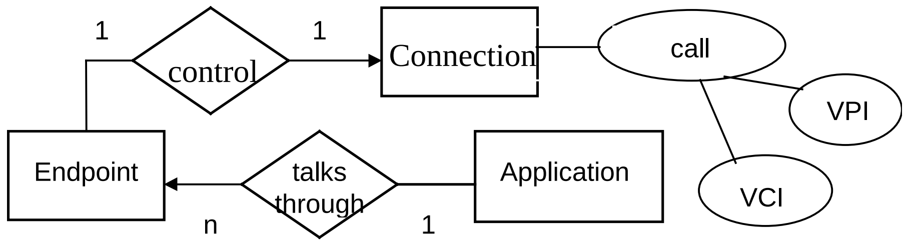

**Figure 10 E-R Diagram in A10 State**

In this state, the relationship between an API_endpoint and an API_connection is one-to-one. Each API_endpoint may be associated with a maximum of one API_connection, and vice-versa. The significance of this is that multiple API_connections may not share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

There exist “connection attributes” that specify various characteristics of the present call. These are modeled as an attribute of API_endpoint. The application issues primitives to query the attributes. Examples of these attributes would include AAL type, forward peak cell rate, quality of service class, and AAL1 clock frequency recovery method.

Every API_endpoint in this state is associated with one and only one VCC that carries user data to/from the application. The VCC may be designated by a VPI and a VCI.

#### 2.5.2.12.Connection Terminated (A11)

There is no entity-relationship diagram for this state.

The lifetime of API_connection is now over. There is no remaining association between an API_endpoint and the API_connection. Hence, there is no entity-relationship diagram needed for this state.

### 2.5.3. States L1 & L2

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application as shown in Figure 11.

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

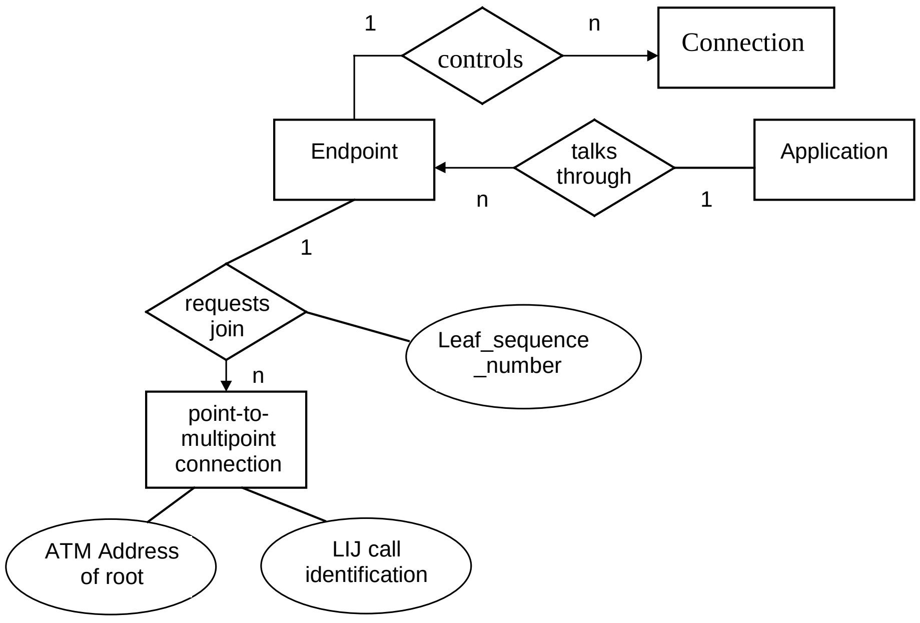

**Figure 11 Entity Relationship Diagram for Leaf ATM device**

An API_endpoint may initiate the joining of multiple point-to-multipoint connections. Each potential point-to-multipoint connection can be characterized by the ATM address of the connection’s root, plus the LIJ call identification, which is beyond the scope of this document and Signaling 4.0. Because an API_endpoint can have multiple join requests pending, the cardinality of the “requests join” relationship is one-to-many. Note that a given Pt-MPt connection can be associated with a maximum of one API_endpoint in the ATM device.

Because an API_endpoint can have multiple join requests pending, the API must correlate individual join requests with subsequent responses. The leaf_sequence_number is used to correlate individual LIJ_join_requests with subsequent responses. The leaf_sequence_number is administered by the NAS.

## 2.5.4. States R1, R2, & R3

In this state, the application has access to an API_endpoint. Native ATM Services are offered to the application via the exchange of primitives between the application and the API_endpoint. The relationship between an application and the API_endpoint is one-to-many; an application may exchange primitives with many API_endpoints, but each API_endpoint is bound to a maximum of one application as shown in Figure 12.

In this state, the relationship between an API_endpoint and an API_connection is one-to-many. A single API_endpoint may be associated with multiple API_connections, but each API_connection is associated with a maximum of one API_endpoint. The significance of this is that many API_connections may share a single API_endpoint in this state. This is done to model environments where an API_connection acquires a new API_endpoint as the API_connection progresses from the early states (call setup) to the data transfer states (A8, A9, A10).

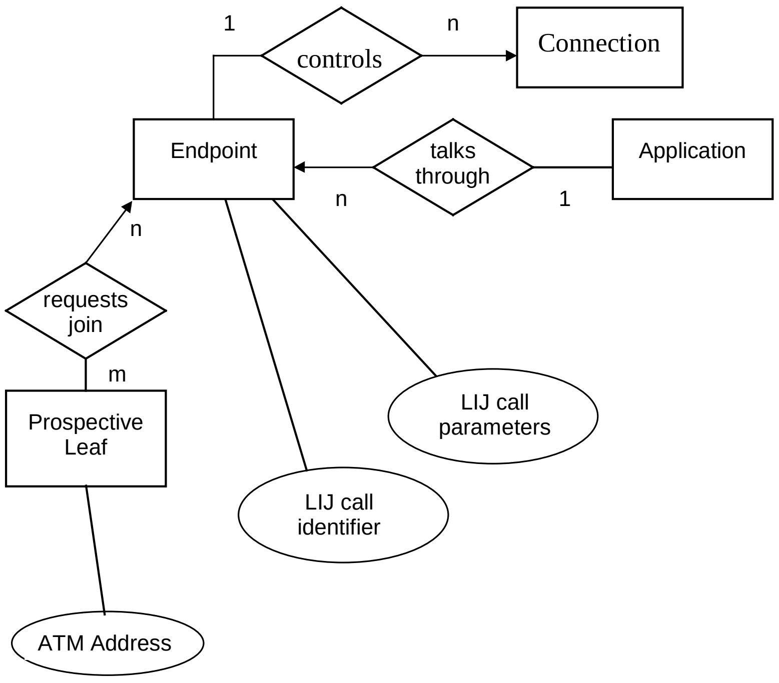

**Figure 12 Entity Relationship Diagram for the Root ATM device**

An API_endpoint may respond to join requests from other ATM devices. For a given point-to-multipoint connection, these join requests can be characterized by the ATM Address of the prospective leaf. A given point-to-multipoint connection can accommodate simultaneous requests from different prospective leafs. Also, any given leaf can simultaneously request a join to several different point-to-multipoint connections originating from the same ATM device . Because of this, the “requests join” relationship has the cardinality of many-to-many.

Any API_endpoint that responds to LIJ requests must have a LIJ call identifier that is unique for a given ATM address. The LIJ call identifier is beyond the scope of this document and Signaling 4.0. Also, any API_endpoint that responds to LIJ requests conforms to one or more LIJ call parameters. Both the LIJ call identifer and any LIJ call parameters are modeled as attributes of API_endpoint.

## 2.6. State Machines

The circles are the states and the edges are primitives that cause transitions between states. These primitives will be described further in chapter 3.

### 2.6.1. Partial State Diagram for API_Connection

Below in Figure 13 is a partial state diagram for API_connection.

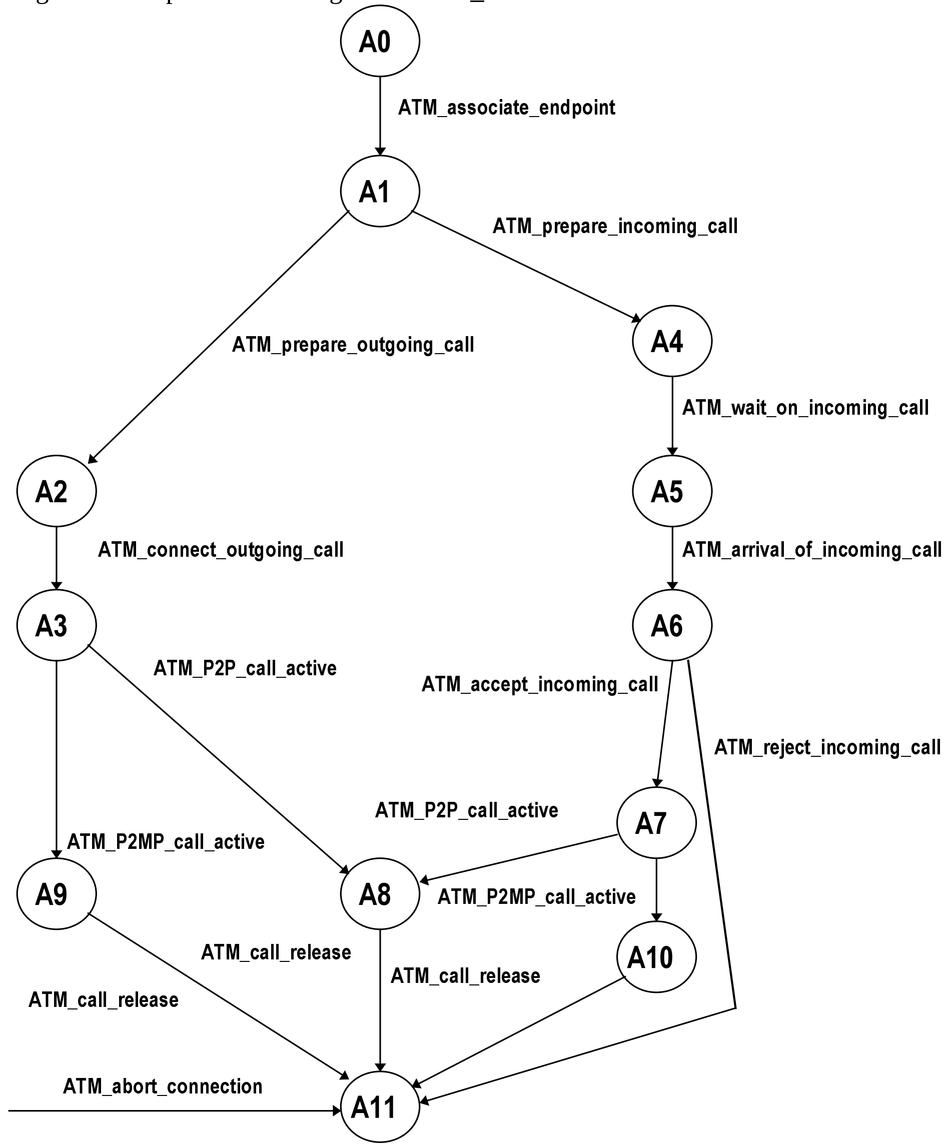

**Figure 13 Partial State Machine**

### 2.6.2. LIJ State Machines

Multiple LIJ state machines are permitted. An ATM device may simultaneously exist as a leaf within multiple multicast connection trees. An API_connection may only exist as a leaf with one multicast connection tree at any given time. An application may attempt to join multiple multicast connections essentially simultaneously. Refer to section 4 for more detailed procedures. Figure 14 below is the complete one-shot state diagram regarding the leaf initiated join request at the leaf. The leaf sequence number is used as a correlator at the LIJ leaf.

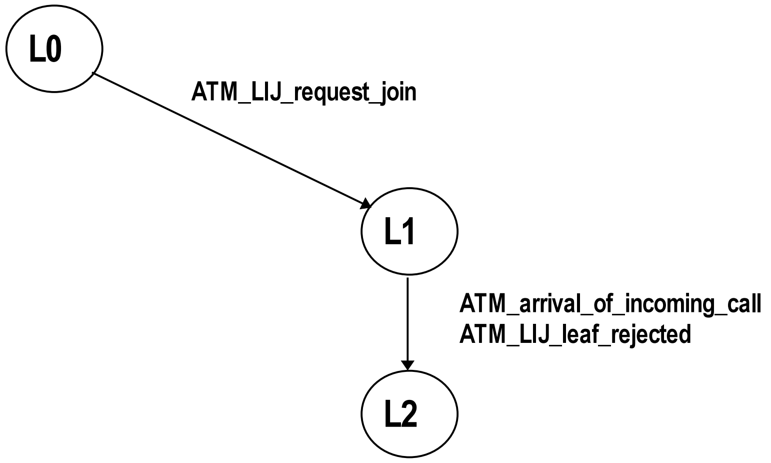

**Figure 14 LIJ Leaf State Machine**

Figure 15 provides the corresponding complete state machine at the root end of the Pt-MPt connection. At the root end, all state machines with the same *call_ID*, are synchronized during initialization by the **ATM_LIJ_associate_call_ID** primitive. The state machines then progress independently with each LIJ request. The ATM address of a leaf is used as a correlator at the LIJ Root.
.

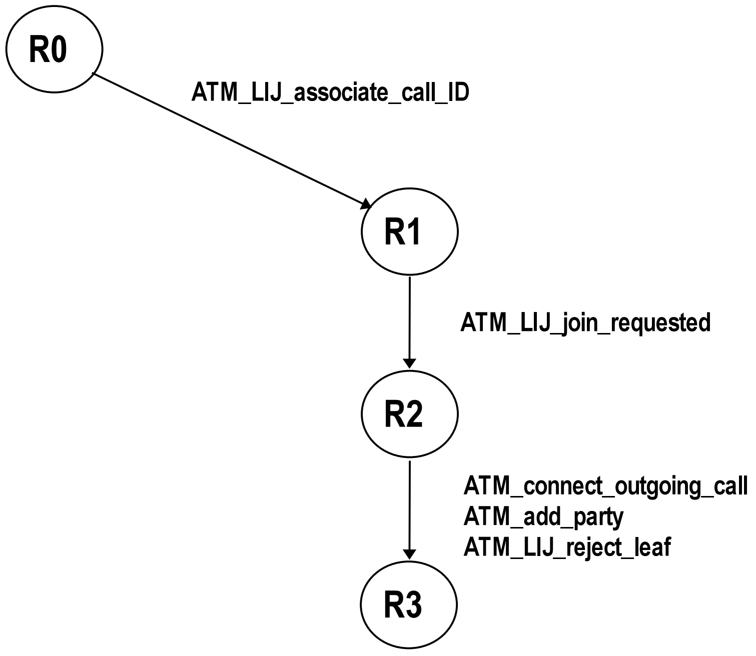

**Figure 15  LIJ Root State Machine**

# 3. PRIMITIVES

The symbol “&&” will be used to denote the logical AND operator. The symbol “||” will be used to denote the logical OR operator. This allows us to build Boolean expressions to specify the valid states for invocation of each primitive.

This section uses the following conventions for the primitives specified herein:

- "IN" - a parameter value supplied by the originator of the primitive (e.g. the caller of a subroutine).
- "OUT" - a parameter value supplied by the recipient of the primitive (e.g. the callee of a subroutine).
- "INOUT" - a parameter value that might be supplied by the primitive's originator, recipient, or both; depending on the implementation.
- "Request", "Indication", "Response", "Confirm" - these terms are conventionally used by ISO to describe the role of the primitive.

## 3.1. Control Plane

---

**ATM_abort_connection** **Request**

**Purpose:** clear the connection due to abnormal conditions

**ATM_abort_connection (**

**IN** *endpoint_identifier***,**

**IN** *cause*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *cause* specifies the abnormal condition

**Return Values:** none

**Valid States:** A1|| A2|| A3|| A4|| A5|| A6|| A7|| A8|| A9|| A10

**State Transitions:** the state becomes A11

This primitive ends the association between the API_connection and an API_endpoint.

This primitive is used to terminate a communication endpoint. All connections associated with this communication API_endpoint will be aborted and all the resources allocated for the communication API_endpoint are released.

**ATM_accept_incoming_call** **Response**

**Purpose:** application signals acceptance of the pending incoming call

**ATM_accept_incoming_call (**
**IN** *endpoint_identifier***,**
**INOUT** *new_endpoint_identifier*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *new_endpoint_identifier* specifies the new API_endpoint identifier to be associated with this API_connection. In cases where the parent application provides a new API_endpoint identifier, this parameter is IN. In case where the underlying service provides a new API_endpoint identifier, this parameter is OUT.

**Return Values:**

SUCCESS

CONNECTION_PREVIOUSLY_ABORTED

**Valid States:** A6

**State Transitions:**
- If the return value is SUCCESS, then the API_connection most recently presented to the application moves to state A7. This connection is controlled by new_endpoint_identifier. endpoint_identifier controls the remaining API_connections in the incoming queue, which will be in state A5.
- If the return value is CONNECTION_PREVIOUSLY_ABORTED, then the API_connection most recently presented to the application moves to state A11. This connection is controlled by new_endpoint_identifier. endpoint_identifier controls the remaining API_connections in the incoming queue, which will be in state A5.

This primitive signals that the application wishes to accept the incoming call that is at the head of the incoming call queue.

The connection attribute BLLI_SELECTOR is set by the application to one of the following values:
- 0: no “Broadband Low Layer Information” (BLLI) information element should be returned to the calling party
- 1: the first occurrence of BLLI information element should be returned to the calling party
- 2: the second occurrence of BLLI information element should be returned to the calling party
- 3 - the third occurrence of BLLI information element should be returned to the calling party.

**ATM_add_party** **Request**

**Purpose:** add a leaf to an existing point-to-multipoint call

**ATM_add_party (**
**IN** *endpoint_identifier***,**
**IN** *leaf_identifier***,**
**IN** *leaf_ATM_address*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_identifier* is reference to the ATM device being added to this connection. (NOTE: This value must be non-zero and less than 32,768.)
- *leaf_ATM_address* is the ATM address of the ATM device being added to this connection.

**Return Values:** none

**Valid States:** A9

**State Transitions:** If
(i) this API_endpoint is the root of a LIJ point-to-multipoint connection,
(ii) the LIJ FSM is in state R2, and
(iii) parameter *leaf_ATM_address* specifies the ATM address of the leaf currently attempting to join the connection;
then invocation of this primitive moves the LIJ FSM to state R3.

This primitive allows applications to add new ATM devices to an existing point-to-multipoint call. The new leaf inherits the traffic parameters already in force for the existing leaf(s) of this connection. Note that data traffic in a point-to-multipoint connection only flows from the root node to each leaf node.

**ATM_add_party_reject** **Confirm**

**Purpose:** signal that a previous ATM_add_party request was unsuccessful.

**ATM_add_party_reject (**
**IN** *endpoint_identifier***,**
**IN** *leaf_identifier***,**
**IN** *rejection_cause*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_identifier* is a reference to the ATM device that could not be added.
- *rejection_cause* specifies why the addition of a leaf was rejected.

**Return Values:** none

**Valid States:** A9

**State Transitions:** no transitions

---

**ATM_add_party_success** **Confirm**

**Purpose:** signal that a previous ATM_add_party request was successful.

**ATM_add_party_success (**
 **IN** *endpoint_identifier***,**
 **IN** *leaf_identifier***,**
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_identifier* is reference to the ATM device being added to this connection.

**Return Values:** none

**Valid States:** A9

**State Transitions:** no transitions

---

**ATM_arrival_of_incoming_call** **Indication**

**Purpose:** notifies the application that a call has arrived

**ATM_arrival_of_incoming_call (**
 **IN** *endpoint_identifier***,**
**)**

where

- *endpoint_identifier* specifies the incoming call queue upon which the call is located.

**Return Values:** none

**Valid States:** A5

**State Transitions:** The API_connection that is presented to the application moves from state A5 to A6. All other API_connections for this endpoint_identifier remain in state A5.
If

(i) this API_endpoint is attempting to become a leaf of a LIJ point-to-multipoint connection,
(ii) the LIJ FSM is in state L1, and
(iii) the connection attributes specify the leaf sequence number corresponding to a previous LIJ request. Refer to Annex A for connection attributes.

then invocation of this primitive moves the LIJ FSM to state L2.

This primitive signals the application that an incoming call is present in the incoming call queue.

The connection attribute BLLI_SELECTOR is set by Native ATM Services to one of the following values:

- 0: no “Broadband Low Layer Information” (BLLI) information element was present in the incoming call
- 1: the first occurrence of BLLI information element present in the incoming call forms part of the SAP (over which this call is being received)
- 2: the second occurrence of BLLI information element present in the incoming call forms part of the SAP (over which this call is being received)
- 3 - the third occurrence of BLLI information element present in the incoming call forms part of the SAP (over which this call is being received).

---

**ATM_associate_endpoint** **Request**

**Purpose:** associates an API_endpoint with the potential API_connections that will use that endpoint.

**ATM_associate_endpoint (**
**OUT** *endpoint_identifier***,**
**)**

where

- *endpoint_identifier* is the newly allocated API_endpoint.

**Return Values:** none

**Valid States:** A0

**State Transitions:** API_connection moves to state A1. This primitive initializes any potential root LIJ FSMs to R0 and any potential leaf LIJ FSMs to L0.

This primitive creates an initial association between the API_connection and API_endpoint.

**ATM_call_release** **Request**

**Purpose:** terminates an API_connection that is in an active state (A8|| A9|| A10) by the application.

**ATM_call_release (**
**IN** *endpoint_identifier*,
**IN** *release_cause*,
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *release_cause* identifies the cause of release.

**Return Values:** none

**Valid States:** A8|| A9|| A10

**State Transitions:** API_connection moves to state A11

**ATM_call_release** **Indication**

**Purpose:** terminates an API_connection because of a network or remote device action.

**ATM_call_release (**
**IN** *endpoint_identifier*,
**IN** *release_cause*,
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *release_cause* identifies the cause of release.

**Return Values:** none

**Valid States:** A3|| A7|| A8|| A9|| A10

**State Transitions:** API_connection moves to state A11

**ATM_connect_outgoing_call** **Request**

**Purpose:** initiates a call across the ATM network

**ATM_connect_outgoing_call (**
**IN** *endpoint_identifier***,**
**IN** *destination_SAP,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *destination_SAP* is the address of the target ATM device and target entity within the ATM device. See section 4.4 and section 4.5 for more information on SAP addresses.

**Return Values:** none

**Valid States:** A2

**State Transitions:** The API_Connect state machine moves to state A3.
If
(i) this API_endpoint is the root of a LIJ point-to-multipoint connection,
(ii) the LIJ FSM is in state R2, and
(iii) parameter *destination_SAP* specifies the ATM address of the leaf currently attempting to join the connection;
then invocation of this primitive moves the LIJ FSM to state R3.

This primitive causes the underlying ATM protocols to attempt set up of an outgoing call.

**ATM_drop_party** **Request**

**Purpose:** remove an ATM device from a point-to-multipoint call.

**ATM_drop_party (**
**IN** *endpoint_identifier***,**
**IN** *leaf_identifier***,**
**IN** *drop_cause*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_identifier* specifies the ATM device being dropped.
- *drop_cause* specifies the reason for dropping.

**Return Values:** none

**Valid States:** A9

**State Transitions:** no state transitions

This primitive allows applications to drop a leaf node from a point-to-multipoint connection.

---

**ATM_drop_party** **Indication**

**Purpose:** an ATM device (leaf of point-to-multipoint call) released the connection.

**ATM_drop_party (**
**IN** *endpoint_identifier***,**
**IN** *leaf_identifier***,**
**IN** *drop_cause*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_identifier* specifies the ATM device that dropped off of the connection.
- *drop_cause* specifies the reason for dropping.

**Return Values:** none

**Valid States:** A9

**State Transitions:** no state transitions

This primitive notifies an application that a remote leaf node dropped off a point-to-multipoint connection.

---

**ATM_get_local_port_info** **Request**

**Purpose:** obtain necessary information related to an ATM physical port

**ATM_get_local_port_info (**
**IN** *port_number***,**
**OUT** *address_list*
**OUT** *line_rate*
**)**

where

- *port_number* identifies a physical UNI attachment.
- *address_list* is a list of ATM addresses that are valid for the port number.
- *line_rate* is the maximum cell rate supported by the ATM port in cells per second.

**Return Values:**

SUCCESS

INVALID_PORT_NUMBER

NO_VALID_ADDRESSES

**Valid States:** not applicable

**State Transitions:** not applicable

This primitive reports the result of ILMI address registration, as specified in section 5.8 of the UNI.

---

**ATM_P2MP_call_active** **Confirm**

**Purpose:** signal that the point-to-multipoint call is now in active state

**ATM_P2MP_call_active (**

**IN** *endpoint_identifier*,

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.

**Return Values:** none

**Valid States:** A3|| A7

**State Transitions:** If the API_connection was in state A3, then the new state will be A9. If the API_connection was in state A7, then the new state will be A10.

---

**ATM_P2P_call_active** **Confirm**

**Purpose:** signal that the point-to-point call is now in active state

**ATM_P2P_call_active (**

**IN** *endpoint_identifier*,

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.

**Return Values:** none

**Valid States:** A3|| A7

**State Transitions:** API_connection moves A8.

---

**ATM_prepare_incoming_call** **Request**

**Purpose:** sets up incoming call distribution tables

**ATM_prepare_incoming_call (**
**IN** *endpoint_identifier,*
**IN** *local_SAP,*
**IN** *queue_size*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *local_SAP* is the address of this ATM device and application entity. See section 4.4 for a discussion on SAP addresses.
- *queue_size* specifies the depth of the incoming call queue.

**Return Values:**
SUCCESS
SAP_ALREADY_USED

**Valid States:** A1

**State Transitions:** API_connection moves to state A4

This primitive allows applications to associate an API_endpoint with a local address. The intent of the 2nd parameter is to specify an address that can be used to identify a unique API_endpoint.

---

**ATM_prepare_outgoing_call** **Request**

**Purpose:** sets up data structures that hold the characteristics of the outgoing call

**ATM_prepare_outgoing_call (**
**IN** *endpoint_identifier,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.

**Return Values:** none

**Valid States:** A1

**State Transitions:** API_connection moves to state A2.

---

**ATM_query_connection_attributes** **Request**

**Purpose:** to obtain an attribute value of the connection

**ATM_query_connection_attributes (**
**IN** *endpoint_identifier,*
**IN** *attribute_name,*
**OUT** *attribute_value,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *attribute_name* identifies the attribute.
- *attribute_value* is the value of attribute.

**Return Values:**
SUCCESS
ATTRIBUTE_DOES_NOT_EXIST

**Valid States:** A2|| A6|| A8|| A9|| A10

**State Transitions:** no state transitions

This primitive allows applications to query connection attributes. The attribute_name parameter specifies the name of the attribute (e.g. QoS, peak cell rate). The attribute_value parameter contains the query result.

NOTE: Native ATM Services must provide access to as many as three instances of the connection attributes found in the "Broadband Low Layer Information" information element. One convenient way to do this is to use connection attribute BLLI_SELECTOR to select which of the three instances is being queried. Such implementations should default BLLI_SELECTOR to a value of 1 when API_connection transitions from state A1 to A2.

**ATM_reject_incoming_call** **Response**

**Purpose:** signals that the application does not accept the incoming call

**ATM_reject_incoming_call (**
**IN** *endpoint_identifier,*
**IN** *rejection_cause*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *rejection_cause* specifies why the connection is being rejected.

**Return Values:** none

**Valid States:** A6

**State Transitions:** API_connection moves to state A11

This primitive signals that the application wishes to reject the incoming call that is at the head of the incoming call queue. endpoint_identifier controls the remaining API_connections in the incoming queue, which will be in state A5.

**ATM_set_connection_attributes** **Request**

**Purpose:** to modify an attribute value of the connection

**ATM_set_connection_attributes (**
**IN** *endpoint_identifier,*
**IN** *attribute_name,*
**IN** *attribute_value,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *attribute_name* identifies the attribute.
- *attribute_value* is the desired value of attribute.

**Return Values:**
SUCCESS
ATTRIBUTE_DOES_NOT_EXIST
CAN_NOT_MODIFY
INVALID_VALUE

**Valid States:** A2|| A6

**State Transitions:** no state transitions

This primitive allows applications to modify connection attributes. The attribute_name parameter specifies the name of the attribute (e.g. QoS, peak cell rate). The attribute_value parameter contains the desired new value.

NOTE: Native ATM Services must provide access to as many as three instances of the connection attributes found in the "Broadband Low Layer Information" information element. One convenient way to do this is to use connection attribute BLLI_SELECTOR to select which of the three instances is being manipulated. Such implementations should default BLLI_SELECTOR to a value of 1 when API_connection transitions from state A1 to A2.

---

**ATM_wait_on_incoming_call** **Request**

**Purpose:** activates the incoming call distribution function for this endpoint.

**ATM_wait_on_incoming_call (**
**IN** *endpoint_identifier,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.

**Return Values:** none

**Valid States:** A4

**State Transitions:** All API_connections associated with this API_endpoint move to state A5

This primitive causes the application to wait for an incoming call to enter the incoming call queue.

---

**ATM_LIJ_associate_call_ID** **Request**

**Purpose:** associates an API_endpoint with a locally unique leaf-initiated join call identifier.

**ATM_LIJ_associate_call_ID (**

**IN** *endpoint_identifier***,**

**IN** *LIJ_call_identifier***,**

**IN** *LIJ_call_parameters***,**

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *LIJ_call_identifier* specifies a value (unique to this ATM device ) that is used to identify a particular leaf-initiated point-to-multipoint connection.
- *LIJ_call_parameters* specifies whether or not the root is notified of leaf-initiated join requests.

**Return Values:**
SUCCESS
LIJ_ID_ALREADY_USED
LIJ_ONLY_ONE_ID_ALLOWED

**Valid States:** A2 && R0

**State Transitions:** The primitive moves any potential root LIJ FSMs to R1.

---

**ATM_LIJ_request_join** **Request**

**Purpose:** a leaf uses this primitive to attempt to join a point-to-multipoint connection.

**ATM_LIJ_request_join (**

**IN** *endpoint_identifier***,**

**IN** *root_ATM_address***,**

**IN** *LIJ_call_identifier***,**

**OUT** *leaf_sequence_number*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *root_ATM_address* is the ATM address of the connection's root.
- *LIJ_call_identifier* specifies a value (unique to the root's ATM device ) that is used to identify a particular leaf-initiated point-to-multipoint connection.
- leaf_sequence_number specifies a unique value for a multicast connection session. It is used to identify one of potentially several simultaneous multicast connections.

**Return Values:** none

**Valid States:** (A4 || A5) && L0

**State Transitions:** The leaf's LIJ FSM moves to state L1.

---

**ATM_LIJ_join_requested** **Indication**

**Purpose:** notifies the root that a leaf is attempting to join a point-to-multipoint connection.

**ATM_LIJ_join_requested (**
**IN** *endpoint_identifier*,
**IN** *leaf_ATM_address*,
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_ATM_address* is the ATM address of the leaf attempting to join the connection.

**Return Values:** none

**Valid States:** (A2 || A9) && R1

**State Transitions:** The root's LIJ FSM moves to state R2.

---

**ATM_LIJ_reject_leaf** **Response**

**Purpose:** reject leaf's attempt to join a point-to-multipoint connection.

**ATM_LIJ_reject_leaf (**
**IN** *endpoint_identifier*,
**IN** *leaf_ATM_address*,
**IN** *rejection_cause*,
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *leaf_ATM_address* is the ATM address of the leaf attempting to join the connection.
- *rejection_cause* specifies why the join attempt is being rejected.

**Return Values:** none

**Valid States:** (A2 || A9) && R2

**State Transitions:** The root's LIJ FSM moves to state R3.

---

**ATM_LIJ_leaf_rejected** **Confirm**

**Purpose:** notifies leaf that an attempt to join a point-to-multipoint connection was rejected.

**ATM_LIJ_leaf_rejected (**

**IN** *endpoint_identifier***,**

**IN** *rejection_cause***,**

**IN** *leaf_sequence_number*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *rejection_cause* specifies why the join attempt was rejected.
- leaf_sequence_number specifies a unique value for a multicast connection session. It is used to identify one of potentially several simultaneous multicast connections.

**Return Values:** none

**Valid States:** A5 && L1

**State Transitions:** The leaf's LIJ FSM moves to state L2.

This primitive must be locally generated by Native ATM Services when the current pending request to join a point-to-multipoint connection times out.

---

## 3.2. Data Plane

There are three major approaches to the implementation of the data transfer primitives:

- polling
- blocking
- messaging

Another issue is the location of the data. The data bound for the application could exist in:

- memory space that the operating system kernel manages
- memory space that the application manages
- hardware (e.g. the ATM adapter card, the video card, the sound card, a private data bus)

It is possible that there is a total hardware path between the ATM function and an application implemented in hardware. In this case, the data plane primitives are realized in hardware.

Implementation choices are outside the scope of this document. However, for completeness, this section considers the differing semantics associated with these different implementation approaches. See Appendix A for a discussion of pragmatic issues concerning these primitives.

NOTE: the data plane primitives to support AAL1 and User defined AAL traffic are implementation specific at this time. This topic is for further study.

### 3.2.1. Sending Data

**ATM_send_data** **Request**

**Purpose:** to send data on the API_connection

**ATM_send_data (**
**IN** *endpoint_identifier,*
**IN** *data_source,*
**OUT** *sending_result*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *data_source* describes the data. For example, it could be a buffer location and amount of data, or even a collection of buffers. As another example, it could be an instance of the application's data to be transmitted across the ATM network.
- *sending_result* is a status indication. In some implementations, it is simply SUCCESS or FAILURE. In other implementations, it could be the amount of data transferred.

**Return Values:**
SUCCESS
NO_CONNECTION

**Valid States:** A8|| A9

**State Transitions:** no state transition

### 3.2.2. Receiving Data
Only one of the following implementations needs to be supported in a conforming system.

#### 3.2.2.1. Polling Implementation

**ATM_receive_data** **Request**

**Purpose:** to receive data on the API_connection

**ATM_receive_data (**

**IN** *endpoint_identifier,*

**INOUT** *data_receptor,*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *data_receptor:*

  Before this primitive is invoked, *data_receptor* describes where the received data should be placed. After this primitive returns, *data_receptor* describes the data.

  For example, before the primitive is invoked, *data_receptor* could be a buffer location and buffer size. After the primitive returns, *data_receptor* would be a buffer location and amount of data.

  As another example, before the primitive is invoked, *data_receptor* could be a character string of zero length and a maximum string size. After the primitive returns, data_receptor would be a character string.

**Return Values:**
SUCCESS
DATA_NOT_PRESENT
NO_CONNECTION

**Valid States:** A8|| A10

**State Transitions:** no state transition

---

#### 3.2.2.2. Blocking Implementation

**ATM_receive_data** **Request**

**Purpose:** to receive data on the API_connection

**ATM_receive_data (**

**IN** *endpoint_identifier,*

**INOUT** *data_receptor,*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *data_receptor:*

  Before this primitive is invoked, *data_receptor* describes where the received data should be placed. After this primitive returns, *data_receptor* describes the data.

For example, before the primitive is invoked, *data_receptor* could be a buffer location and buffer size. After the primitive returns, *data_receptor* would be a buffer location and amount of data.

As another example, before the primitive is invoked, *data_receptor* could be a character string of zero length and a maximum string size. After the primitive returns, *data_receptor* would be a character string.

**Return Values:**
SUCCESS
NO_CONNECTION

**Valid States:** A8|| A10

**State Transitions:** no state transition

---

### 3.2.2.3. Messaging Implementation

**ATM_receive_data** **Indication**

**Purpose:** to receive data on the API_connection

**ATM_receive_data (**
**IN** *endpoint_identifier,*
**IN** *receive_data*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *receive_data* describes the data. Consider the following examples for this parameter:
    1. a buffer location and the amount of data
    2. an instance of the data (received across the ATM network) traversing a bus from the ATM hardware to some other hardware device
    3. an indication to an implementation-specific "receive handler" (see below) that the data either has been or is in the process of being received across the ATM network.

**Return Values:** none

**Valid States:** A8|| A10

**State Transitions:** no state transition

In some cases, a “receive handler” would be installed in the system. An environment-specific function is performed to install the receive handler. The receive handler becomes the recipient of the ATM_receive_data primitive. Additionally, the sending application could include application-defined control information in an AAL frame and the application-defined receive handler could interpret this control information.

### 3.2.3. Flow Control Management

Flow control management primitives apply for connections using the ABR service only. These primitives provide a way to be notified of the Allowed Cell Rate and to control the Allowed Cell Rate within the limit of the traffic contract negotiated at connection establishment. These primitives are available at both the source and destination. The Allowed Cell Rate can be retrieved directly (polling) or indicated (asynchronous) as specified. The variation of the Allowed Cell Rate must follow the Traffic Management Specification. These primitives only provide a way to specify a requested Cell Rate, not to force its value.

Appendix D provides informative material regarding the flow control management primitives associated with the ABR service.

#### 3.2.3.1. Source Cell Rate Notification

An application might choose to tune its outbound traffic according to the actual available bandwidth. The following primitives allow an application to be notified of the Allowed Cell Rate as a source.

**ATM_query_outbound_rate** **Request**

**Purpose:** Query the ACR (Allowed Cell Rate) used to send U-Plane data in the outbound direction.

**ATM_query_outbound_rate (**
**IN** *endpoint_identifier,*
**OUT** *allowed_cell_rate*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *allowed_cell_rate* specifies the Allowed Cell Rate used to send in Cells/sec.

**Return Values:** none

**Valid States:** A8 | | A9

**State Transitions:** no state transition

**ATM_set_outbound_notification_threshold** **Request**

**Purpose:** specifies the thresholds for outbound data rate values (ACR) that trigger asynchronous notifications to the application.

**ATM_set_outbound_notification_threshold(**

**IN** *high,*

**IN** *low*

**)**

where

- *high* specifies in Cells/sec the upper threshold of ACR for triggering asynchronous notifications
- *low* specifies in Cells/sec the lower threshold of ACR for triggering asynchronous notifications

**Return Values:** none

**Valid States:** A8 | | A9

**State Transitions:** no state transition

**Note:** Implementations that support the ATM_outbound_rate_changed primitive will use this primitive to specify conditions under which the ATM_outbound_rate_changed indication primitive shall be invoked. An implementation may internally translate the threshold parameters to some other convenient metric (e.g. queue length) .

---

**ATM_outbound_rate_changed** **Indication**

**Purpose:** notifies the application that the outbound rate (Allowed Cell Rate) has exceeded the high threshold, or fallen below the low threshold.

**ATM_outbound_rate_changed(**

**IN** *endpoint_identifier,*

**IN OPTIONAL** *allowed_cell_rate*

**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *allowed_cell_rate* specifies ( in cells/sec) the Allowed Cell Rate used to send.

**Return Values:** none

**Valid States:** A8 | | A9

**State Transitions:** no state transition

**Note:** An implementation may or may not support this indication. If supported, this indication depends on the ATM_set_outbound_notification_threshold primitive to establish and maintain the high and low thresholds. If the optional allowed_cell_rate parameter is not supplied, then the application may determine the value of this parameter by use of the ATM_query_outbound_rate primitive.

#### 3.2.3.2. Source Cell Rate Control

An application might be willing to give up some of the Traffic Contract Quality Of Service, provided it can be granted full benefit when needed. The following primitive provides a way to request an Allowed Cell Rate different from the Peak Rate.

**ATM_request_outbound_rate** **Request**

**Purpose:** to request to change the outbound rate (Explicit Rate) from the source.

**ATM_request_outbound_rate(**
**IN** *endpoint_identifier,*
**IN** *requested_cell_rate*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- The *requested_cell_rate* in cells/sec.

**Return Values:**
SUCCESS
INVALID_VALUE

**Valid States:** A8|| A9

**State Transitions:** no state transition

**Note:** The actual Allowed Cell Rate achieved and the speed of the adaptation depends on the Traffic Contract, the network load and traffic management policy as well as on the destination. The value of the requested_cell_rate must be compatible with the existing traffic contract i.e. less than Peak Cell Rate (PCR) and also greater than or equal to the Minimum Cell Rate (MCR). Values of requested_cell_rate that do not meet the traffic contract will result in an appropriate error code (INVALID_VALUE).

#### 3.2.3.3. Destination Cell Rate Notification

An application might choose to tune its inbound traffic according to the actual available bandwidth. The following primitives allow an application to be notified of the Allowed Cell Rate as a source.

**ATM_query_inbound_rate** **Request**

**Purpose:** Query the ACR (Allowed Cell Rate) used to receive U-Plane data in the inbound direction.

**ATM_query_inbound_rate (**
**IN** *endpoint_identifier,*
**OUT** *allowed_cell_rate*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *allowed_cell_rate* specifies the Allowed Cell Rate used to send in Cells/sec.

**Return Values:** none

**Valid States:** A8 | | A10

**State Transitions:** no state transition

---

**ATM_set_inbound_notification_threshold** **Request**

**Purpose:** specifies the thresholds for inbound data rate values (ACR) that trigger asynchronous notifications to the application.

**ATM_set_inbound_notification_threshold(**
**IN** *high,*
**IN** *low*
**)**

where

- *high* specifies in Cells/sec the upper threshold of ACR for triggering asynchronous notifications
- *low* specifies in Cells/sec the lower threshold of ACR for triggering asynchronous notifications

**Return Values:** none

**Valid States:** A8 | | A10

**State Transitions:** no state transition

**Note:** Implementations that support the ATM_inbound_rate_changed primitive will use this primitive to specify conditions under which the ATM_inbound_rate_changed indication primitive shall be invoked. An implementation may internally translate the threshold parameters to some other convenient metric (e.g. queue length) .

**ATM_inbound_rate_changed** **Indication**

**Purpose:** notifies the application that the inbound rate (Allowed Cell Rate) has exceeded the high threshold, or fallen below the low threshold.

**ATM_inbound_rate_changed(**
**IN** *endpoint_identifier,*
**IN OPTIONAL** *allowed_cell_rate*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *allowed_cell_rate* specifies ( in cells/sec) the Allowed Cell Rate used to send.

**Return Values:** none

**Valid States:** A8 | | A10

**State Transitions:** no state transition

**Note:** An implementation may or may not support this indication. If supported, this indication depends on the ATM_set_inbound_notification_threshold primitive to establish and maintain the high and low thresholds. If the optional allowed_cell_rate parameter is not supplied, then the application may determine the value of this parameter by use of the ATM_query_inbound_rate primitive.

### 3.2.3.4. Destination Cell Rate Control

An application might be willing to give up some of the Traffic Contract Quality Of Service, provided it can be granted full benefit when needed. The following primitive provides a way to request an Allowed Cell Rate different from the Peak Rate.

**ATM_request_inbound_rate** **Request**

**Purpose:** to request to change the inbound rate (Explicit Rate) .

**ATM_request_inbound_rate(**
**IN** *endpoint_identifier,*
**IN** *requested_cell_rate*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- The *requested_cell_rate* in cells/sec.

**Return Values:**
SUCCESS
INVALID_VALUE

**Valid States:** A8|| A10

**State Transitions:** no state transition

**Note:** The actual Allowed Cell Rate achieved and the speed of the adaptation depends on the Traffic Contract, the network load and traffic management policy as well as on the destination. The value of the requested_cell_rate must be compatible with the existing traffic contract i.e. less than Peak Cell Rate (PCR) and also greater than or equal to the Minimum Cell Rate (MCR). Values of requested_cell_rate that do not meet the traffic contract will result in an appropriate error code (INVALID_VALUE).

## 3.3. Management Plane

**ATM_confirm_loopback** **Confirm**

**Purpose:** confirms completion of loopback test

**ATM_confirm_loopback (**
**IN** *endpoint_identifier***,**
**IN** *correlator*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *correlator* allows the management application to match this confirmation to a previous ATM_initiate_loopback request.

**Return Values:** none

**Valid States:** A8

**Note:** When this confirmation is received, it means that the loopback test was successful. If no confirmation to a corresponding ATM_intiate_loopback is received, then the loopback test failed.

---

**ATM_indicate_error** **Indication**

**Purpose:** indicates an error was detected among the managed objects.

**ATM_indicate_error (**
**IN** *error_code,*
**OPTIONAL IN** *endpoint_identifier,*
**)**

where

- *error_code* is the error that occurred.
- *endpoint_identifier* specifies the connection to which this primitive applies, when *error_code* identifies an error that is associated with a single connection. Otherwise, the value of *endpoint_identifier* is unspecified.

**Return Values:** none

**Error_Code Values:**

**Notes:**

1. These are network-oriented error messages. They arise from errors detected over the UNI. These values are for further study.

2. Only A0 (null) and A11 (call terminated) states cannot generate an error when an API_endpoint identifier is provided.

---

**ATM_indicate_fault_alert** **Indication**

**Purpose:** indicates reception of an OAM alarm signal

**ATM_indicate_fault_alert (**
**IN** *endpoint_identifier,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.

**Return Values:** none

**Valid States:** A8

---

**ATM_initiate_loopback** **request**

**Purpose:** initiates a loopback test to the nearest switch or the far end point of a connection.

**ATM_initiate_loopback (**
**IN** *endpoint_identifier,*
**IN** *loopback_extent,*
**IN** *correlator,*
**)**

where

- *endpoint_identifier* specifies the connection to which this primitive applies.
- *loopback_extent* specifies nearest switch loopback or end-to-end loopback.
- *correlator* is a value that will be returned with the ATM_confirm_loopback primitive.

**Return Values:** none

**Valid States:** A8

**ATM_query_mgmt_variable** **Request**

**Purpose:** queries the value of the specified management variable.

**ATM_query_mgmt_variable (**

**IN** *variable_name,*

**IN** *table_index,*

**OUT** *variable_value,*

**)**

where

- *variable_name* identifies the managed object.
- *table_index* identifies the table location when this variable is located in a table. For variables not located in a table, this parameter is ignored.
- *variable_value* is the value of the managed object.

**Return Values:**
SUCCESS
VARIABLE_NOT_SUPPORTED
INDEX_OUT_OF_RANGE
VARIABLE_DOES_NOT_EXIST

**ATM_set_mgmt_variable** **Request**

**Purpose:** modifies the value of the specified management variable

**ATM_set_mgmt_variable (**

**IN** *variable_name,*

**IN** *table_index,*

**IN** *desired_value,*

**)**

where

- *variable_name* identifies the managed object.
- *table_index* identifies the table location when this variable is located in a table. For variables not located in a table, this parameter is ignored.
- *desired_value* is the desired value of the managed object.

**Return Values:**
SUCCESS
VARIABLE_NOT_SUPPORTED

INDEX_OUT_OF_RANGE
VARIABLE_DOES_NOT_EXIST
CAN_NOT_MODIFY
INVALID_VALUE

# 4. Procedures

## 4.1. PVC Provisioning

These procedure shall be limited to trusted applications, such as a management application. General user applications that use a PVC shall only receive an indication that a PVC is active.

### 4.1.1. Establishment

The management application shall perform the following steps in order to provision a PVC:

1. Choose an unused table entry in the atmfVccGroup.
2. Using the **ATM_set_mgmt_variable** primitive, set all the parameters for this table entry of the atmfVccGroup except for atmfVccOperStatus. It is presumed that atmfVccOperStatus has the value unknown or localDown.
3. Using the **ATM_set_mgmt_variable** primitive, set atmfVccOperStatus in atmfVccGroup to localUpEnd2endUnknown. This is a signal to native ATM services to perform all locally required setup of the VC. This includes hardware register manipulation and buffer allocation. During this time atmfVccEndpointIdentifier shall be assigned by native ATM services. The value for atmfVccOperStatus will retain the value of unknown or localDown until the initialization of the VC is complete. After the initialization is completed atmfVccOperStatus is set to localUpEnd2endUnknown by Native ATM services.
4. Using the **ATM_query_mgmt_variable** primitive, poll and query atmfVccOperStatus in atmfVccGroup until the value is localUpEnd2endUnknown.
5. Using the **ATM_query_mgmt_variable** primitive, obtain the value for atmfVccEndpointIdentifier and pass it to the entity that is to use this PVC.

### 4.1.2. Termination

The management application shall perform the following steps in order to terminate a PVC:

1. Using the **ATM_set_mgmt_variable** primitive, set atmfVccOperStatus in atmfVccGroup to localDown. This is a signal to native ATM services to perform all locally required teardown of the VC. This includes hardware register manipulation and buffer deallocation.

## 4.2. SVC Provisioning

### 4.2.1. Initiating a Call

#### 4.2.1.1. Establishment

The application first acquires a local connection API_endpoint with which primitives may be exchanged that offer native ATM services. This is done via the **ATM_associate_endpoint** primitive. The *endpoint_identifier* that is returned from this primitive is used in the future to identify the newly acquired connection endpoint.

Next, the application must signal its intent to initiate an outgoing call. The application does this via the **ATM_prepare_outgoing_call** primitive. In response to the primitive, the connection API_endpoint creates data structures that initially hold default values for various connection attributes. Examples of connection attributes include the AAL type, forward peak cell rate, and QOS class; for a complete listing see Annex A. The application may examine any of these default connection attributes via the **ATM_query_connection_attributes** primitive. In addition, the application may optionally modify some of these attributes via the **ATM_set_connection_attributes** primitive. Depending on the implementation, each of these attributes may or may not be settable by the application. Also, the value to which the application attempts to set a given connection attribute may be modified by the connection endpoint, due to local resource constraints or the local implementation of Native ATM Services.

The application can optionally select additional data plane services. The default is that no additional data plane service is provided beyond AAL5. The only such additional option supported at this time is the

SSCOP protocol; SSCOP adds reliability to the ATM connection. Option selection is performed via the SSCS connection attribute.

If the application wishes to place a point-to-multipoint call, then the application signals this to the connection API_endpoint by setting the *user plane connection configuration attribute* of the *Broadband Bearer capability Information Element* within the connection attributes to reflect this (refer to Annex A). The default value for the *user plane connection configuration attribute* shall be to select a point-to-point call. When a point-to-multipoint connection is selected, then Native ATM Services will enforce a reverse bandwidth of zero and indicate a leaf identifier of zero in the Q.2931 network messages.

When the application is satisfied that the connection attributes, as represented by the connection endpoint, conform to the application's requirements, then a call is placed across the ATM network. The application initiates this via the **ATM_connect_outgoing_call** primitive. Included as a parameter of this primitive is *destination_SAP*. The destination SAP is the ATM address of the remote ATM device plus the additional information allowing the call to reach the correct target software entity within the remote ATM device.

Native ATM Services places the call across the ATM network to the target ATM device. If a point-to-point call attempt is successful, then Native ATM Services signals the application via an **ATM_P2P_call_active** primitive. If a point-to-multipoint call attempt is successful, then Native ATM Services signals the application via an **ATM_P2MP_call_active** primitive. In the event that the call attempt was unsuccessful, then Native ATM Services signals the application via an **ATM_call_release** primitive. This primitive includes the cause for the lack of success.

#### 4.2.1.2. Adding and Removing Leafs

For point-to-multipoint calls, the application originating the call has the ability to add and remove additional parties of the call. In this section, the originating application is referred to as the "root" of the call; all other participants of the call are referred to as "leafs".

To add a leaf, the root issues an **ATM_add_party** primitive. The parameters for this primitive include the ATM address of the leaf, plus a *leaf_identifier*, which is used to identify the leaf in future primitives that affect that leaf. Note that the application is responsible for generating values of *leaf_identifier*. If the leaf is successfully added to the point-to-multipoint call, then the application is notified of this via the **ATM_add_party_success** primitive. Otherwise, the application is notified that the request to add a leaf failed, via the **ATM_add_party_reject** primitive.

To remove a leaf, the root issues an **ATM_drop_party (request)** primitive. Included as a parameter of this primitive the root's reason for removing the leaf.

If a leaf decides to remove itself from a point-to-multipoint call, then Native ATM Services notifies the application via the **ATM_drop_party (indication)** primitive. Included as a parameter of this primitive is the leaf's reason for removing itself. This primitive would also be used in the case of the ATM network deciding to remove the leaf.

#### 4.2.1.3. Termination

When the application wishes to terminate the connection, then the application invokes the **ATM_call_release (request)** primitive. If the remote ATM device or the ATM network terminates the connection, then the application is notified of this via the **ATM_call_release (indication)** primitive. Data in transit at the time a connection is terminated is lost. It is therefore up to the cooperating applications to determine that all necessary data has successfully passed through the network prior to terminating the connection.

### 4.2.2. Responding to a Call

#### 4.2.2.1. Establishment

The application first acquires a local connection API_endpoint with which primitives may be exchanged that offer Native ATM Services. This is done via the **ATM_associate_endpoint** primitive. The *endpoint_identifier* that is returned from this primitive is used in the future to identify the newly acquired connection endpoint.

Next, the application must signal its intent to respond to an incoming call. The application does this via the **ATM_prepare_incoming_call** primitive. Included as parameters of this primitive are the ATM address of the local ATM device, plus the SAP information allowing call notification to reach the application. During the processing of this primitive, Native ATM Services ensures that no other application is using the same SAP.

The application then issues the **ATM_wait_on_incoming_call** primitive to request that incoming calls be queued and presented to the application. When such an incoming call does arrive, the connection API_endpoint notifies the application via the **ATM_arrival_of_incoming_call** primitive.

The application must make a decision as to whether or not to accept the call. The application may examine the connection attributes of the newly arrived incoming call via the **ATM_query_connection_attributes** primitive. In addition, the application may modify a small number of attributes via the **ATM_set_connection_attributes** primitive.

If the application decides to accept the call, the application invokes the **ATM_accept_incoming_call** primitive. Otherwise, the application rejects the call via the **ATM_reject_incoming_call** primitive.

Even if call has been accepted, the application must wait for the ATM network to award the call. After an incoming point-to-point call has been awarded, the application is notified via the **ATM_P2P_call_active** primitive. After an incoming point-to-multipoint call has been awarded, the application is notified via the **ATM_P2MP_call_active** primitive. If an error occurred during the awarding of the call, then the application is notified via the **ATM_call_release** primitive.

#### 4.2.2.2. Termination

When the application wishes to terminate the connection, then the application invokes the **ATM_call_release (request)** primitive. If the remote ATM device or the ATM network terminates the connection, then the application is notified of this via the **ATM_call_release (indication)** primitive. Data in transit at the time a connection is terminated is lost. It is therefore up to the cooperating applications to determine that all necessary data has successfully passed through the network prior to terminating the connection.

### 4.2.3. Leaf Initiated Join Procedures

A new Pt-MPt connection may be established in response to an LIJ request as shown in Figure 16. The root will respond with a SETUP message. If the root maintains control ("ROOT LIJ") then further LEAF_SETUP_REQUEST messages from other leaves will be responded to by the root as shown in Figure 17. Alternatively, if the Pt-MPt- connection is identified as a Network LIJ in the SETUP message, the network will be responsible for responding to further LEAF_SETUP_REQUEST messages as shown in Figure 18.

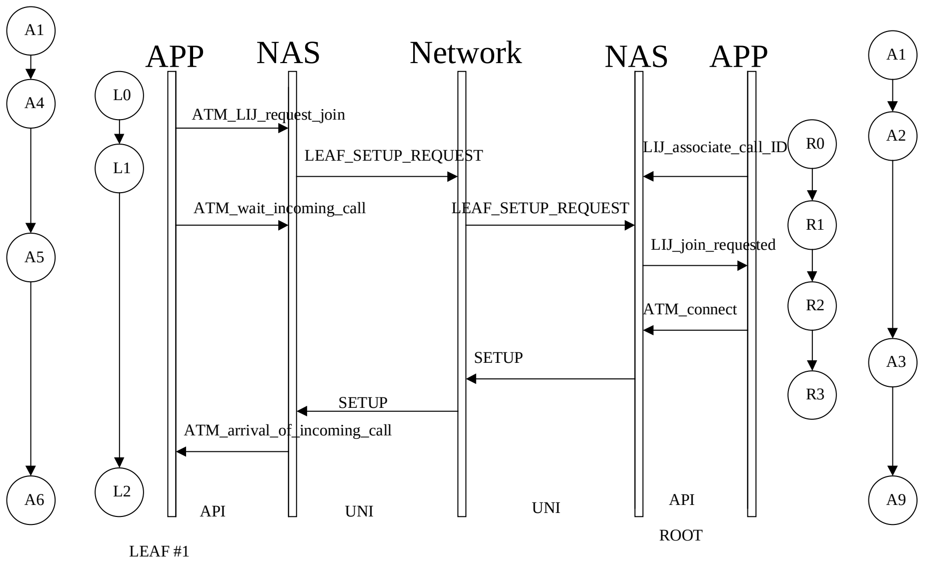

**Figure 16 Example Message Sequence for Pt-MPt LIJ : New Connection Case**

The NAS at the root must be able to associate the LIJ_join_request and the corresponding SETUP, ADD_PARTY, or LEAF_SETUP_FAILURE message. In particular the association is required to be able to provide the leaf_atm_address and leaf_sequence_number parameters *per application API_endpoint* for these messages.

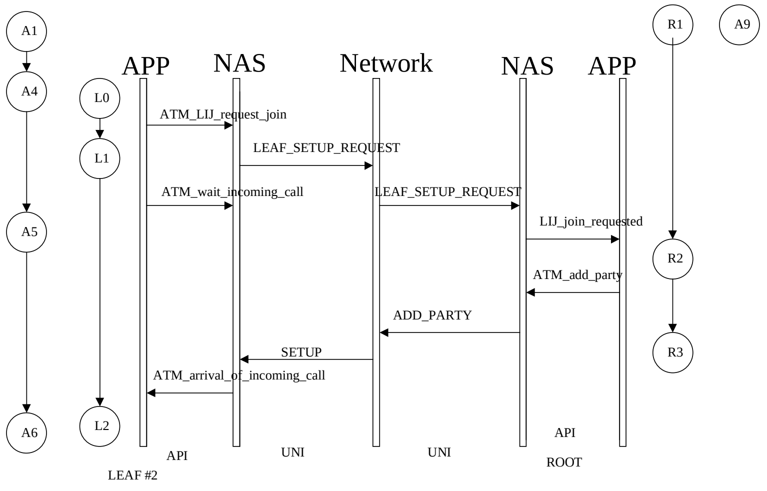

**Figure 17 Example Message Sequence for Root Join LIJ procedure**

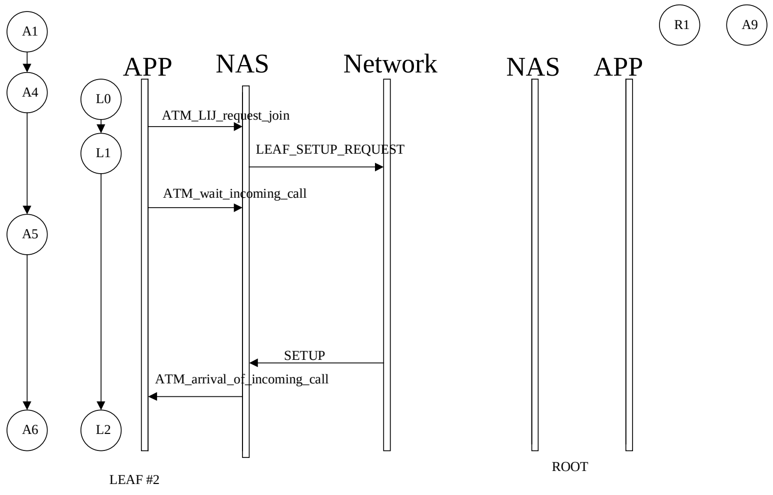

**Figure 18 Example Message Sequence for Pt-MPt LIJ Network LIJ Case of Existing Connection**

In the case of an existing Pt-MPt connection configured for Network LIJ, the Network may originate the LEAF_SETUP_FAILURE message (rather than the root). In Figure 19, the application at the leaf may also receive an ATM_LIJ_leaf_rejected primitive due to the expiry of the local timer (T331 of the UNI 4.0 stack).

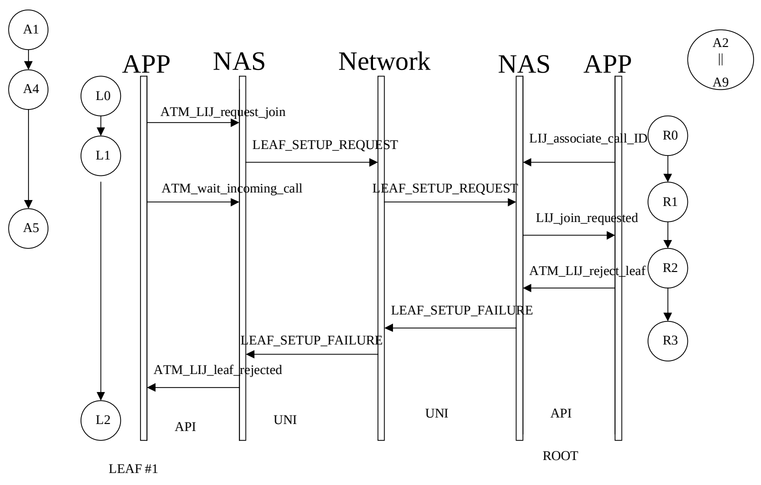

**Figure 19 Example Message Sequence for Call Rejection Case of LIJ**

## 4.3. Synchronization and Coordination Function

This section describes the synchronization and coordination function (SCF) that Native ATM Services performs between the control and data planes. This service is related to and can be selected via the local service connection attribute SSCS that is modified with the *ATM_set_connection_attribute* primitive.

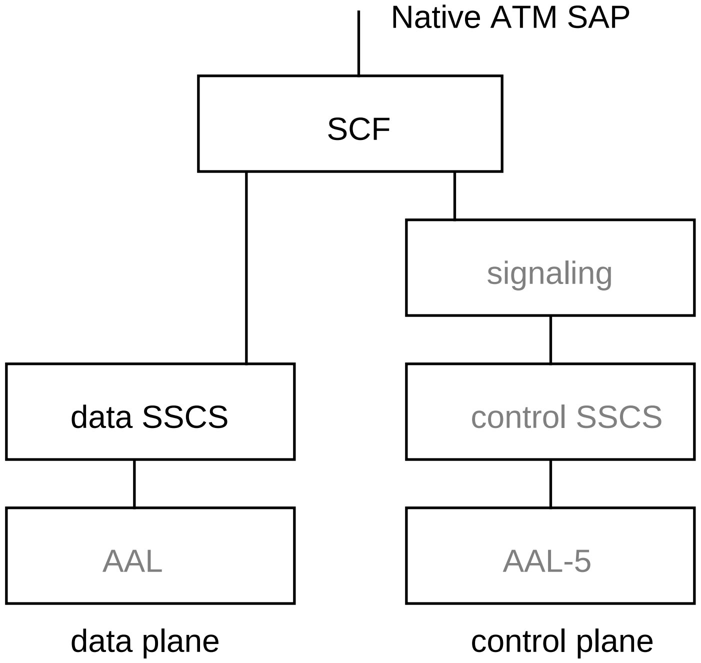

**Figure 20 Synchronization and Co-ordination Function**

### 4.3.1. NULL_SSCS

In this case, the data plane protocol stack for Native ATM Services includes the SAR function and CPCS layer (AAL5, AAL1, or a user-defined AAL). The only function that SCF performs is to maintain the state of API_connection and verify that the **ATM_send_data** and **ATM_receive_data** primitives are issued in the appropriate states.

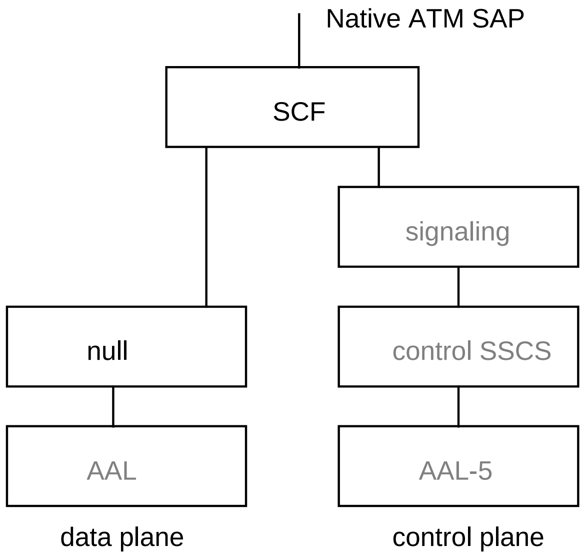

**Figure 21 Native ATM SAP with Null SCCS**

### 4.3.2. SSCOP_RELIABLE_SSCS

In this case, the data plane protocol stack for Native ATM Services includes the SAR function, the CPCS layer (AAL type 5 only), and the SSCOP protocol. The SCF is defined below. Note that this option is only available for point-to-point connections.

"SSCOP" in this discussion is restricted to an instance of the SSCOP protocol that provides reliability in the data plane; it has nothing to do with the instance of SSCOP that provides a similar service for the signaling protocol. Note that the only SSCOP signals used by SCF are: AA-ESTABLISH, AA-RELEASE, AA-DATA, and MAA-ERROR.

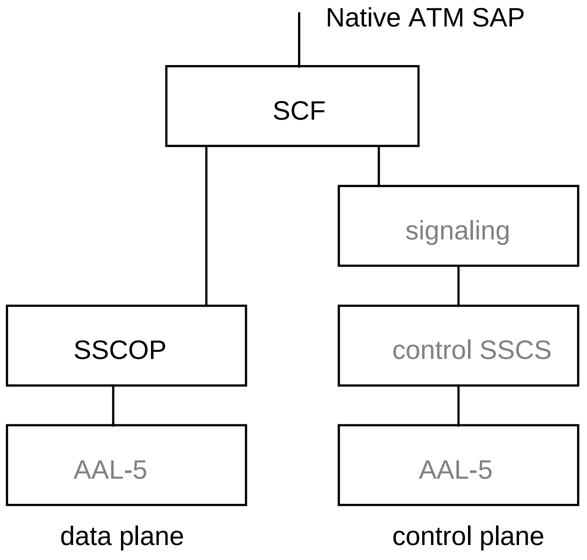

**Figure 22 Native ATM SAP with Reliable SSCS**

#### 4.3.2.1. Connection Establishment

After **ATM_connect_outgoing_call** is invoked, the SCF in the ATM device initiating the ATM call shall establish an SSCOP connection in the following manner:

1. After the Q.2931 CONNECT message is received from the ATM network, the SCF will invoke the AA-ESTABLISH (request) signal to the SSCOP function. Parameter SSCOP-UU shall be null. Parameter BR shall be YES.

2. The SCF shall wait for an AA-ESTABLISH (confirm) signal from the SSCOP function. Parameter SSCOP-UU shall be ignored.
3. Native ATM Services then indicates that the connection is in state A8, via **ATM_P2P_call_active** primitive.

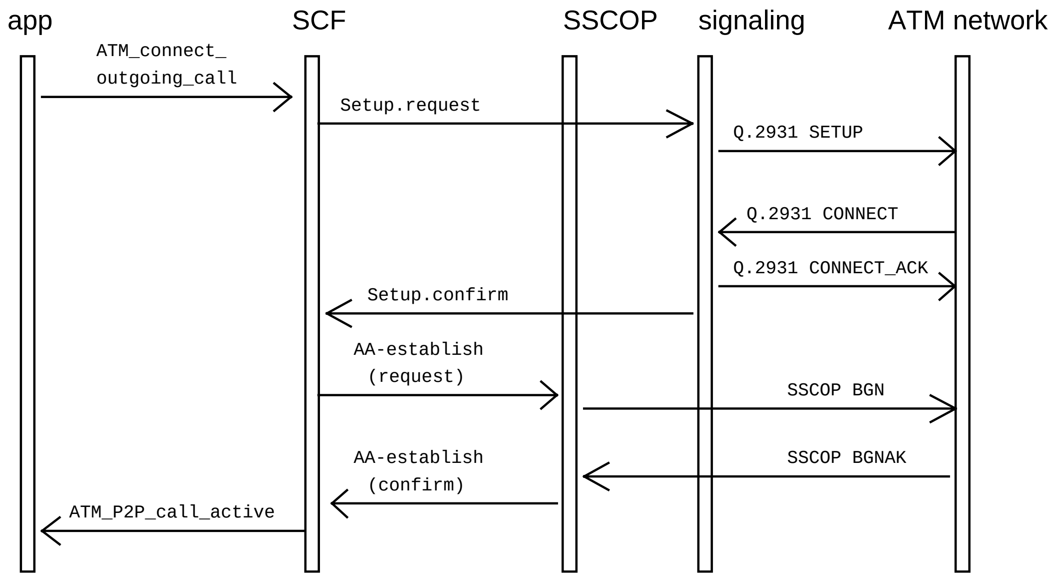

**Figure 23 Example  Message Sequence for Connection Origination**

After **ATM_accept_incoming_call** is invoked, the SCF in the ATM device receiving the ATM call shall establish an SSCOP connection in the following manner:

1. After the Q.2931 CONNECT_ACK message is received from the ATM network, the SCF continues to wait.
2. The SCF shall wait for an AA-ESTABLISH (indication) signal from the SSCOP function. Parameter SSCOP-UU shall be ignored.
3. The SCF will invoke the AA-ESTABLISH (response) signal to the SSCOP function. Parameter SSCOP-UU shall be null.  Parameter BR shall be YES.
4. Native ATM Services then indicates that the connection is in state A8, via **ATM_P2P_call_active** primitive.

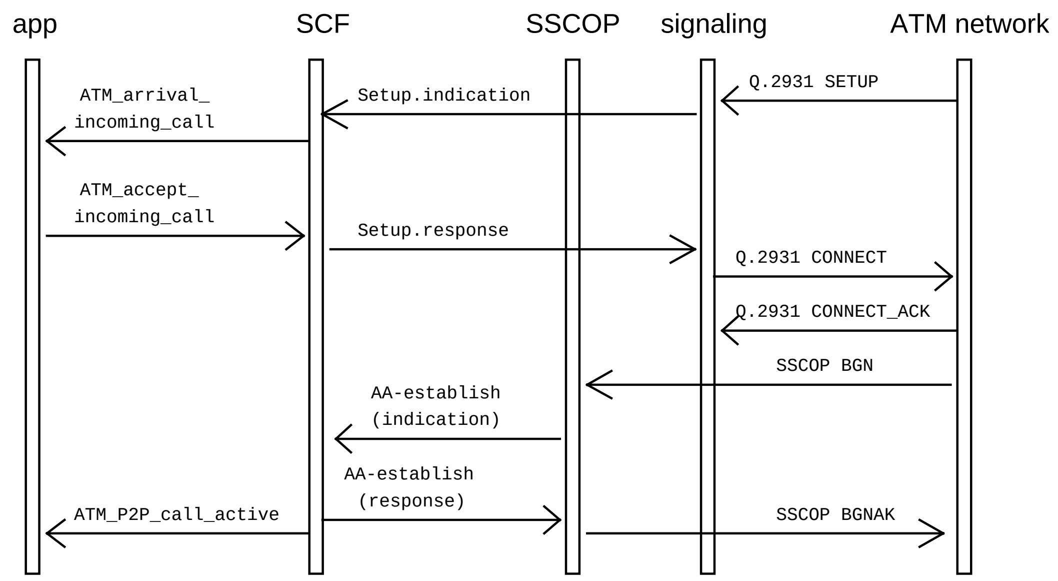

**Figure 24 Example Message Sequence for Connection Establishment**

### 4.3.2.2. Data Transfer

When the **ATM_send_data** primitive is issued, the SCF will invoke an AA-DATA (request) signal to the SSCOP function. When the SSCOP function signals AA-DATA (indication) signal to SCF, then the **ATM_receive_data** primitive is completed. Parameter SN from SSCOP is ignored.

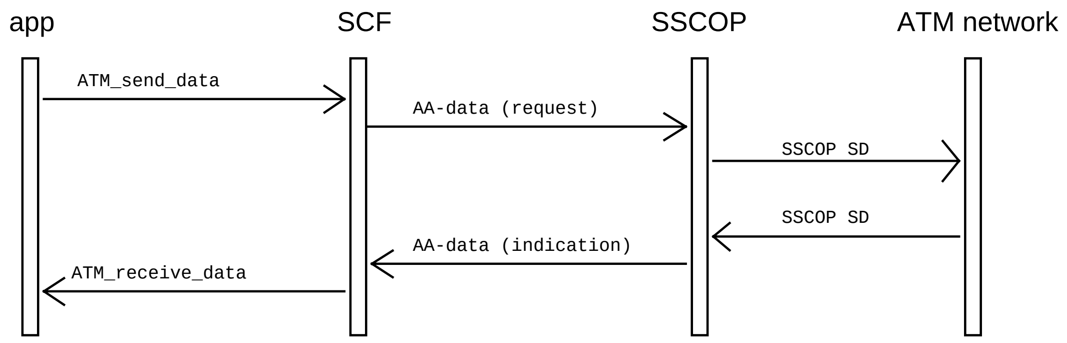

**Figure 25 Example Message Sequence for Data Transfer**

### 4.3.2.3. Connection Termination

If the **ATM_call_release (request)** primitive is issued, the SCF invokes an AA-RELEASE (request) signal to the SSCOP function. Parameter SSCOP-UU shall be null. After this, the ATM device sends a Q.2931 RELEASE message to the ATM network. If the instantiation of SSCOP requires any cleanup, this should be performed after the SCF has requested the Q.2931 RELEASE message to be sent.

Note that depending on a race condition, an SSCOP ENDAK PDU may or may not arrive across the ATM network before the Q.2931 RELEASE_COMPLETE message. This SSCOP ENDAK PDU should be ignored.

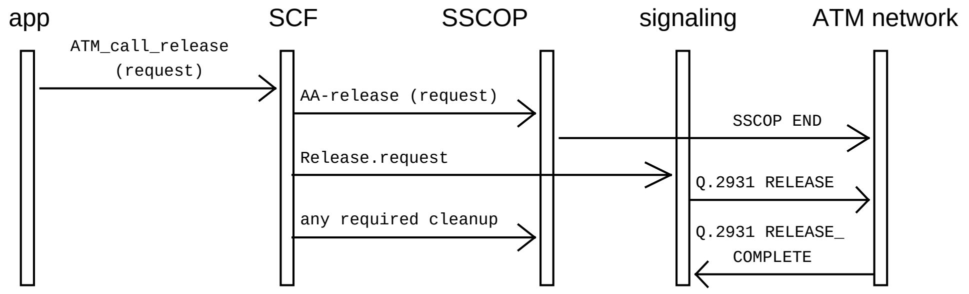

**Figure 26 Example Message Sequence for Connection Termination**

As a result of the previous scenario, the peer SCF on the other end of the ATM connection will receive an AA-RELEASE (indication) signal from the SSCOP function. (The other possibility of the race condition -- a Q.2931 RELEASE being received first -- will be covered below.) When the SCF receives an AA-RELEASE (indication) signal from the SSCOP function, and the SOURCE parameter indicates that the peer user requested the release, then SCF will simply indicate to the application that the connection is no longer available. It is assumed that a Q.2931 RELEASE message will arrive shortly from the ATM network. After the Q.2931 RELEASE message is received, the SCF cleans up any resources from the instantiation of the SSCOP function.

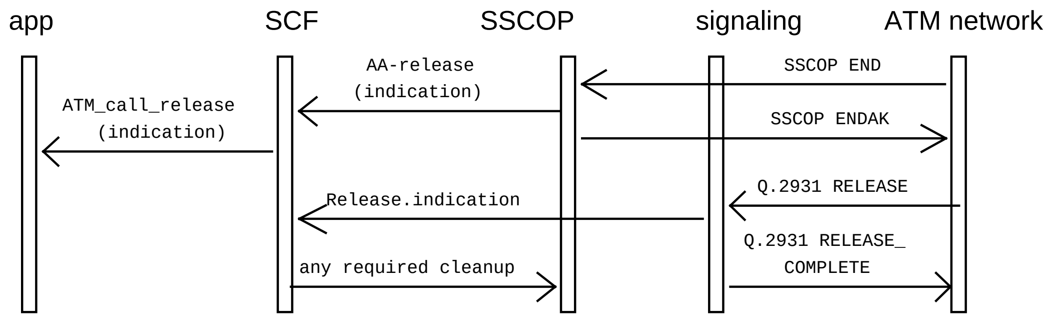

**Figure 27 Example Message Sequence for SSCOP Termination**

The ATM device may receive an unexpected Q.2931 RELEASE message from the ATM network. This could be the result of the race condition presented in the first flow diagram of this section, or else the ATM network terminates the connection. In these cases, SCF issues a **ATM_call_release (indication)** primitive and cleans up any resources from the instantiation of the SSCOP function.

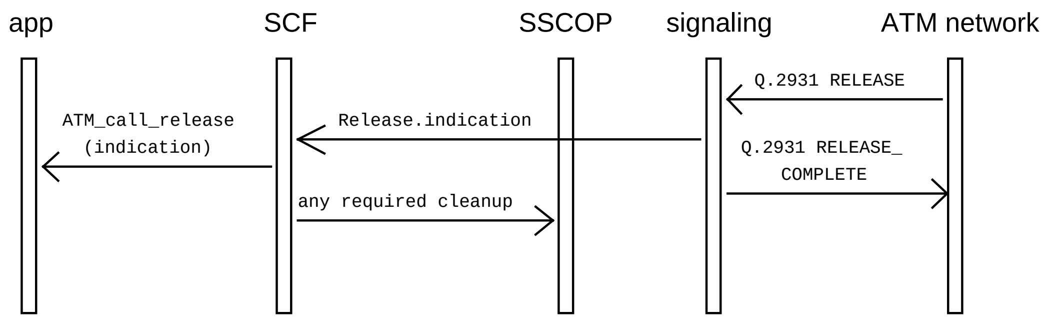

**Figure 28 Example Message Sequence for Q.2931 Release**

If the SCF receives an AA-RELEASE (indication) with the SOURCE parameter indicating the local SSCOP has terminated the connection, or MAA-ERROR signal from the SSCOP function; then SCF issues a **ATM_call_release (indication)** primitive and sends a Q.2931 RELEASE message to the ATM network. If the instantiation of SSCOP requires any cleanup, this should be performed after the SCF has requested the Q.2931 RELEASE message to be sent.

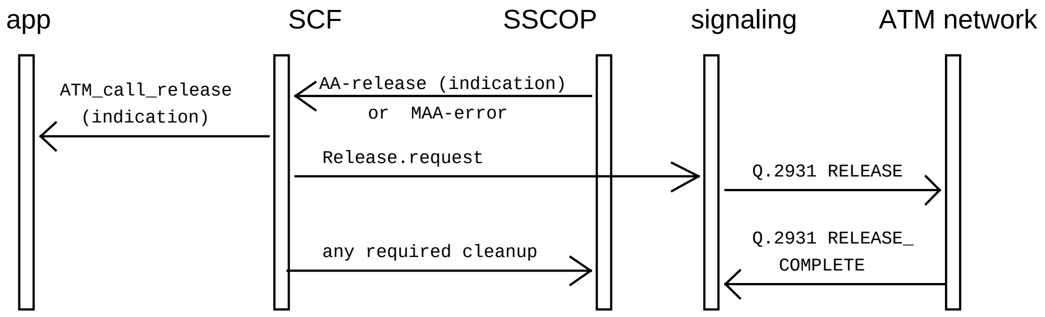

**Figure 29 Example Message Sequence for Q.2931 Release Completion**

### 4.3.3. SSCOP_UNRELABLE_SSCS

The SCF for this SSCS is for further study.

## 4.4. Specification of SAP Address

In this document, a "SAP" (service access point) is used for the following purposes:

1. When the application initiates an outgoing call to a remote ATM device, a *destination_SAP* specifies the ATM address of the remote device, plus further addressing that identifies the target software entity within the remote device.
2. When the application prepares to respond to incoming calls from remote ATM devices, a *local_SAP* specifies the ATM address of the device housing the application, plus further addressing that identifies the application within the local device.

NOTE: The preceding explanation refers to a single destination SAP. This is a simplification; Native ATM Services actually supports up to three destination SAPs. For a discussion of this, see section 4.6.

### 4.4.1. SAP Address as a Vector

The SAP address may be expressed as a vector, (*ATM_addr*, *ATM_selector*, *BLLI_id2*, *BLLI_id3*, *BHLI_id*), where:

- *ATM_addr* corresponds to the 19 most significant octets of a device's 20-octet ATM address (private ATM address structure) or the entire E.164 address (E.164 address structure)
- *ATM_selector* corresponds to the least significant octet of a device's 20-octet ATM address (private ATM address structure only)
- *BLLI_id2* corresponds to a set of octets in the Q.2931 BLLI information element that identify a layer 2 protocol
- *BLLI_id3* corresponds to a set of octets in the Q.2931 BLLI information element that identify a layer 3 protocol
- *BHLI_id* corresponds to a set of octets in the Q.2931 BHLI information element that identify an application (or session layer protocol of an application)

#### 4.4.1.1. SAP Vector Element (SVE)

Each element of the SAP vector is called a SAP Vector Element, or SVE. Each SVE semantically consists of a *SVE_tag*, *SVE_length*, and *SVE_value* field, as defined below. The method used to realize this is implementation specific.

- *SVE_tag* determines the interpretation of the SVE. There are three values defined: PRESENT, ABSENT, and ANY. The value of PRESENT means that the *SVE_value* field contains valid data, which may be compared against corresponding octets found in the Q.2931 SETUP message. The value of ABSENT means that the *SVE_value* field is invalid, and the corresponding octets in the Q.2931 SETUP message are not present. The value of ANY means that the *SVE_value* field is invalid, and that the corresponding octets in the Q.2931 SETUP message may either be not present or assume any value.

The parameter *destination_SAP* in primitive **ATM_connect_outgoing_call** has these special restrictions:

1. An *SVE_tag* of ANY is not allowed for any SVE.
2. An *SVE_tag* of ABSENT is not allowed for the SVE that conveys the ATM address.
3. For private ATM addresses, the *SVE_tag* for the ATM selector byte is PRESENT; for public ATM addresses, the *SVE_tag* for the ATM selector byte is ABSENT.

- *SVE_length* contains the size, in bytes, of the *SVE_value* field. Note that if the value of *SVE_tag* is ABSENT or ANY, then *SVE_length* shall contain a value of zero.
- *SVE_value*, when valid, contains a value to be compared against corresponding octets found in the Q.2931 SETUP message. For example, the *SVE_value* field for the *ATM_selector* SVE may be compared against the Q.2931 Called Party Number information element. As another example, the *SVE_value* field for the *BHLI_id* SVE may be compared against the Q.2931 BHLI information element.

### 4.4.2. SVE Encoding

This section specifies the mapping between the *SVE_value* field and the corresponding Q.2931 information element, for each of the five SVE types. See the preceding section for the encoding of the *SVE_tag* and *SVE_length* fields.

In the mappings listed below, the following conventions are used:

- to label bytes of the *SVE_value* field: the first byte is called “1”, the second byte is called “2”, and so on
- to label octets of the Q.2931 information element: the same label specified in Q.2931 is used.

#### 4.4.2.1. ATM_addr SVE

- Byte 1 of the *SVE_value* field corresponds to octet 5 of the Q.2931 “Called Party Number” information element (“Type of number” and “Addressing/Number plan identification”). Note that this determines whether the ATM address uses private ATM address structure or E.164 address structure.
- When the ATM address uses the private ATM address structure, then bytes 2 - 20 of the *SVE_value* field correspond to octets 6 through the next-to-last octet of the Q.2931 “Called Party Number” information element.
  When the ATM address uses the E.164 address structure, then bytes 2 - N of the *SVE_value* field correspond to octets 6 and beyond of the Q.2931 “Called Party Number” information element.

#### 4.4.2.2. ATM_selector SVE

When the ATM address uses the private ATM address structure, then byte 1 of the *SVE_value* field corresponds to the last octet of the Q.2931 “Called Party Number” information element.

When the ATM address uses the E.164 address structure, then the *SVE_value* field cannot exist; the *SVE_tag* field for the SVE must equal ABSENT or ANY.

#### 4.4.2.3. BLLI_id2 SVE

In this case, there are two options for the structure of the *SVE_value*.

1. Option 1
   - Byte 1 of the *SVE_value* field corresponds to octet 6 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 2 protocol”). Note that for this option, the value of this byte must not correspond to User Specified (b’10000’).
2. Option 2
   - Byte 1 of the *SVE_value* field corresponds to octet 6 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 2 protocol”). Note that for this option, the value of this byte must correspond to User Specified (b’10000’).

- Byte 2 of the *SVE_value* field corresponds to octet 6a of the Q.2931 “Broadband Low Layer Information” information element (“User specified layer 2 protocol information”).

NOTE: This option must be restricted to experimental applications. Uniqueness of these values cannot be guaranteed in a non-experimental ATM environment.

#### 4.4.2.4. BLLI_id3 SVE

In this case, there are five options for the structure of the *SVE_value*.

1. Option 1
   - Byte 1 of the *SVE_value* field corresponds to octet 7 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 3 protocol”). Note that for this option, the value of this byte must not correspond to ISO/IEC TR 9577 (b’01011’) or User Specified (b’10000’).
2. Option 2
   - Byte 1 of the *SVE_value* field corresponds to octet 7 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 3 protocol”). Note that for this option, the value of this byte must correspond to User Specified (b’10000’).
   - Byte 2 of the *SVE_value* field corresponds to octet 7a of the Q.2931 “Broadband Low Layer Information” information element (“User specified layer 3 protocol information”).

   NOTE: This option must be restricted to experimental applications. Uniqueness of these values cannot be guaranteed in a non-experimental ATM environment.
3. Option 3
   - Byte 1 of the *SVE_value* field corresponds to octet 7 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 3 protocol”). Note that for this option, the value of this byte must correspond to ISO/IEC TR 9577 (b’01011’).
   - Byte 2 of the *SVE_value* field corresponds to octets 7a and 7b of the Q.2931 “Broadband Low Layer Information” information element (“Initial Protocol Identifier”). Note that for this option, the value of this byte must not correspond to IEEE 802.1 SNAP (b’10000000’).
4. Option 4
   - Byte 1 of the *SVE_value* field corresponds to octet 7 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 3 protocol”). Note that for this option, the value of this byte must correspond to ISO/IEC TR 9577 (b’01011’).
   - Byte 2 of the *SVE_value* field corresponds to octets 7a and 7b of the Q.2931 “Broadband Low Layer Information” information element (“Initial Protocol Identifier”). Note that for this option, the value of this byte must correspond to IEEE 802.1 SNAP (b’10000000’).
   - Bytes 3 - 7 of the *SVE_value* field correspond to octets 8.1- 8.5 of the Q.2931 “Broadband Low Layer Information” information element (“OUI” and “PID”).
5. Option 5
   - Byte 1 of the *SVE_value* field corresponds to octet 7 of the Q.2931 “Broadband Low Layer Information” information element (“User information layer 3 protocol”). Note that for this option, the value of this byte must correspond to ISO/IEC TR 9577 (b’01011’).
   - Byte 2 of the *SVE_value* field corresponds to octets 7a and 7b of the Q.2931 “Broadband Low Layer Information” information element (“Initial Protocol Identifier”) is not present. Therefore, *SVE_length* equals 1.

   NOTE: This option is an exception case of BLLI_Id3 that does not identify a layer 3 protocol. The semantic meaning of this option is that each frame in the data plane will contain a header identifying the layer 3 protocol for that frame.

   NOTE: This option is equivalent to *SVE_tag* value of ABSENT except for when the vector is used to specify a destination SAP.

#### 4.4.2.5. BHLI_id SVE

In this case, there are three options for the structure of the *SVE_value*.

1. Option 1
   - Byte 1 of the *SVE_value* field corresponds to octet 5 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information Type”). Note that for this option, the value of this byte must correspond to ISO (b’0000000’).

- Bytes 2 - 9 of the *SVE_value* field correspond to octets 6 - 13 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information”).

2. Option 2
   - Byte 1 of the *SVE_value* field corresponds to octet 5 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information Type”). Note that for this option, the value of this byte must correspond to User Specific (b’0000001’).
   - Bytes 2 - 9 of the *SVE_value* field correspond to octets 6 - 13 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information”).

   NOTE: This option must be restricted to experimental applications. Uniqueness of these values cannot be guaranteed in a non-experimental ATM environment.
3. Option 3
   - Byte 1 of the *SVE_value* field corresponds to octet 5 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information Type”). Note that for this option, the value of this byte must correspond to Vendor-Specific (b’0000011’).
   - Bytes 2 - 8 of the *SVE_value* field correspond to octets 6 - 12 of the Q.2931 “Broadband High Layer Information” information element (“High Layer Information”).

## 4.5. Registration of a Local SAP Address

This section specifies the actions taken by Native ATM Services for the purposes of registering a local SAP address by an application through the **ATM_prepare_incoming_call()** primitive.
Native ATM Services must maintain knowledge about all local SAP addresses registered by all applications. When a new SAP is being registered, Native ATM Services must search for a match with all currently registered SAPs and accept the new SAP only when no match is found. If a match is found, Native ATM Services must reject the SAP with a return value of SAP_ALREADY_USED.

### 4.5.1. General SAPs

A match between two SAPs occurs when all the SVEs match according to the rules in the table below. A "registered vector" may be defined as the result of mapping any SAP address currently registered with Native ATM Services into the structure as specified in section 4.4.1. A new vector may be defined as the result of mapping the SAP address which the application is attempting to register into the structure as specified in section 4.4.1.

| **New SVE** | **Registered SVE** | | |
|---|---|---|---|
| | **SVE_tag = ABSENT** | **SVE_tag = PRESENT** | **SVE_tag = ANY** |
| **SVE_tag = ABSENT** | Match | No Match | Match |
| **SVE_tag = PRESENT** | No Match | Match if Values Match [1] | Match |
| **SVE_tag = ANY** | Match | Match | Match |

1. For a more precise specification – the *SVE_value* field of the new vector is compared to the *SVE_value* field of the registered vector; if the values equal each other, then there is a match.

Native ATM Services does not allow multiple overlapping SAPs to be registered at the same time. The use of overlapping SAPs is for further study. However, this does not preclude, wild card SAPs which use the ANY value in SVE_tag field.

### 4.5.2. Special SAPs

#### 4.5.2.1. Narrowband Services SAP

For implementations supporting Narrowband ISDN Services, a special SAP is used provide notification that an incoming call has arrived that uses narrowband bearer services. This SAP address can be registered by only one application on an end system. It is anticipated that the "application" which registers for this SAP address could be a multiplexing service that provides multiple narrowband SAPs to support multiple simultaneous narrowband applications. For any incoming call that uses narrowband bearer services, the Narrowband Services SAP delivers notification of such incoming call to the application registered for the Narrowband Services SAP address.

#### 4.5.2.2. Catch-all SAP

An implementation may optionally allow a special SAP with ANY as the SVE_tag of every SVE. This SAP violates the above rules, but could be allowed as a special case. This SAP can be registered by only one application on an end system. It receives indication of all calls not distributed to other SAPs (see next section for the description of incoming call distribution).

## 4.6. Incoming Call Distribution

### 4.6.1. Incoming Call Distribution

This section specifies actions taken by native ATM services for the purposes of call distribution to the correct entity at the destination ATM device.
An entity shall be identified by a SAP address. A SAP address shall only identify one entity. An entity may be identified by more than one SAP address. A "registered vector" may be defined as the result of mapping the valid address registered by an entity into the structure as specified in section 4.4.1. An entity shall register a SAP address by issuing the **ATM_prepare_incoming_call** primitive.

### 4.6.2. SVE Matching

An "incoming vector" is defined as the result of mapping the incoming call's Q.2931 information elements into the structure as specified in section 4.4.1. Native ATM Services supports up to three incoming vectors for a single incoming call:

- The "first choice" incoming vector's elements are generated from the following Q.2931 information elements: "Called Party Number", "Broadband High Layer Information", and the first occurrence (if any) of "Broadband Low Layer Information".
- The "second choice" incoming vector's elements are generated from the following Q.2931 information elements: "Called Party Number", "Broadband High Layer Information", and the second occurrence of "Broadband Low Layer Information". If only one occurrence of "Broadband Low Layer Information" is present, then there is no second choice incoming vector.
- The "third choice" incoming vector's elements are generated from the following Q.2931 information elements: "Called Party Number", "Broadband High Layer Information", and the third occurrence of "Broadband Low Layer Information". If no more than two occurrences of "Broadband Low Layer Information" is present, then there is no third choice incoming vector.

When an incoming call arrives, the list of registered vectors is searched for a match with the first choice, then second choice, then third choice incoming vectors. A match between a registered vector and one of the incoming vectors occurs when all of the SVEs match according to the rules in the following table. If a registered vector is found that matches one of the incoming vectors, then the entity corresponding to the vector shall be notified of the incoming call via the **ATM_arrival_of_incoming_call** primitive. If a registered vector is not found, the ATM native services rejects the call with a cause value 88 (incompatible destination).

| **Incoming SVE** | **Registered SVE** | | |
|---|---|---|---|
| | **SVE_tag = ABSENT** | **SVE_tag = PRESENT** | **SVE_tag = ANY** |
| **SVE_tag = ABSENT** | Match | No Match | Match |
| **SVE_tag = PRESENT** | No Match | Match if Values Match [1] | Match |

**Notes**
1. For a more precise specification – the *SVE_value* field of the incoming vector is compared to the *SVE_value* field of the registered vector; if the two values are equal, then there is a match.

### 4.6.3. Narrowband Services

When an incoming call is received across the UNI that contains the Narrowband Bearer Capability information element, then that call is considered to use narrowband ISDN services. The notification of such a incoming call shall be provided to the application registered for the Narrowband Services SAP.

## 4.7. Compatibility Checking

The ATM Forum Technical Committee User-Network Interface (UNI) Specification Version 3.1 Annex B, and SIG 4.0 describe the compatibility checking that is required for an incoming connection.
Call distribution uses some of the parameters that must be checked for compatibility. The parameters of an incoming call will be checked against the parameters registered by applications. If there is a match, the connections will be offered to that application via the *ATM_arrival_of_incoming_call* primitive. If there is no match, the call shall be cleared.
The application shall perform compatibility checking, according to the specified procedure in UNI 3.x, while the connection is in state A6. If this procedure determines that the connection is not compatible, the application shall reject the connection using the *ATM_reject_incoming_call* primitive; otherwise it may accept the connection via the *ATM_accept_incoming_call* primitive.

## 4.8. Negotiation of Parameters

The ATM Forum Technical Committee SIG 4.0 and User Network Interface Specification Version 3.1 Annex C describes negotiation for Broadband Low Layer Information (BLLI). Annex F describes negotiation for the ATM Adaptation Layer (AAL). These procedures are supported by Native ATM Services.

As part of the BLLI negotiation, the application initiating the call can specify up to three destination SAPs. These SAPs are composed of the following:

- The first choice destination SAP is completely specified by the *destination_SAP* parameter of the **ATM_connect_outgoing_call** primitive.
- The second choice destination SAP contains the ATM_addr SVE, ATM_selector SVE, and BHLI_id SVE from the *destination_SAP* parameter of the **ATM_connect_outgoing_call** primitive. The BLLI_id2 SVE and BLLI_id3 SVE are obtained from the second instance of the connection attributes located in the “Broadband Low Layer Information” information element.
- The third choice destination SAP contains the ATM_addr SVE, ATM_selector SVE, and BHLI_id SVE from the *destination_SAP* parameter of the **ATM_connect_outgoing_call** primitive. The BLLI_id2 SVE and BLLI_id3 SVE are obtained from the third instance of the connection attributes located in the “Broadband Low Layer Information” information element.

### 4.8.1. Outgoing calls

While the outgoing connection is in state A2, all negotiable parameters may be set using the *ATM_set_connection_attributes* primitive. This includes the setting of up to 3 BLLI choices.
After the connection is in state A8 or A9, the results of negotiation may be obtained via the *ATM_query_connection_attributes* primitive.

### 4.8.2. Incoming calls

While incoming connection is in state A6, the proposed values for negotiable parameters may be read via the *ATM_query_connection_attributes* primitive. If alternate values are required, they are set via the *ATM_set_connection_attributes* primitive. If nothing is set, the proposed values shall be used for the connection. This includes the first proposed value for BLLI.

## 4.9. Procedures for Application Control of ABR Traffic U-Plane Parameters

Examples of the usage of the ABR primitives are shown in Figure 30. These primitives are identified with regard to the direction of flow of U plane data on an ABR connection. Both source and destination of the U-plane data connection can query the value of the ACR parameter using the appropriate primitives (ATM_query_outbound_rate and ATM_query_inbound_rate respectively). Setting a threshold at the source or destination may result in a notification, if that threshold is exceeded in some RM cell received later. If the notification parameters are not provided, the application can query using the appropriate get request. It is also possible for the applications to explicitly knock down the values of the ER set in the RM cells. Using the appropriate set_ER primitives.

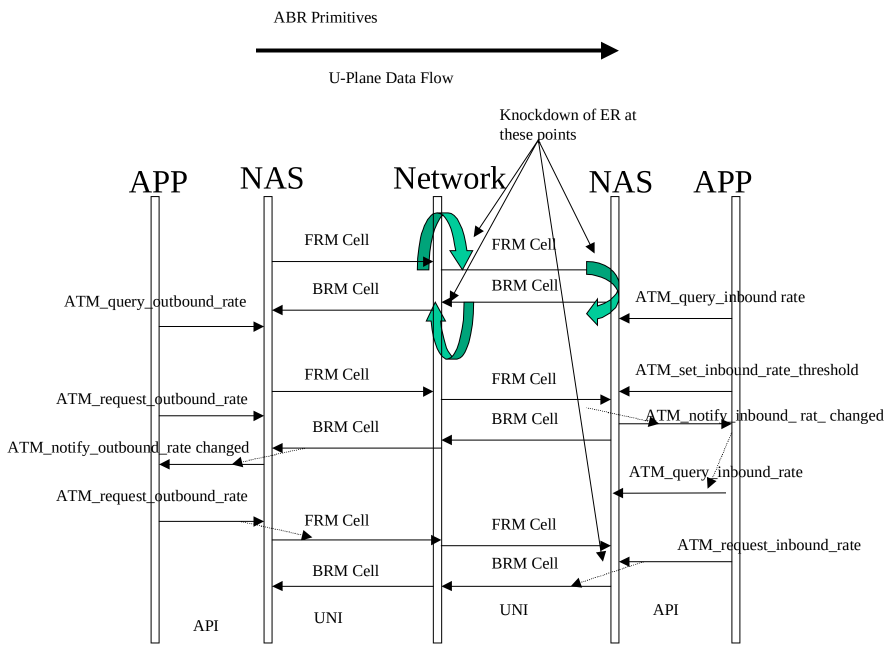
ABR Primitives
U-Plane Data Flow
Knockdown of ER at these points
APP
NAS
Network
NAS
APP
FRM Cell
BRM Cell
ATM_query_outbound_rate
ATM_query_inbound rate
ATM_set_inbound_rate_threshold
ATM_request_outbound_rate
ATM_notify_inbound_ rat_ changed
ATM_notify_outbound_rate changed
ATM_query_inbound_rate
ATM_request_outbound_rate
ATM_request_inbound_rate
API
UNI
UNI
API

**Figure 30 ABR Primitives Examples**

# 5. Reference Documents

The following documents are referred to within this document. They define various aspects of the ATM layers that are being abstracted.

| | | |
|---|---|---|
| I.361 | B-ISDN ATM layer specification | |
| I.362 | B-ISDN ATM Adaptation Layer (AAL) Functional Description | |
| I.363 | B-ISDN ATM Adaptation Layer (AAL) Specification | |
| Q.2100 | B-ISDN Signaling ATM Adaptation Layer Overview Description | |
| Q.2110 | B-ISDN - Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP) | |
| Q.2130 | B-ISDN Signaling ATM Adaptation Layer - Service Specific Coordination Function for Support of Signaling at the User-to-Network Interface (SSCF at UNI) | |
| NAS | Native ATM Service: Semantic description, af-saa-0048.000, February 1996 | |
| ILMI | ATM Forum ILMI 4.0 Specification, af-ilmi-0065.000, September 1996 | |
| VTOA | Voice and Telephony Over ATM to the Desktop, af-vtoa-0083.000, May 1997 | |
| UNI 3.x | ATM User-Network Interface Specification 3.x: | |
| | Version 3.0, | af-uni-0010.001, September 1993 |
| | Version 3.1, | af-uni-0010.002, 1994 |
| UNI 4.0 | ATM Forum Anchorage Accord Specifications related to UNI: | |
| | TM4.0 | af-tm-0056.000, April 1997 |
| | ABR Addendum, | af-tm-0077.000, January 1997 |
| | SIG4.0 | af-sig-0061.000, July 1996 |
| | Signalling ABR Addendum, af-sig-0076.000, January 1997 | |

# Annex A CONNECTION ATTRIBUTES

## Annex A.1 Connection Attributes of UNI 4.0 Signalling

| **Column Headings:** | **Interpretation** |
|---|---|
| UNI 4.0 Information Element | IE's specified in UNI 4.0 ( and UNI 3.x) |
| Information Element Attribute | The attributes of the IEs specified in UNI 4.0 |
| Octet# | The octet number of the attributes within the IE |
| UNI 3.0 | x : the information element / attribute applies in this version of UNI signalling<br>- : the information element / attribute does not apply in this version of UNI signalling |
| UNI 3.1 | x : the information element / attribute applies in this version of UNI signalling<br>- : the information element / attribute does not apply in this version of UNI signalling |
| UNI 4.0 | x : the information element / attribute applies in this version of UNI signalling<br>- : the information element / attribute does not apply in this version of UNI signalling |
| UNI 3.x API Item Name | This is a cross reference to names used in previous version of Semantics specification. There is no syntax implies by these names . The use of these names is deprecated in favor of explicit identification of the signalling information elements and their attributes. |

| UNI 4.0 Information Element | Information Element Attribute | Octet # | State Set | UNI 3.0 | UNI 3.1 | UNI 4.0 | UNI 3.x API ITEM Name |
|---|---|---|---|---|---|---|---|
| Narrowband bearer capability | | | | - | - | x | |
| Cause | Coding Standard | 2 | (note 4) | x | x | x | CAUSE_CODING |
| Cause | Location | 5 | (note 4) | x | x | x | CAUSE_LOCATION |
| Cause | Cause value | 6 | (note 4) | x | x | x | CAUSE_VALUE |
| Cause | Diagnostic | 7 etc. | (note 4) | x | x | x | CAUSE_DIAGNOSTICS |
| Call state | | | | - | - | x | - |
| Progress indicator | | | | - | - | x | - |
| Notification indicator | | | | - | - | x | - |
| End-to-end transit delay | | | | - | - | x | - |
| Connected number | | | | - | - | x | - |
| Connected subaddress | | | | - | - | x | - |
| Endpoint reference | | | | - | - | x | - |
| Endpoint state | | | | - | - | x | - |
| ATM adaptation layer parameters | AAL Type | 5 | A2 | x | x | x | AAL_TYPE |
| ATM adaptation layer parameters | Subtype | 6.1 | A2 | x | x | x | AAL1_SUBTYPE |
| ATM adaptation layer parameters | CBR Rate | 7.1 | A2 | x | x | x | AAL1_CBR_RATE |
| ATM adaptation layer parameters | | 8.1 - 8.2 | A2 | x | x | x | AAL1_MULTIPLIER |
| ATM adaptation layer parameters | Clock Recovery Type | 9.1 | A2 | x | x | x | AAL1_CLOCK_RECOVERY_TYPE |
| ATM adaptation layer parameters | Error Correction Type | 10.1 | A2 | x | x | x | AAL1_ERROR_CORRECTION |
| ATM adaptation layer parameters | Structured Data Transfer | 11.1 - 11.2 | A2 | x | x | x | AAL1_STRUCTURED_DATA_TRANSFER |

| **UNI 4.0 Information Element** | **Information Element Attribute** | **Octet #** | **State Set** | **UNI 3.0** | **UNI 3.1** | **UNI 4.0** | **UNI 3.x API ITEM Name** |
|---|---|---|---|---|---|---|---|
| ATM adaptation layer parameters | Partially Filled Cells | 12.1 | A2 | x | x | x | AAL1_PARTIALLY_FILLED_CELLS |
| ATM adaptation layer parameters | Forward Maximum CPCS-SDU Size | 6.1 - 6.2 | A2+A6 | x | x | x | AAL5_FWD_MAX_SDU |
| ATM adaptation layer parameters | Backward Maximum CPCS-SDU Size | 7.1 - 7.2 | A2+A6 | x | x | x | AAL5_BAK_MAX_SDU |
| ATM adaptation layer parameters | MID Size | 8.1-8.2 | | x | - | - | - |
| ATM adaptation layer parameters | Mode | 9.1 | | x | - | - | - |
| ATM adaptation layer parameters | SSCS Type | 8.1 (note 1) | A2 | x | x | x | AAL5_SSCS_TYPE |
| ATM adaptation layer parameters | User Defined AAL Information | 6 - 6.3 | A2+A6 | x | x | x | USER_DEFINED_AAL_INFO |
| ATM traffic descriptor | Forward Peak Cell Rate (CLP=0) | 5.1 - 5.3 | A2 | x | x | x | FWD_PCR_CLP0 |
| ATM traffic descriptor | Forward Peak Cell Rate (CLP=0+1) | 7.1 - 7.3 | A2 | x | x | x | FWD_PCR_CLP1 |
| ATM traffic descriptor | Backward Peak Cell Rate (CLP=0) | 6.1 - 6.3 | A2 | x | x | x | BAK_PCR_CLP0 |
| ATM traffic descriptor | Backward Peak Cell Rate (CLP=0+1) | 8.1 - 8.3 | A2 | x | x | x | BAK_PCR_CLP1 |
| ATM traffic descriptor | Forward Sustainable Cell Rate (CLP=0) | 9.1 - 9.3 | A2 | x | x | x | FWD_SCR_CLP0 |
| ATM traffic descriptor | Forward Sustainable Cell Rate (CLP=0+1) | 11.1 - 11.3 | A2 | x | x | x | FWD_SCR_CLP1 |
| ATM traffic descriptor | Backward Sustainable Cell Rate (CLP=0) | 10.1 - 10.3 | A2 | x | x | x | BAK_SCR_CLP0 |
| ATM traffic descriptor | Backward Sustainable Cell Rate (CLP=0+1) | 12.1 - 12.3 | A2 | x | x | x | BAK_SCR_CLP1 |
| ATM traffic descriptor | Forward Maximum Burst Size (CLP=0) | 13.1 - 13.3 | A2 | x | x | x | FWD_MBS_CLP0 |
| ATM traffic descriptor | Forward Maximum Burst Size (CLP=0+1) | 15.1 - 15.3 | A2 | x | x | x | FWD_MBS_CLP1 |
| ATM traffic descriptor | Backward Maximum Burst Size (CLP=0) | 14.1 - 14.3 | A2 | x | x | x | BAK_MBS_CLP0 |
| ATM traffic descriptor | Backward Maximum Burst Size (CLP=0+1) | 16.1 - 16.3 | A2 | x | x | x | BAK_MBS_CLP1 |
| ATM traffic descriptor | Best Effort Indicator | 17 | A2 | x | x | x | BEST_EFFORT |
| ATM traffic descriptor | Tagging Forward | 18.1 | A2 | x | x | x | FWD_TAGGING |
| ATM traffic descriptor | Tagging backward | 18.1 | A2 | x | x | x | BAK_TAGGING |
| ATM traffic descriptor | Forward Frame Discard | 17.1 | | - | - | x | - |
| ATM traffic descriptor | Backward Frame Discard | 17.1 | | - | - | x | - |

| UNI 4.0 Information Element | Information Element Attribute | Octet # | State Set | UNI 3.0 | UNI 3.1 | UNI 4.0 | UNI 3.x API ITEM Name |
|---|---|---|---|---|---|---|---|
| ATM traffic descriptor | Forward ABR Minimum Cell Rate | 19.1-19.3 | | - | - | x | - |
| ATM traffic descriptor | Backward ABR Minimum Cell Rate | 20.1-20.3 | | - | - | x | - |
| Connection identifier | | | A2 (note 6) | x | x | x | - |
| Quality of service parameter | QoS Class Forward | 5 | A2 | x | x | x | FWD_QOS_CLASS |
| Quality of service parameter | QoS Class Backward | 6 | A2 | x | x | x | BAK_QOS_CLASS |
| Broadband high layer information | | 5 | | | x | x | APPL_ID_TYPE |
| Broadband high layer information | | 6 - 13 | | | x | x | APPL_ID |
| Broadband bearer capability | Bearer Class | 5 | A2 | x | x | x | BEARER_CLASS |
| Broadband bearer capability | ATM Transfer Capability | 5a | | - | - | x | - |
| Broadband bearer capability | Traffic Type | 5a | A2 | x | x | - | TRAFFIC_TYPE |
| Broadband bearer capability | Timing Requirements | 5a | A2 | x | x | - | TIME_REQ |
| Broadband bearer capability | Susceptibility to clipping | 6 | A2 | x | x | x | CLIPPING_IND |
| Broadband bearer capability | User plane connection configuration | 6 | A2 | x | x | x | CONNECT_CONFIG |
| Broadband low-layer information | - | - | A2+A6 | - | - | - | BLLI_SELECTOR (note 3) |
| Broadband low-layer information | User Information Layer 2 Protocol | 6 | A2 | x | x | x | LAYER_2_ID |
| Broadband low-layer information | Mode | 6a | A2+A6 | x | x | x | LAYER_2_MODE |
| Broadband low-layer information | Window Size (k) | 6b | A2+A6 | x | x | x | LAYER_2_WINDOW_SIZE |
| Broadband low-layer information | User Specified Layer 2 Protocol Information | 6a | A2 | x | x | x | LAYER_2_USER_ID |
| Broadband low-layer information | User Information Layer 3 Protocol | 7 | A2 | x | x | x | LAYER_3_ID |
| Broadband low-layer information | Mode | 7a | A2+A6 | x | x | x | LAYER_3_MODE |
| Broadband low-layer information | Default Packet Size | 7b | A2+A6 | x | x | x | LAYER_3_PACKET_SIZE |
| Broadband low-layer information | Packet Window Size | 7c | A2+A6 | x | x | x | LAYER_3_WINDOW_SIZE |
| Broadband low-layer information | User Specified Layer 3 Protocol Information | 7a | A2 | x | x | x | LAYER_3_USER_ID |

| UNI 4.0 Information Element | Information Element Attribute | Octet # | State Set | UNI 3.0 | UNI 3.1 | UNI 4.0 | UNI 3.x API ITEM Name |
|---|---|---|---|---|---|---|---|
| Broadband low-layer information | ISO/IEC TR 9577 Initial Protocol Identification (IPI) (bits 8-2) | 7a - 7b | A2 | x | x | x | LAYER_3_IPI_ID |
| Broadband low-layer information | OUI | 8.1 - 8.3 | A2 | x | x | x | LAYER_3_OUI_ID |
| Broadband low-layer information | PID | 8.4 - 8.5 | A2 | x | x | x | LAYER_3_PID_ID |
| Broadband low-layer information | Terminal Type | 7a | | - | - | x | - |
| Broadband low-layer information | Forward Multiplexing | 7b | | - | - | x | - |
| Broadband low-layer information | Backward Multiplexing | 7b | | - | - | x | - |
| Broadband locking shift | | | | x | x | x | - |
| Broadband non-locking shift | | | | x | x | x | - |
| Broadband sending complete | | | | x | x | x | - |
| Broadband repeat indicator | | | | x | x | x | - |
| Calling party number | Type of Number | 5 | A2 | x | x | x | CALLING_ADDR_FORMAT(note 2) |
| Calling party number | Addressing/ numbering plan identification | 5 | A2 | x | x | x | CALLING_ADDR_FORMAT(note 2) |
| Calling party number | Address | 6 etc. | A2 | x | x | x | CALLING_ADDR |
| Calling party number | Presentation Indicator | 5a | A2 | x | x | x | PRESENTATION_IND |
| Calling party number | Screening Indicator | 5a | A2 | x | x | x | SCREENING_IND |
| Calling party subaddress | Type of Sub address | 5 | A2 | x | x | x | CALLING_SUBADDR_TYPE |
| Calling party subaddress | Subaddress information | 6 etc. | A2 | x | x | x | CALLING_SUBADDR |
| Called party number | Type of Number | 5 | | x | | x | CALLED_ADDR_FORMAT(note 2) |
| Called party number | Addressing/ numbering plan identification | 5 | | x | | x | CALLED_ADDR_FORMAT (note 2) |
| Called party number | Address | 6 etc. | | x | x | x | CALLED_ADDR |
| Called party subaddress | Type of Sub address | 5 | A2 | x | x | x | CALLED_SUBADDR_TYPE |
| Called party subaddress | Subaddress information | 6 etc. | A2 | x | x | x | CALLED_SUBADDR |
| Transit network selection | Network Identification | 6 etc. | A2 | x | x | x | NETWORK_ID |

| UNI 4.0 Information Element | Information Element Attribute | Octet # | State Set | UNI 3.0 | UNI 3.1 | UNI 4.0 | UNI 3.x API ITEM Name |
|---|---|---|---|---|---|---|---|
| Restart indicator | | | | | | x | - |
| Narrowband low layer compatibility | | | | - | - | x | - |
| Narrowband high layer compatibility | | | | - | - | x | - |
| Generic identifier transport | | | | - | - | x | - |
| Minimum acceptable traffic descriptor | | | | - | - | x | - |
| Alternative ATM traffic descriptor | | | | - | - | x | - |
| ABR setup parameters | Forward ABR Initial Cell Rate Identifier (CLP = 0+1) | 5 | (note 5) | - | - | x | - |
| ABR setup parameters | Forward ABR Initial Cell Rate | 5.1-5.3 | (note 5) | - | - | x | - |
| ABR setup parameters | Backward ABR Initial Cell Rate | 6.1-6.3 | (note 5) | - | - | x | - |
| ABR setup parameters | Forward ABR Transient Buffer Exposure | 7.1-7.3 | (note 5) | - | - | x | - |
| ABR setup parameters | Backward ABR Transient Buffer Exposure | 8.1-8.3 | (note 5) | - | - | x | - |
| ABR setup parameters | Cumulative RM Fixed Round Trip Time | 9.1-9.3 | (note 5) | - | - | x | - |
| ABR setup parameters | Forward Rate Increase Factor | 10.1 | (note 5) | - | - | x | - |
| ABR setup parameters | Backward Rate Increase Factor | 11.1 | (note 5) | - | - | x | - |
| ABR setup parameters | Forward Rate Decrease Factor | 12.1 | (note 5) | - | - | x | - |
| ABR setup parameters | Backward Rate Decrease Factor | 13.1 | (note 5) | - | - | x | - |
| Leaf initiated join call identifier | LIJ Call Identifier Type | 5 | | - | - | x | - |
| Leaf initiated join call identifier | Identifier Value | 6-9 | | - | - | x | - |

| **UNI 4.0 Information Element** | **Information Element Attribute** | **Octet #** | **State Set** | **UNI 3.0** | **UNI 3.1** | **UNI 4.0** | **UNI 3.x API ITEM Name** |
|---|---|---|---|---|---|---|---|
| Leaf initiated join parameters | Screening Indication | 5 | | - | - | x | - |
| Leaf sequence number | Leaf sequence number | 5-8 | | - | - | x | - |
| Connection scope selection | Connection scope selection | 6 | | - | - | x | - |
| ABR additional parameters | Forward Additional Parameters Record | 5.1-5.4 | (note 5) | - | - | x | - |
| ABR additional parameters | Backward Additional Parameters Record | 6.1-6.4 | (note 5) | - | - | x | - |
| Extended QoS parameters | Origin | 5 | | - | - | x | - |
| Extended QoS parameters | Acceptable Forward Peak-to-Peak Cell Delay Variation | 6.1-6.3 | | - | - | x | - |
| Extended QoS parameters | Acceptable Backward Peak-to-Peak Cell Delay Variation | 7.1-7.3 | | - | - | x | - |
| Extended QoS parameters | Cumulative Forward Peak-to-Peak Cell Delay Variation | 8.1-8.3 | | - | - | x | - |
| Extended QoS parameters | Cumulative Backward Peak-to-Peak Cell Delay Variation | 9.1-9.3 | | - | - | x | - |
| Extended QoS parameters | Acceptable Forward Cell Loss Ratio | 10.1 | | - | - | x | - |
| Extended QoS parameters | Acceptable Backward Cell Loss Ratio | 11.1 | | - | - | x | - |

Notes:

1. This connection attribute is encoded at octet 10.1 for AAL 3 /4 and octet 9.1 for AAL5 in UNI 3.0.
2. The UNI 3.0 API specification merged these two information element attributes into one data item.
3. This connection attribute does not appear in a Q.2931 message; it is used to select the choice of BLLI that other primitives operate on. See primitives **ATM_query_connection_attributes**, **ATM_set_connection_attributes**, **ATM_arrival_of_incoming_call**, and **ATM_accept_incoming_call** for the semantics of this attribute.
4. This connection attribute is set via the ATM_abort_connection, ATM_call_release, and ATM_reject_incoming_call primitives.
5. Refer to the TM4.0 Specification
6. This connection attribute may only be set for UNI 4.0.

## Annex A.2 Connection Attributes of Local Services

| Item | Set in State | | Values | Notes |
|---|---|---|---|---|
| | A2 | A6 | | |
| SSCS (Service Specific Convergence Sublayer) | X | X | NULL,<br>SSCOP_RELIABLE,<br>SSCOP_UNRELIABLE | 1 |

Notes:

1. This selects the services that native ATM services supplies in the data plane for AAL5.

# Annex B Management Variables

## Annex B.1 UNI Defined

The semantic meaning and codings of these management variables may be found in ATM User Network Specification (3.x) Chapter 4, and in the ILMI 4.0 specification, which introduced the notation:

- R required
- D deprecated, use only for backward compatibility

| MIB Group and Variable | 3.0 | 3.1 | 4.0 | Table Entry |
|---|---|---|---|---|
| **atmfPhysicalGroup** | | | | |
| atmfPortIndex | X | X | R | X |
| atmfPortTransmissionType | X | X | D | X |
| atmfPortMediaType | X | X | D | X |
| atmfPortOperStatus | X | X | D | X |
| atmfPortSpecific | X | X | D | X |
| atmfPortMyIfName | | X | R | X |
| atmfPortMyIfIdentifier | | | R | |
| atmfMySystemIdentifier | | | R | |
| atmfMyIpNmAddress | | X | R | |
| atmfMyOsiNmNsapAddress | | X | R | |
| | | | | |
| **atmfAtmLayerGroup** | | | | |
| atmfAtmLayerIndex | X | X | R | X |
| atmfAtmLayerMaxVPCs | X | X | R | X |
| atmfAtmLayerMaxVCCs | X | X | R | X |
| atmfAtmLayerConfiguredVPCs | X | X | R | X |
| atmfAtmLayerConfiguredVCCs | X | X | R | X |
| atmfAtmLayerMaxVpiBits | X | X | R | X |
| atmfAtmLayerMaxVciBits | X | X | R | X |
| atmfVccTable | | | R | |
| atmfAtmLayerMaxSvpcVpi | | | R | |
| atmfAtmLayerMaxSvccVpi | | | R | |
| atmfAtmLayerMinSvccVci | | | R | |
| atmfAtmLayerDeviceType | | | R | |
| atmfAtmLayerIlmiVersion | | | R | |
| atmfAtmLayerUniType | X | X | R | X |
| atmfAtmLayerUniVersion | | X | R | X |
| | | | | |
| **atmfAtmStatsGroup** | | | D | |
| atmfAtmStatsIndex | X | X | D | X |
| atmfAtmStatsReceivedCells | X | X | D | X |
| atmfAtmStatsDroppedReceivedCells | X | X | D | X |
| atmfAtmStatsTransmittedCells | X | X | D | X |
| | | | | |
| **atmfVccGroup** | | | | |
| atmfVccPortIndex | X | X | R | X |
| atmfAtmLayerConfiguredVCCs | | | R | |
| atmfVccBestEffortIndicator | | | R | |
| atmfVccTransmitFrameDiscard | | | R | |

| MIB Group and Variable | 3.0 | 3.1 | 4.0 | Table Entry |
|---|---|---|---|---|
| atmfVccReceiveFrameDiscard | | | R | |
| atmfVpcServiceCategory | | | R | |
| atmfVccVpi | X | X | R | X |
| atmfVccVci | X | X | R | X |
| atmfVccOperStatus | X | X | R | X |
| atmfVccTransmitTrafficDescriptorType | X | X | | X |
| atmfVccTransmitTrafficDescriptorParam1 | X | X | | X |
| atmfVccTransmitTrafficDescriptorParam2 | X | X | | X |
| atmfVccTransmitTrafficDescriptorParam3 | X | X | | X |
| atmfVccTransmitTrafficDescriptorParam4 | X | X | | X |
| atmfVccTransmitTrafficDescriptorParam5 | X | X | | X |
| atmfVccReceiveTrafficDescriptorType | X | X | | X |
| atmfVccReceiveTrafficDescriptorParam1 | X | X | | X |
| atmfVccReceiveTrafficDescriptorParam2 | X | X | | X |
| atmfVccReceiveTrafficDescriptorParam3 | X | X | | X |
| atmfVccReceiveTrafficDescriptorParam4 | X | X | | X |
| atmfVccReceiveTrafficDescriptorParam5 | X | X | | X |
| atmfVccTransmitQoSClass | X | X | D | X |
| atmVccReceiveQoSClass | X | X | D | X |
| | | | | |
| **atmfVccAbrGroup** | | | | |
| atmfVccAbrPortIndex | | | R | |
| AtmfVccAbrVpi | | | R | |
| atmfVccAbrVci | | | R | |
| atmfVccAbrTable | | | R | |
| | | | | |

## Annex B.2 Native ATM Services Defined

| | |
|---|---|
| atmfVccEndpointIdentifier | This element is an extension of atmfVccEntry defined in the atmfVccGroup of UNI 3.0 Chapter 4. Its purpose is to correlate a VCC with an *API_endpoint*.<br><br>**NOTE:** this is a table entry and requires an index parameter. |

# Annex C Usage of Wildcards in SAPs

This annex describes the care that must be taken when using “wildcards” to specify SAPs. A “wildcard” is defined as an SVE (SAP vector element) with a tag of ANY. The use of wildcards in a SAP offers flexibility to an application developer. However, this feature shall optionally be used only after the following implications are understood:

- The *destination_SAP* parameter of the **ATM_connect_outgoing_call** primitive may not contain any wildcards.
- When an application successfully registers a SAP with a wildcard, the SAP occupies a large number of the available SAP combinations, based on the matching rules in section 4.5. Thus, careless use of wildcards may preclude other desired applications from registering their SAPs.
- When an application registers a SAP containing a wildcard, the application implies that it supports all specific values for the matching SVE of an incoming vector (including the ABSENT tag) on an incoming call.
- Once an application has accepted an incoming call on an API_endpoint associated with a wildcard SAP, the application must support and employ the protocols specified by the incoming vector that matched the applicationís registered SAP.

# Annex D API Conformance Statement

The following tables detail compatibility of an implementation of Native ATM Services with this specification. Various implementations of the NAS may use different programming languages and operating system services. These will result in the use of a concrete syntax instantiation of the NAS primitives at the API. In general, this API conformance is intended to support software based implementations of Native ATM Services, however, this intent should not preclude hardware implementations of some primitives.

## Annex D.1 Native ATM Services Support

The set of services supported by NAS is implementation-specific.

| **Service Index** | **Native ATM Service** | **Notes** | **Does the Implementation support this Service?** |
|---|---|---|---|
| 1 | PVC Connection | | |
| 2 | Pt-Pt SVC Connection | | |
| 3 | Pt-Mpt SVC Connection | 1 | |
| 4 | leaf-initiated-join | 2 | |
| 5 | data reception via polling | 3 | |
| 6 | data reception via blocking | 3 | |
| 7 | data reception via messaging | 3 | |
| 8 | data transmission service | | |
| 9 | ABR flow control management | | |
| 10 | ABR flow control indications | 4 | |
| 11 | OAM loopback testing | | |
| 12 | OAM fault alerts | | |
| 13 | ILMI MIB access | | |
| 14 | SSCOP in user plane | 1, 5 | |
| 15 | Narrowband Services access via “narrowband services SAP” | 1, 6 | |
| 16 | default incoming call notification via “catch-all SAP” | 1, 6 | |
| 17 | BLLI negotiation | 1, 7 | |

**Table 1 Service Support**

Notes
1. This service requires Pt-Pt SVC connection service.
2. This service requires Pt-Mpt SVC connection service.
3. At least one data reception service (polling, blocking, messaging) is required in order to receive data.
4. This service requires ABR flow control management service.
5. This service is specified in section 4.3.2.
6. This service is specified in section 4.5.2.
7. This service is specified in section 4.8 and section 3.1 (refer to usage of the BLLI_SELECTOR connection attribute in the **ATM_query_connection_attributes, ATM_set_connection_attributes, ATM_arrival_of_incoming_call, ATM_accept_incoming_call** primitives).

## Annex D.2 Compatibility of Control Plane Functional Primitives

The services for which the primitive is mandatory are indicated by the service index.

| Native ATM Services Primitive | Type | Service Index | Concrete Syntax Primitives | Notes |
|---|---|---|---|---|
| **ATM_abort_connection** | Request | 2 | | |
| **ATM_accept_incoming_call** | Request | 2 | | |
| **ATM_add_party** | Request | 3 | | |

| | | | | |
|---|---|---|---|---|
| **ATM_add_party_reject** | Confirm | 3 | | |
| **ATM_add_party_success** | Confirm | 3 | | |
| **ATM_arrival_of_incoming _call** | Indication | 2 | | |
| **ATM_associate_endpoint** | Request | 2 | | |
| **ATM_call_release** | Request | 2 | | |
| **ATM_call_release** | Confirm | 2 | | |
| **ATM_connect_outgoing_call** | Request | 2 | | |
| **ATM_drop_party** | Request | 3 | | |
| **ATM_drop_party** | Indication | 3 | | |
| **ATM_get_local_port _info** | Request | 2 | | |
| **ATM_P2MP_call_active** | Confirm | 3 | | |
| **ATM_P2P_call_active** | Confirm | 2 | | |
| **ATM_prepare_incoming_call** | Request | 2 | | |
| **ATM_prepare_outgoing_call** | Request | 2 | | |
| **ATM_query_connection_ attributes** | Request | 2 | | |
| **ATM_reject_incoming_call** | Response | 2 | | |
| **ATM_set_connection_ attributes** | Request | 2 | | |
| **ATM_wait_on_incoming_call** | Request | 2 | | |
| **ATM_LIJ_associate_call_ID** | Request | 4 | | |
| **ATM_LIJ_reqest_join** | Request | 4 | | |
| **ATM_LIJ_join_requested** | Indication | 4 | | |
| **ATM_LIJ_reject_leaf** | Response | 4 | | |
| **ATM_LIJ_leaf_rejected** | Confirm | 4 | | |

**Table 2 Control Plane Primitives Support**

## Annex D.3 Compatibility of Data Plane Functional Primitives

The services for which the primitive is mandatory are indicated by the service index.

| Native ATM Services Primitive | Type | Service Index | Concrete Syntax Primitives | Notes |
|---|---|---|---|---|
| **ATM_send_data** | Request | 8 | | |
| **ATM_receive_data** (polling) | Request | 5 | | |
| **ATM_receive_data** (blocking) | Request | 6 | | |
| **ATM_receive_data** (messaging) | Request | 7 | | |
| **ATM_query_outbound_rate** | Request | 9 | | |
| **ATM_set_outbound_notification _threshold** | Request | 10 | | |
| **ATM_outbound_rate_changed** | Indication | 10 | | |
| **ATM_request_outbound_rate** | Request | 9 | | |
| **ATM_query_inbound_rate** | Request | 9 | | |
| **ATM_set_inbound_notification_t hreshold** | Request | 10 | | |
| **ATM_inbound_rate_changed** | Indication | 10 | | |
| **ATM_request_inbound_rate** | Request | 9 | | |

**Table 3 Data Plane Primitives Support**

## Annex D.4 Compatibility of Management Plane Functional Primitives

The services for which the primitive is mandatory are indicated by the service index.

| Native ATM Services Primitive | Type | Service Index | Concrete Syntax Primitives | Notes |
|---|---|---|---|---|
| **ATM_confirm_loopback** | Confirm | 11 | | |
| **ATM_indicate_error** | Indication | | | |
| **ATM_indicate_fault** | Indication | 12 | | |
| **ATM_initiate_loopback** | Request | 11 | | |
| **ATM_query_mgmt_variable** | Request | 1, 13 | | |
| **ATM_set_mgmt_variable** | Request | 1, 13 | | |

**Table 4 Management Plane Primitives Support**

## Annex D.5 State Compatibility

While the implementation using these specific state variables is not required, in many implementations, it is expected that the state may be observable via some state management variable. The tables in this section provide a mechanism to identify such state variables within the concrete syntax implementation of the NAS

| Native ATM Services State | Concrete Syntax Equivalent | Notes /Examples |
|---|---|---|
| **Null (A0)** | | |
| **Initial (A1)** | | |
| **Outgoing Call Preparation (A2)** | | |
| **Outgoing Call Requested (A3)** | | |
| **Incoming Call Preparation (A4)** | | |
| **Wait Incoming Call (A5)** | | |
| **Incoming Call Present (A6)** | | |
| **Incoming Call Requested (A7)** | | |
| **Point-to-Point Data Transfer (A8)** | | |
| **Point-to-Multipoint Root Data Transfer (A9)** | | |
| **Point-to-Multipoint Leaf Data Transfer (A10)** | | |
| **Connection Terminated (A11)** | | |

**Table 5 API_connection State Machine**

| Native ATM Services State | Concrete Syntax Equivalent | Notes /Examples |
|---|---|---|
| **LIJ Leaf Null (L0)** | | |
| **LIJ Leaf Initialized (L1)** | | |
| **LIJ Leaf Closed (L2)** | | |
| **LIJ Root Null (R0)** | | |
| **LIJ Root Initialized (R1)** | | |
| **LIJ Root (R2)** | | |
| **LIJ Root Closed(R3)** | | |

**Table 6 LIJ State Machines**

## Appendix A Pragmatics of Data Send and Receive [INFORMATIVE]

Fundamental issues in sending data are:

1. identifying the location for outgoing data,
2. determining when the source location(s) may be reused, and
3. recognizing/handling sender overflow.

The most common way to specify the data source is to provide the location of a buffer (e.g. via a pointer) and amount of data to send starting from that location. Alternatively, the source data may be specified as an immediate operand to the send operation, i.e., the data is contained on the stack or in a register.

The source buffer cannot be reused until either the data has been deemed sent (or the attempt aborted) or the data has been copied to another buffer. Thus returning from a send operation does not necessarily mean the data has actually been sent. Depending on the communication protocol, (in the case of AAL type 5, depending on assured vs. unassured modes), the data might not be deemed sent until it is known to be successfully received at the destination. Typically, a send operation blocks until the source data is copied or mapped to an intermediate buffer (e.g. in the operating system/network driver). An implementation could also block until the data is deemed sent, but since this incurs latency waiting for the network this usually has poor performance. Yet other possibilities are to poll until the reuse is indicated or to asynchronously interrupt/notify/callback the application to permit buffer reuse.

Sender overflow may occur if the network is not able to send data as fast as an application(s) can transfer data via send operations. In particular, intermediate buffers in the operating system/network driver may overflow. This is an important issue for UBR (unspecified bit rate) traffic: the network capacity devoted to such a connection can change at any time. Thus no amount of sender rate preplanning or buffer size can prevent overflow. Furthermore, the ABR connection capacity can change rapidly. Consequently, implementations may provide a feedback mechanism for indicating and controlling sender overflow. (Of course, a particular implementation is free to simply drop data while an overflow condition exists, but this is not a satisfactory solution for all applications.) Fundamental ways to accomplish this feedback are by blocking (e.g. on a send call), polling (either explicitly or based on some value returned by a send call), or interrupt/notification/callback to the application. It is often effective to tie the source generation rate to the availability of buffers for reuse.

Fundamental issues in receiving data are:

1. identifying the location for depositing incoming data,
2. indicating arrival of data to the application, and
3. recognizing/handling receiver overflow.

Typically, the operating system/network driver chooses a temporary intermediate location in operating system memory for the incoming data. However, it is also possible for the application to specify a location for the incoming data. For example, the application can indicate a memory range that can be transferred to the operating system/network driver as a buffer.

The application can determine the arrival of data via polling or blocking until the data arrives. Alternatively, the application can be informed of data arrival via an interrupt/notification/callback. In polling or blocking, once data has arrived it is transferred via copying or mapping to a location pre-specified by the application in the receive operation. In the case of interrupt/notification/callback, the application can provide a location at the time of such an event to which the data it is transferred via copying or mapping.

Receiver overflow may occur if the application is not able to retrieve (or use data) as fast as the network receives it. As with sender overflow, intermediate buffers in the operating system/network driver may overflow. This is an important issue for UBR (unspecified bit rate) traffic: the network could deliver data at any point; moreover, the application may not be scheduled. Unless the application is guaranteed to attempt to receive data at regular intervals (e.g. via a timer) and the connection PCR (peak cell rate) is set appropriately, it is possible that receiver overflow occurs. Receiver overflow is also an issue for CBR (constant bit rate) connections, unless the application can be guaranteed to be scheduled regularly and consume data. Since the application may not be scheduled and thus not able to consume and thereby replenish the intermediate buffers, there should be an interrupt/notification/callback mechanism for receiver overflow feedback.

For guaranteed connection rates (e.g. CBR), underflow may also be an issue. Sender underflow arises when there is insufficient data available for the network to send. For CBR connections, sender underflow may result in a violation of the guaranteed rate to the destination application. Likewise, receiver underflow arises when there may not be sufficient data available for the application.

Finally, it is very common to use operating system/network driver memory as a intermediate buffer for sending and receiving. Various work has explored eliminating the overhead of this approach by obtaining the source data directly from application memory and storing incoming network data directly into application memory.

## Appendix B Mapping between Native ATM Service primitives and ATM Forum UNI 3.x/4.0 Signalling messages [INFORMATIVE]

This annex presents the mapping between the UNI 3.x/4.0 messages provided by the signaling and the Native ATM Services primitives.

### Appendix B.1 Primitives invoked by the application

The left column of the following table contains the Native ATM Services primitives and in the right column the resulting signaling messages. The Native ATM Services primitives refer to request or response primitives.

| Native ATM Services primitives | ATM Forum signaling messages |
|---|---|
| **ATM_abort_connection** | RELEASE |
| **ATM_accept_incoming_call** | CONNECT |
| **ATM_add_party** | ADD PARTY |
| **ATM_associate_endpoint** | None (local significance) |
| **ATM_call_release** | RELEASE |
| **ATM_connect_outgoing_call** | SETUP |
| **ATM_drop_party** | DROP PARTY |
| **ATM_prepare_incoming_call** | None (local significance) |
| **ATM_prepare_outgoing_call** | None (local significance) |
| **ATM_query_connection_attributes** | None (local significance) |
| **ATM_reject_incoming_call** | RELEASE |
| **ATM_set_connection_attributes** | None (local significance) |
| **ATM_wait_on_incoming_call** | None (local significance) |
| **ATM_LIJ_associate_call_ID** | N/A- local action |
| **ATM_LIJ_request_join** | LEAF_SETUP_REQUEST |
| **ATM_LIJ_reject_leaf** | LEAF_SETUP_FAILURE |
| **ATM_query_outbound_rate** | N/A- local action |
| **ATM_set_outbound_notification_threshold** | N/A- local action |
| **ATM_request_outbound_rate** | N/A- affects FRM cell |
| **ATM_query_inbound_rate** | N/A- local action |
| **ATM_set_inbound_notification_threshold** | N/A- local action |
| **ATM_request_inbound_rate** | N/A- affects BRM cell |
| | |

Messages indicated in the ATM Forum signaling message column without correspondence primitive in the Native ATM Services primitives column should not be generated by the application.

## Appendix B.2 Messages received from Network

The left column of the following table contains the signaling messages and in the right column the resulting Native ATM Services primitives. The Native ATM Services primitives refer to indication or confirm primitives.

| ATM Forum signaling messages | Native ATM Services primitives |
|---|---|
| **CALL PROCEEDING** | None |
| **CONNECT** | ATM_P2P_call_active or ATM_P2MP_call_active |
| **CONNECT ACKNOWLEDGE** | ATM_P2P_call_active or ATM_P2MP_call_active |
| **SETUP** | ATM_arrival_of_incoming_call |
| **RELEASE** | ATM_call_release |
| **RELEASE COMPLETE** | None |
| **STATUS** | None [1] |
| **STATUS INQUIRY** | None |
| **RESTART** | ATM_call_release |
| **RESTART ACKNOWLEDGE** | None |
| **ADD PARTY** | Invalid under the restrictions in this specification |
| **ADD PARTY ACKNOWLEDGE** | ATM_add_party_success |
| **ADD PARTY REJECT** | ATM_add_party_reject |
| **DROP PARTY** | ATM_drop_party |
| **DROP PARTY ACKNOWLEDGE** | None |
| **LEAF_SETUP_REQUEST** | ATM_LIJ_join_requested |
| **LEAF_SETUP_FAILURE** | ATM_LIJ_leaf_rejected |
| **N/A – triggered by RM cell** | ATM_outbound_rate_changed |
| **N/A – triggered by RM cell** | ATM_inbound_rate_changed |
| | |

Signaling messages that correspond to ‘None’ (in the Native ATM Services primitives column) do not generate a primitive that reaches the application.

1 This depends on the cause information element. In some cases, this message may imply the release of the connection.

# Appendix C Example SAP Combinations [INFORMATIVE]

This appendix provides examples of users of Native ATM Services can be specify SAPs. This is not meant to restrict implementations, but rather to provide helpful guidance.

## Appendix C.1 SAPs for Data Link Layer Protocols

### Characteristics

This group of SAPs is used when the destination of the connection is a Data Link Layer protocol entity. This provides support for the transport of several network protocols over a single VC, which may be cost effective when the cost of a VC is a consideration. Examples of these data link protocols include LLC (Logical Link Control) and PPP (Point-to-Point Protocol).

### Semantic Definition

The SAP address consists of the ATM address, including the selector byte, and the identification of a Layer 2 protocol

### Specific Coding

The Layer 2 protocol specification is encoded in the following way:

ATM_addr.SVE_tag = PRESENT

ATM_addr.SVE_value = the ATM address, minus the selector byte

ATM_selector.SVE_tag = PRESENT (private ATM address) or ABSENT (E.164 address)

ATM_selector.SVE_value = the selector byte (from the private ATM address)

BLLI_id2.SVE_tag = PRESENT

BLLI_id2.SVE_value = the identification of the layer 2 protocol

BLLI_id3.SVE_tag = ABSENT

BHLI_id.SVE_tag = ABSENT

## Appendix C.2 SAPs for Network Layer Protocols

### Characteristics

This group of SAPs is used when the destination of the connection is a Network or Transport Layer protocol entity, or a service transport entity. This provides support for a single network or transport protocol, or a single service transport over a single VC. Examples of these network protocols include X.25 and IP. Examples of transport protocols are TCP and TP4. Examples of service transport entities are ATM LAN emulation and ATM Circuit emulation.

### Semantic Definition

The SAP address consists of the ATM address, including the selector byte, and the specification of a Layer 3 protocol.

### Specific Coding

The Layer 3 protocol specification is encoded in the following way:

ATM_addr.SVE_tag = PRESENT

ATM_addr.SVE_value = the ATM address, minus the selector byte

ATM_selector.SVE_tag = PRESENT (private ATM address) or ABSENT (E.164 address)

ATM_selector.SVE_value = the selector byte (from the private ATM address)

BLLI_id2.SVE_tag = ANY

BLLI_id3.SVE_tag = PRESENT

BLLI_id3.SVE_value = the identification of the layer 3 protocol (options 1 - 3 only)

BHLI_id.SVE_tag = ABSENT

## Appendix C.3 SAPs for Higher Layer Protocols and ATM-Aware Applications

### Characteristics

This group of SAPs is used when the destination of the connection is an ATM-aware application, or more properly, the session layer of an application. This provides for the transport of a single application's data over a single VC.

### Semantic Definition

The SAP address consists of the ATM address, including the selector byte, identification of the layer-2 protocol used (if present), identification of the layer-3 protocol used (if present), and the identification of the higher layer protocol or an ATM-aware application.

### Specific Coding

The ATM-aware application specification is encoded in the BHLI information element.

ATM_addr.SVE_tag = PRESENT

ATM_addr.SVE_value = the ATM address, minus the selector byte

ATM_selector.SVE_tag = PRESENT (private ATM address) or ABSENT (E.164 address)

ATM_selector.SVE_value = the selector byte (from the private ATM address)

BLLI_id2.SVE_tag = ABSENT or PRESENT

BLLI_id2.SVE_value = identification of the layer 2 protocol (if tag = PRESENT)

BLLI_id3.SVE_tag = ABSENT or PRESENT

BLLI_id3.SVE_value = identification of the layer 3 protocol (if tag = PRESENT)

BHLI_id.SVE_tag = PRESENT

BHLI_id.SVE_value = the identification of the ATM-aware application

# Appendix D Implementation Guidelines for the ABR API options of the Native ATM Services API [INFORMATIVE]

## Appendix D.1 Rationale for an API access to ABR Services

For connections using ATM layer CBR, VBR and even UBR services, their User-plane traffic characteristics are determined at the connection setup time. They are categorized as using "preventive control" for User-plane flow control. Applications or upper layer protocols send and receive their User-plane data in accordance with these pre-negotiated characteristics. For connections with ABR service, the application relies on a "reactive control" for the network based flow control mechanism. Traffic characteristics of ABR change (depending on the network status) during the lifetime of a connection.

The concept of dynamically changing connection attributes during the lifetime of a connection is a new and completely different function from the previous (UNI 3.x) API semantics. This flow control function is a User-plane capability.

Since the concept of flow control API is new, there have been a lot of discussions regarding the implementation impacts of ABR-API primitives at ATM Forum meetings. However, it is apparent from the simulation results for TCP-ABR interworking that the more we deal with high-speed networks, the more the importance of this new API capability increases.

### Appendix D.1.1 API Objectives

- **Application Perspective**

Once an access method to the lower layer capability is established for an application, other applications will also want to use the same or a similar access method. If another application wants to use exactly the same method, it is desirable for application writers to have a standardized API for the access method. When another application wants to use a similar access method creating a common standard API is beneficial. This is because an application can usually absorb differences from the standard API capability.

- **Network Perspective**

From the view-point of the network, the API is a very important functionality. At the initial stage of computer communication, networks had limited capability and as a result, the only APIs regarding User-plane data transmission provided simple read and write functionality. With the deployment of more advanced ATM technology, the network can provide various new capabilities to upper layer protocols regarding the data transmission. For ABR service, the new network capability of adapting the flow rate to network conditions can be made available for application layer use.

- **Relation of application and network capability**

Developing a new network service (ABR service) means deploying a new network capability. This is new capability can provide

1) improved performance for existing applications or
2) a base for new applications motivated by the new network capability or other applications we did not consider initially.

Once the APIs are specified, various applications that utilize the API may be independently developed and evolved. Providing the semantic specification of the API to access a new network capability eliminates the chicken and egg problem of which comes first – the application or the network capability.

### Appendix D.1.2 ABR Service Motivation

Flow control to avoid cell loss during network congestion was the motivation for the creation of the ABR service. By using ABR service for an ATM connection, we can solve cell loss problems at the ATM layer.

However, networked applications have been continuously evolving since the concept of ABR was developed. The ABR service can be deployed without the application layers being aware of the service parameters. For an implementation such as an ATM NIC in a PC, this requires adequate buffering on the NIC. Buffer requirements may be difficult to characterize, or may be considered excessive for general purpose hardware deployments. Performance for upper layer protocols (such as a TCP protocol stack implementation that has been tuned up for high-speed transmission) may be deteriorated. Although the ABR service may ensure there is minimal cell loss in the network, there may be buffer overruns between the PC and the NIC, which are seen as cell loss by the upper layer protocols. In these situations an API to extend the ABR capabilities to the upper layer protocols may be useful.

### Appendix D.1.3 Motivation for ABR API

When the ATM Forum specified the ABR Service, it was assumed that there may be several applications which can utilize the benefit of flow control. This can be true if we utilize flow control information from upper layers or if we install enough buffers to tolerate sudden bandwidth change due to flow control. For existing (legacy) applications, they cannot utilize feedback information. When we use these applications, sufficient buffers at ATM-edge devices are required. The adaptability to the flow control depends on NIC buffer size. Most of current (legacy) applications are based on software platform. Software control is slow ( compared to link transmission speeds) and takes time at least in the order of several hundred milli-seconds before the control effectively works. If we want to use such legacy applications in a high-speed ABR network environment, at least we need enough buffers to absorb the impact of flow control. Alternatively, we may utilize the ABR feedback information at the upper layer protocols. By using this information, the application itself can utilize the benefit of flow control.

### Appendix D.1.4 Application dependent usage

For example, software-based applications cannot adapt effectively to high-speed networks if they only guess network status by measuring packet loss etc. To effectively adapt the applications to high-speed networks, explicit rate notifications are necessary. To do this, APIs regarding the flow control capability, such as the ABR API, provided in this specification are necessary. Here, we must note that specifying semantics of ABR API does not mean we also specify only one implementation technique to use that capability. How often it is invoked or the implementation syntax are application specific aspects. However, this semantic specification be clarifies for the upper layer users what kind of information is available from the ATM layer.

We also note that there are not only software-based applications but also there are hardware-based applications. For hardware-based applications, they can also adapt to feedback from the high-speed network. A common semantic specification can also be used as a reference for hardware implementation of the primitives as well as software implementations.

### Appendix D.1.5 API implementation

The set of ABR API Primitives that the ATM device, (e.g. a NIC) implements is considered to be implementation specific. For example:

- If an ATM device (NIC) is intended to support all types of (hardware and/or software-based) ABR-applications, it is strongly recommended to implement all of the primitives. This approach, however, increases the NIC costs.
- If the ATM device (NIC) is expected to play more optimized role, only part of the primitives may be useful for a carefully targeted set of applications.

### Appendix D.1.6 ABR user

Finally, we would like to note who is the user of ABR service. We could say that the user is not restricted to end-users or application writers. All of the following could be ABR users:

- End-user
- Network administrator
- Application writer
- Protocol/OS developer
- NIC vendor
- Network provider

## Appendix D.2 Example Applications use of the ABR API Primitives

In many applications of the ABR service, the use of the ABR service is transparent to the application software layers. In these cases the ABR service parameters are configured via other means (e.g. management plane actions) and the ABR control loop may be entirely terminated within the NIC. In other circumstances, the application may wish to take advantage of the information provided by the ABR loop.

In particular, the rationale behind two primitives: "ATM_query_outbound_rate" and "ATM_query_inbound_rate" are considered in this section. The basis for this material is work on using ABR like mechanisms for supporting rate-adaptive video in ATM networks. Giving access to these primitives for other rate adaptive applications may also be desirable (e.g., rate adaptive audio).

### Appendix D.2.1 Specifying a Demand

Although typically data applications that are interested in using all of the available bandwidth would prefer to request the Peak Cell Rate (PCR) from the network, there are several cases where it may be useful to ask for a rate that is different from the PCR. The local application may be aware of its ability to use a rate, ER, that is lower than the PCR. It would result in better use of the network's resources to request an ER that is lower than PCR. In addition, we feel that an application that has the ability to ask for a rate that is lower than the PCR may be useful for rate-adaptive applications (not necessarily only best-effort data applications). These applications look to utilize the rate based feedback control mechanism specified in ATM's ABR service to adapt themselves to the available capacity in the network. An example of this is rate-adaptive video.

Because the ABR protocol allows the network to control the source rate directly, ABR schemes can be designed and tuned to control queueing delays at switches, in addition to controlling cell-loss rates. Given this, and if the Minimum Cell Rate (MCR) selected for the ABR connection is chosen to correspond to the minimum rate required by the video encoder, then the Available Cell Rate (ACR) determined by the ABR scheme may be used to determine the rate of the video encoder.

### Appendix D.2.2 Overview of an example implementation

One can use the inherent negotiation in ABR, via the ER field of RM cells, to allow sources to indicate their needed rates (demand) over very short intervals. The desired ideal rate that the source needs to transmit video at the ideal quality is communicated in the ER field of the Resource Management (RM) cells transmitted by the source. It is highly desirable for the application to request an ER value (ATM_request_outbound rate) based on the desired ideal rate for the video frame to be transmitted. This allows the network to potentially allocate a rate that is commensurate with the demand from the source (as seen in the ER field at the time of origination of the RM cell) and the network conditions.

### Appendix D.2.3 Using the Allocated Rate

As per the specifications in TM 4.0, the network computes an explicit rate that the source may transmit at, based on the available resources. This rate is communicated back to the source in the RM cell that returns.

The Available Cell Rate (ACR) value is then updated at the source, following the source rules specified in TM 4.0. The video source now adapts its rate to the rate communicated back by the network, whenever necessary. The source enhances its adaptation by using information about the source buffer occupancy and a recent set of allowed rates, subject to the constraints imposed by the ABR specification.

There may be a variety of video-source rate-adaptation mechanisms. For example, one can use a source buffer between the video encoder and the ATM layer. The rate at which the compressed video data was drained from the source buffer by the ATM layer may be the ACR that it is allowed to transmit at. However, the video encoder has to adapt it's bit rate to match the bandwidth granted by the network, to prevent overflow of the source buffer between the encoder and the ATM layer.

The source buffer serves to isolate the encoder from the rapid changes in the rate provided by the network, and also acts as an integrator of the difference between the encoder's desired rate and the allowed rate, ACR, over time. One can use a combination of the source buffer and the recent history of ACR returned to adapt the coder's rate. The source buffer also smooths errors in estimation of the feedback delay, thus minimizing rapid fluctuations in the coding rate, and hence avoiding impacting adversely the quality of the video. The encoder rate adaptation function accounts for both the ACR and the state of the source buffer. If the encoder were not to adapt, then either the source buffer would be very large (adversely impacting the end-end delay) or there would be overflow of the source buffer. Therefore, it is very desirable that the adaptation function have available to it the ACR that the network returns. ATM Forum contributions have been presented providing simulation results to demonstrate the effectiveness of an adaptation function, that utilized the ACR value.

## Appendix D.3 Example ABR Notification Mechanisms

The use of the ABR service class by an application opens unique opportunities, however its use is coincident with several tradeoffs and choices. An ABR service connection setup involves specifying several parameters; among these are peak cell rate (PCR), minimum cell rate (MCR), allowed cell rate (ACR), and initial cell rate (ICR).

Of these, the ACR is dynamic, its value varying between the MCR and PCR. Since the ACR (presumably) varies based on the load of the network, the service an application receives from the network also varies. On the other hand, some applications may be able to make clever use of the ACR and either meter data onto the network or perform more complex traffic shaping in an attempt to more closely match its demand of the network to the currently available supply of network service. The implication is that the application must, in some manner, be made aware of the rate at which it may send data. This section examines choices and tradeoffs in communicating this information to an application. It then proposes a simple two parameter mechanism which can be employed to control the level of active involvement of the NIC, and its associated hardware and drivers, in this effort. The two parameter mechanism specified can be implemented by either directly monitoring the ACR parameter or by using some other proxy ( e.g. queue size in the buffer).

### Appendix D.3.1 Discussion of ABR Notification mechanisms

An ABR capable NIC is the first to become aware of the ACR available on a particular connection, consequently it also keeps record of this value. Enabling an application to make use of this information means transferring this information from the NIC to the application and can happen one of several ways:

(1) The application POLLS the NIC
  pros:
    - no unnecessary communication,
    - tight application control of polling frequency
      or rate.
  cons:
    - may miss significant ACR behavior due to granularity,
    - high involvement of application in information
      gathering.

(2) NIC asynchronously "notifies" the application
  pros:
  - significant ACR events are not missed,
  - low "active" involvement of application.

  cons:
  - unnecessary notifications (periodic and no new info),
  - high level of NIC involvement in application oriented matters.

These choices collapse into some basic issues:

- Polling vs. notification
- Polling/notification frequency?
- Reducing the frequency by defining some trigger event. What is an "ACR event?" and further, what is a "significant" event?

High polling or notification frequency increases the granularity, hence the accuracy of the application's idea of the "actual" ACR. If the frequency is too small, information is lost. If the frequency is too high it creates other problems:

- It demands resources (that might be used for other work) just to convey information;
- the information may be redundant if the frequency may be higher than the change rate of the ACR.

The problem then is to find a balance between all these issues, but with the added caveat that it is simple to implement from the NIC perspective and easy to use from the application perspective.

Several issues must be considered when devising a solution:

- applications are all different and a specific application has the best knowledge of its own prospective service requirements of a network. This implies that the solution must be adaptable to the application without undue restrictions or complexity. Basically, there is no one "ideal" mechanism for all applications.
- it should be simple to implement in the NIC.
- it should be easy to use by the application while providing a wide range of choices as to method and frequency of ACR information.

## Appendix D.3.2 Implementation of ABR Notification based on ACR

Adhering to these constraints, a simple method was devised using a pair of parameters: *low* and *high*. These two parameters are specified at ABR connection setup time but can be changed throughout the life of the connection. The mechanism provides the ability to use either polling or notification, it allows variance of the polling or notification frequency, also providing the ability to combine the two methods if needed, and allows the disabling of the entire mechanism.

Imagine a line (Figure 31) which represents cell rates for a connection from zero up to some number N. We see the prescribed Minimum Cell Rate and Peak Cell Rate. Also in Figure 31 is the current ACR at some instant in time and is appropriately positioned within the bounds of the MCR and the PCR.

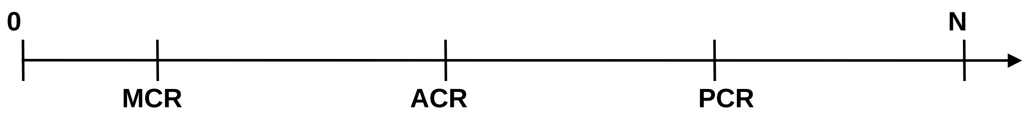

**Figure 31 ABR Cell Rates**

The new parameters *low* and *high* have a valid range of values is 0 to N inclusive where *low* <= *high*. They specify when to send a notification to the application. In general, notification is sent when the ACR rate either drops below the *low* or climbs above the *high*. There is a special provision to avoid thrashing across a boundary by mandating that *low* notifications occur only after a *high*, and vice versa. More formally:

*low* triggers a notification when:
1) the ACR changes from a higher value to a lower value, AND
2) the new value of ACR is less than or equal to *low*, AND
3) either (a) this is the first notification OR the last notification was triggered from *high*.

Similarly, *high* triggers a notification when:

1) the ACR changes from a lower value to a higher value, AND
2) the new value of ACR is greater than or equal to *high*, AND
3) either (a) this is the first notification OR (b) the last notification was triggered from *low*.

Several flavors of communication are now possible. The simplest, disabling the notification mechanism, is easily accomplished via setting the *low* and *high* to values above the PCR and is shown in Figure 32. Since the ACR will never be in that range, no notifications will occur. The application can now use polling or may ignore all ACR information altogether.

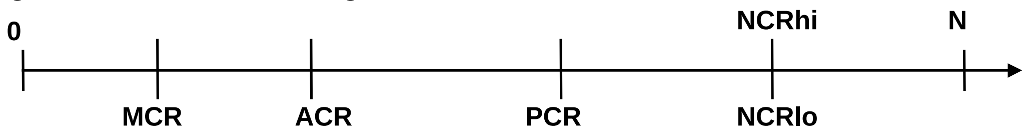

**Figure 32 ABR Rates in relation to *low* and *high* thresholds for disabled notifications**

Another possibility is that of simple binary notification, in other words two states: Service, No Service. This is shown in Figure 33 by setting the *low* = MCR-1 and *high* = MCR. In this case, when the ACR drops below the MCR (actually zero) a *low* notification occurs and when it rises back to at least MCR a *high* notification is triggered.

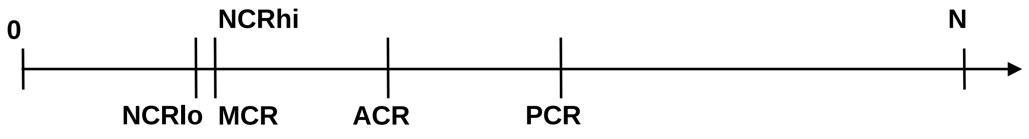

**Figure 33 ABR Rates and simple binary notifications**

To enable notification only if the ACR is at the MCR or the PCR, one would specify *low*=MCR and *high*=PCR. In this case no notifications occur if the ACR remains bounded by MCR and PCR. This is shown in Figure 34.

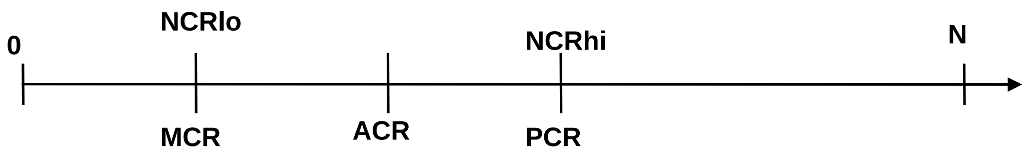

**Figure 34 ABR Rates with notifications at PCR/MCR**

This does not preclude the possibility of *low* and *high* being set within the bounds of MCR and PCR. Figure 35 shows one possibility in this scenario with ACR(n) and ACR(n-1) meaning the current ACR and the previously calculated ACR respectively.

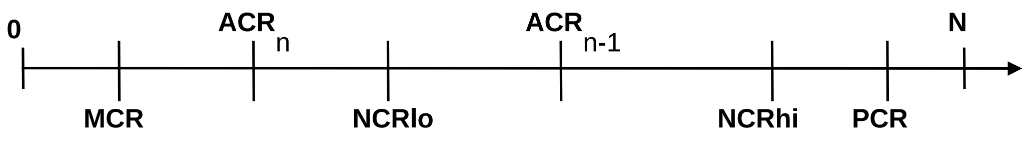

**Figure 35 ABR Rates with thresholds for adaptation**

Here the ACR has dropped from a higher value to a lower value and in the process crossed the *low* boundary. This will cause *low* notification. Another notification will not occur until the ACR rises to greater than or equal to the *high*. This mechanism then can be used to increase or decrease the frequency of notifications by decreasing or increasing the distance between *low* and *high* respectively. By setting *low* =*high* a notification will occur every time that boundary is crossed, however this can lead to wasteful thrashing about that boundary. The other direction where ACR(n) > ACR(n-1) works in a similar fashion.

## Appendix D.3.3 Implementation of ABR Notification based on Queue Size as a Proxy for ACR

The parameter for the service is specified as a cell rate. It may be convenient to utilize simple proxy parameters that are readily available in the NIC. The transmit queue size is one such proxy mechanism that may provide a convenient implementation approach, in that it is expected that this approach would not require any hardware changes to typical existing NIC designs.

Consider there are two implementation specific threshold H and L (where H>L) for the NIC transmit queue. When the queue length exceeds the threshold H, then ACR value is notified. After that, when the queue length decreases less than L, then the ACR is notified. In this case the NIC must prepare one bit of memory space per VC to remember whether the queue length has been above H and has still been above L.

# Appendix E Narrowband Voice Implementation Guidelines [Informative]

The incoming call distribution model has been extended from the previous version of the API semantics to take into account N-ISDN services provided in B-ISDN. The ATM Forum specification "Voice and Telephony Over ATM to the Desktop" is dependent on this for Native ATM Voice Services. For N-ISDN services, there is now a single, dedicated SAP. The application successfully registering this SAP would receive notification of all incoming calls that carry the "Narrow-band bearer capability" information element. The *destination_SAP* parameter of the ATM_connect_outgoing_call primitive is similarly extended.

For scenarios where there is only a single narrowband application present, then the narrowband application directly registers with Native ATM Services for the special narrowband SAP.

For scenarios where there are multiple narrowband applications present, then each narrowband application registers itself with a new functional component labeled herein "Narrowband Services Call Distribution". This new component has the responsibility of sharing the special narrowband SAP from Native ATM Services among multiple narrowband applications. In this case, the "Narrowband Services Call Distribution" component shall register with Native ATM Services for the special narrowband SAP.

There must be an API between the narrowband applications and the "Narrowband Services Call Distribution" component. The specification of this API is for further study.

A diagram showing the relationships between components follows in Figure 36. The normative text describing narrowband services SAP is found in section 4.5.2.1 and 4.6.3.

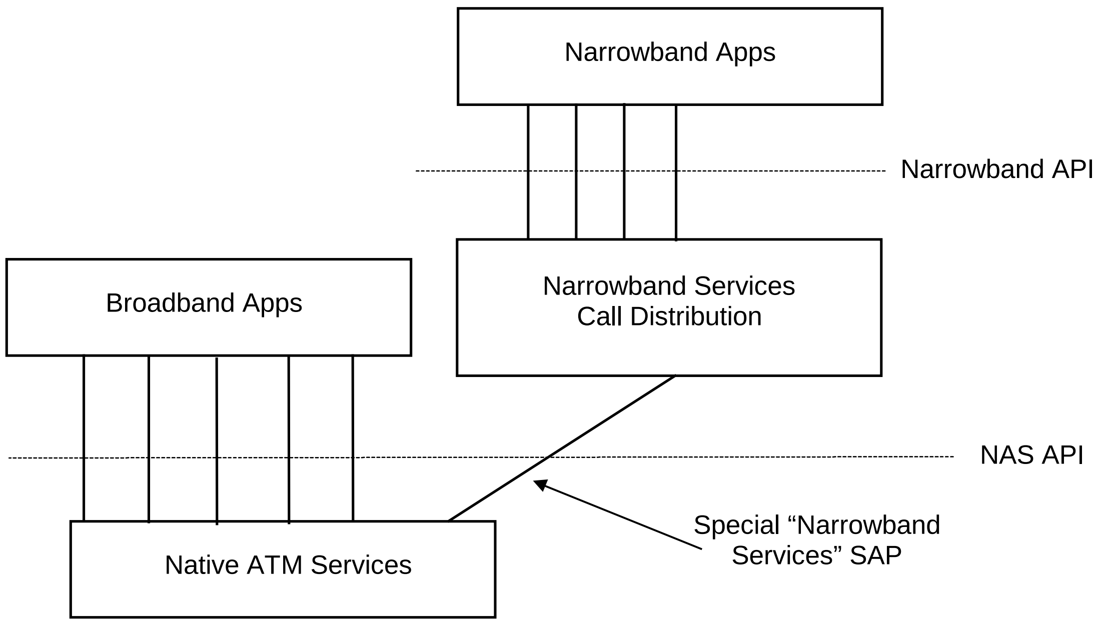
Narrowband Apps
Narrowband API
Broadband Apps
Narrowband Services Call Distribution
NAS API
Special "Narrowband Services" SAP
Native ATM Services

**Figure 36 Relationship of Narrowband Interworking SAP and NAS.**

## Appendix F Proxy Signaling Agent Implementation Guidelines [Informative]

Proxy Signaling, defined in Annex 2 of SIG 4.0, is an optional capability - for both the network and the user. This informative appendix contains guidelines for the implementation of a Proxy Signaling Agent (PSA) that utilizes the services of the NAS API. This capability, when supported by prior agreement between the user and the network, allows a user, called the Proxy Signaling Agent (PSA), to perform signalling on behalf of one or more users that do not support signaling. Figure 37, adapted from Annex 2 of SIG4.0, illustrates an example of PSA application usage of the NAS API.

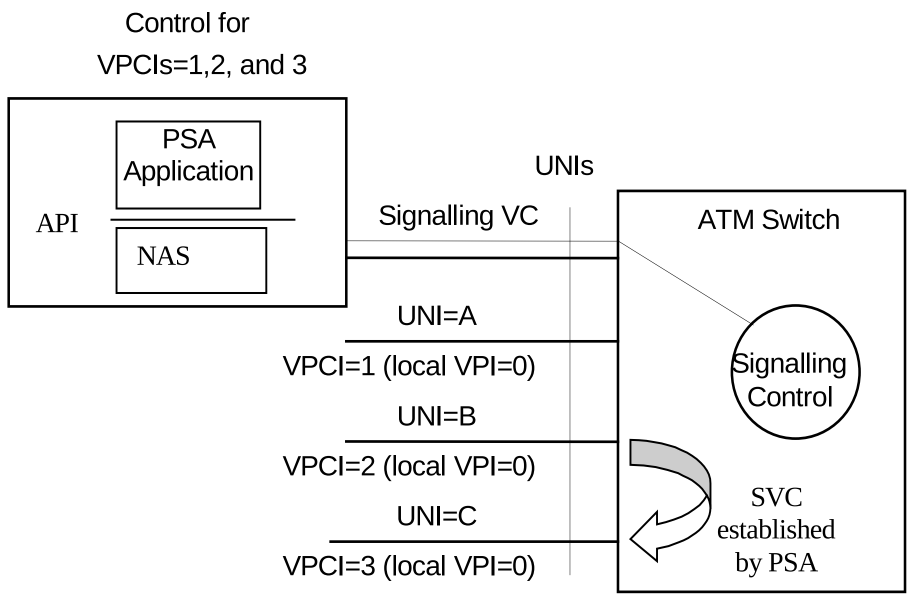

**Figure 37 Example PSA usage of NAS API**

In this example, the devices attached to UNIs A, B, and C do not support UNI signalling. There is one signalling channel between the ATM device containing the PSA and the ATM switch. In order for the PSA to initiate an SVC connection from a device on UNI B to a device on UNI C, the PSA creates an API_Endpoint corresponding to each side of the connection (origination, termination). The originating side API_Endpoint (corresponding to UNI B in this example) follows the procedures of section 4.2.1. The terminating side API_Endpoint (corresponding to UNI C in this example) follows the procedures of section 4.2.2.

The correlation of API_Endpoint to a UNI is done via the VPCI of the connection. On the originating side, the PSA application sets the VPCI in state A2 using the **ATM_set_connection_attributes** primitive. On the terminating side, the PSA application gets the VPCI in state A6 using the **ATM_query_connection_attributes** primitive. The VPCI is obtained via the “connection identifier” information element defined in SIG 4.0.

The coordination of data transfer on UNIs A, B, and C with the invocation of signaling functions by the PSA is beyond the scope of this specification. Per Annex 2 of SIG 4.0, the mapping of VPCI values to a specific UNI and VPI combination is resolved at service subscription time.