Consultative Committee for Space Data Systems

RECOMMENDATION FOR SPACE DATA SYSTEMS STANDARDS

ADVANCED ORBITING SYSTEMS, NETWORKS AND DATA LINKS:

FORMAL SPECIFICATION OF THE VCA SERVICE AND PROTOCOL

ADDENDUM TO CCSDS 701.0-B-2

CCSDS 705.4-B-1

BLUE BOOK

May 1994

AUTHORITY

Issue:	Blue Book, Issue 1
Date:	May 1994
Location:	Villafranca, Spain

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS Recommendations is detailed in reference [1], and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This Recommendation is published and maintained by:

CCSDS Secretariat
Program Integration Division (Code OI)
National Aeronautics and Space Administration
Washington, DC 20546, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of member space Agencies. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed RECOMMENDATIONS and are not considered binding on any Agency.

This RECOMMENDATION is issued by, and represents the consensus of, the CCSDS Plenary body. Agency endorsement of this RECOMMENDATION is entirely voluntary. Endorsement, however, indicates the following understandings:

o Whenever an Agency establishes a CCSDS-related STANDARD, this STANDARD will be in accord with the relevant RECOMMENDATION. Establishing such a STANDARD does not preclude other provisions which an Agency may develop.

o Whenever an Agency establishes a CCSDS-related STANDARD, the Agency will provide other CCSDS member Agencies with the following information:

  -- The STANDARD itself.

  -- The anticipated date of initial operational capability.

  -- The anticipated duration of operational service.

o Specific service arrangements shall be made via memoranda of agreement. Neither this RECOMMENDATION nor any ensuing STANDARD is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this Recommendation will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or, (3) be retired or cancelled.

In those instances when a new version of a RECOMMENDATION is issued, existing CCSDS-related Agency standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each Agency to determine when such standards or implementations are to be modified. Each Agency is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommendation.

FOREWORD

This document, which is a technical Recommendation prepared by the Consultative Committee for Space Data Systems (CCSDS), is intended for use by participating space Agencies in their development of “Advanced Orbiting Systems”.

This Recommendation, written using the ISO Formal Description Technique LOTOS, contains a formal specification of the VCA Protocol and Service, described in Natural Language in reference [2]. Annex A contains a set of tests, also written using LOTOS, which specify the required behaviour of the VCA Protocol and Service under certain control and input conditions.

The Abstract Data Types used within this document are given in full in reference [4], and the rationale behind the production of this formal specification is given in reference [7].

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommendation is therefore subject to CCSDS document management and change control procedures which are defined in reference [1].

DOCUMENT CONTROL

Document	**Title**	**Date**	**Status**
CCSDS 705.4-B-1	Recommendation for Space Data Systems Standards—Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the VCA Service and Protocol—Addendum to CCSDS 701.0-B-2, Issue 1	May 1994	Original Issue

CONTENTS

REFERENCES

[1] *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-6. Yellow Book. Issue 6. Washington, D.C.: CCSDS, May 1994 or later issue.

[2] *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification*. Recommendation for Space Data Systems Standards, CCSDS 701.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, November 1992 or later issue.

[3] *Information Processing Systems—Open Systems Interconnection—LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. ISO 8807. Issue 1. Geneva: ISO, 1989.

[4] *Advanced Orbiting Systems, Networks and Data Links: Abstract Data Type Library—Addendum to CCSDS 701.0-B-2.* Recommendation for Space Data Systems Standards, CCSDS 705.1-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[5] *Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the Path Service and Protocol—Addendum to CCSDS 701.0-B-2*. Recommendation for Space Data Systems Standards, CCSDS 705.2-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[6] *Advanced Orbiting Systems, Networks and Data Links: Formal Specification of the VCLC Service and Protocol—Addendum to CCSDS 701.0-B-2*. Recommendation for Space Data Systems Standards, CCSDS 705.3-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1994 or later issue.

[7] *Advanced Orbiting Systems, Networks and Data Links: Formal Definition of CPN Protocols, Methodology and Approach*. Report Concerning Space Data Systems Standards, CCSDS 705.0-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, October 1993 or later issue.

[8] *Advanced Orbiting Systems, Networks and Data Links: Summary of Concept, Rationale and Performance*. Report Concerning Space Data Systems Standards, CCSDS 700.0-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, November 1992 or later issue.

1 PURPOSE AND SCOPE

This document provides formal specifications of the Consultative Committee for Space Data Systems (CCSDS) Advanced Orbiting Systems (AOS) VCA service and protocol[1] using the ISO LOTOS formal description technique (refer to reference [3]). These formal specifications are not intended as replacements for the natural-language specifications provided in the AOS Blue Book (reference [2]), but as unambiguous expressions of those specifications, which may be used to clarify any problem areas.

This document is one of four CCSDS Recommendations that provide LOTOS specifications for the suite of AOS services and protocols (see references [4] through [6]). The relationship between the main AOS Recommendation and the four LOTOS Specifications is shown below; the numbers to the right are the CCSDS document references for the Recommendations containing the LOTOS Specifications.

ADT Library	705.1
Path Service	705.2
Path Protocol	705.2
VCLC Service	705.3
VCLC Protocol	705.3
VCA Service	705.4
VCA Protocol	705.4

A supporting CCSDS Report (reference [7]) contains the rationale, methodology, and approach used to prepare the LOTOS specifications.

These documents are expected to be of use primarily to the technical experts responsible for the design, configuration, and testing of AOS implementations; a basic knowledge of LOTOS is required to understand the formal specifications. Other users of the AOS services should consult the main AOS Recommendation and the companion CCSDS Report (references [2] and [8]).

[1]The natural-language specifications for the VCA service and protocol are contained in the AOS Blue Book, CCSDS 701.0-B-2, reference [2].

2 VIRTUAL CHANNEL ACCESS PROTOCOL

```
specification vca [vca, vcdu, insert, man, release, bitrate, phys]
                  (InsertErrorChecked : Bool,
                   GenerateInsertLossFlag : Bool,
                   FillPattern : OctetString) : noexit
```

This LOTOS specification represents the functional requirements of the Virtual Channel Access (VCA) Sublayer protocols, for use in the Space Link Subnetwork (SLS), as described in *Advanced Orbiting Systems, Networks and Data Links: Architectural Specification*, reference [2].

2.1 INTRODUCTION

The CCSDS Virtual Channel Access Service provides transfer of a number of different Service Data Units (SDUs) across the physical space channel. These SDUs may be formatted in one of three ways:

- VCA_SDUs are effectively the User Data fields of Virtual Channels. They are used to interact with the VCLC Service, but they may also be used to interact with other SLS Users who choose to format their own VCA_SDUs.

- IN_SDUs are fixed-length data units which are inserted by the VCA Service into each Virtual Channel Data Unit before they are transmitted. This service is used to transfer isochronous data such as audio.

- VC_PDUs are identical in format to the Protocol Data Unit generated by the VCA Service. This service is used to transfer Virtual Channel Data Units created by an independent ‘VCA’ entity.

2.1.1 Specification Architecture

The LOTOS specification is broken down internally as shown in Figure 2-1. The Space Link ARQ Procedures have been retained for the sake of compatibility with the AOS Blue Book (reference [2]); their operation is not covered in this specification. The interfaces are described in more detail below.

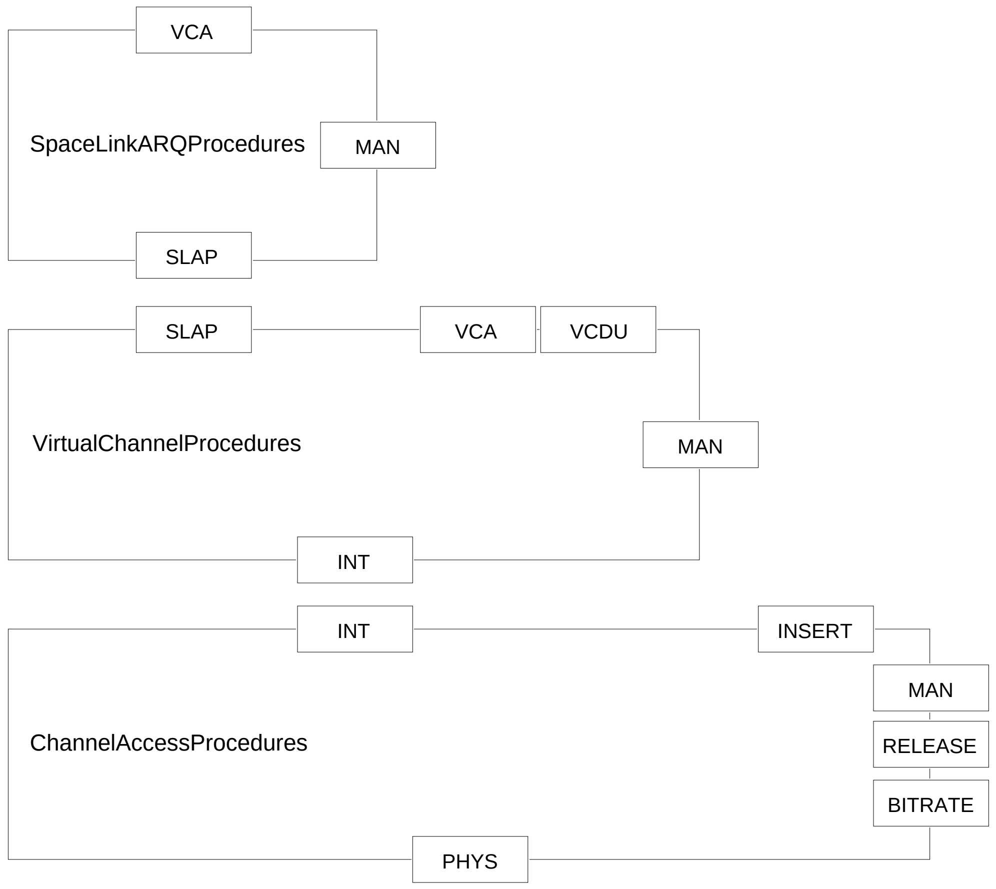

Figure 2-1: Internal Breakdown of LOTOS Specification

[VCA] represents the interface between the VCLC Service (or an SLS User with user-formatted VCA_SDUs) and a Virtual Channel Procedure (either directly or through the Space Link ARQ Procedures) .

[VCDU] represents the interface between an independent 'VCA' entity generating its own VCDUs and a Virtual Channel Procedure.

[INSERT] represents the interface between an Insert Service User and the Channel Access Procedures which implement (among other services) the Insert Service.

[RELEASE] represents the VCDU Release Algorithm which determines when VCDUs should be transmitted over the Link. The presence of such an algorithm is defined by the AOS Blue Book (reference [2]), but its constitution is project-specific; thus the LOTOS specification treats the Release Algorithm as an abstract interface over which instructions to release VCDUs are received.

[BITRATE] represents an abstract 'clock' which is used to synchronize the sending and receiving of bits to and from the Link.

[MAN] represents the interface with management for configuring the proper protocol layers, sub-layers, or procedures. Since the specifications are written to require a separate instantiation for each different stream of data (i.e., service instance) it is data received over this interface that effect the instantiation of entities with the required management information.

[PHYS] represents the interface between the VCA Service and the physical space channel (or Link).

Note – The [man] gate does not represent a true management interface; functions such as stopping, restarting and interrogating the protocol entities have not been specified. The [man] gate interaction only performs instantiation of protocol entities.

```
library VCDU, VCDULossFlag, InsertLossFlag, VCDUHeaderErrorControl,
        ReedSolomonCheckSymbols,SynchronizationMarker,
        VCDUOperationalControlField, Boolean, Bit, VCDUID,
        VCDUErrorControlField endlib
```

The Direction ADT is associated with both Physical and Virtual Channels, and is used in the Managed Information passed to the VCA Layer.

```
type  Direction is Boolean
sorts Direction
opns  Incoming        : -> Direction
      Outgoing        : -> Direction
      _Eq_            : Direction, Direction -> Bool
      _Ne_            : Direction, Direction -> Bool
eqns  forall x, y : Direction

      ofsort Bool

      Incoming Eq Incoming         = True ;
      Incoming Eq Outgoing         = False ;
      Outgoing Eq Incoming         = False ;
      Outgoing Eq Outgoing         = True ;

      x Ne y                       = Not(x Eq y) ;
endtype
```

The UpperLayer ADT represents part of the Managed Information passed to the VCA layer; it associates an upper-layer service (one of VCA, VCDU or SLAP) with each Virtual Channel.

```
type  UpperLayer is Boolean
sorts UpperLayer
opns  VCALayer    : -> UpperLayer
      VCDULayer   : -> UpperLayer
      SLAPLayer   : -> UpperLayer
      _Eq_        : UpperLayer, UpperLayer -> Bool
      _Ne_        : UpperLayer, UpperLayer -> Bool
eqns  forall x, y : UpperLayer

      ofsort Bool

      VCALayer  Eq VCALayer         = True ;
      VCALayer  Eq VCDULayer        = False ;
      VCALayer  Eq SLAPLayer        = False ;
      VCDULayer Eq VCALayer         = False ;
      VCDULayer Eq VCDULayer        = True ;
      VCDULayer Eq SLAPLayer        = False ;
      SLAPLayer Eq VCALayer         = False ;
      SLAPLayer Eq VCDULayer        = False ;
      SLAPLayer Eq SLAPLayer        = True ;

      x Ne y                        = Not(x Eq y) ;
endtype
```

The LinkID ADT is used to distinguish between Physical Channels in the case where the VCA Layer is connected to more than one Physical Channel. It is not carried in the VCDU Structure.

```
type  LinkID is Bit, Boolean
sorts LinkID
opns  NullLINKID            : -> LinkID
      Add                 : Bit, LinkID -> LinkID
      _Eq_                : LinkID, LinkID -> Bool
      _Ne_                : LinkID, LinkID -> Bool
eqns  forall b1, b2 : Bit, LINKID1, LINKID2 : LinkID

      ofsort Bool

      NullLINKID Eq NullLINKID             = True ;
      NullLINKID Eq Add(b1, LINKID1)       = False ;
      Add(b1, LINKID1) Eq NullLINKID       = False ;
      Add(b1, LINKID1) Eq Add(b2, LINKID2) = (b1 Eq b2) And (LINKID1 Eq LINKID2) ;
      LINKID1 Ne LINKID2                   = Not(LINKID1 Eq LINKID2) ;
endtype
```

The LinkIDList ADT represents a list of Link IDs, and is used by VCA to provide multicasting of a Virtual Channel over more than one Physical Channel.

```
type  LINKIDList is LinkID
sorts LINKIDList
opns  NullLINKIDList    : -> LINKIDList
      Add             : LinkID, LINKIDList -> LINKIDList
      Tail            : LINKIDList -> LINKIDList
      Head            : LINKIDList -> LinkID
eqns  forall x, y : LinkID, l : LINKIDList

      ofsort LinkID

      Head(NullLINKIDList)         = NullLINKID ;
      Head(Add(x, NullLINKIDList)) = x ;
      Head(Add(x, Add(y, l)))    = Head(Add(y, l)) ;

      ofsort LINKIDList

      Tail(NullLINKIDList)         = NullLINKIDList ;
      Tail(Add(x, NullLINKIDList)) = NullLINKIDList ;
      Tail(Add(x, Add(y, l)))    = Add(x, Tail(Add(y, l))) ;
endtype
```

```
type  VCDUQueue is VCDU, VCDUID
sorts VCDUQueue
opns  create   : -> VCDUQueue
      AddBack : VCDU, VCDUQueue -> VCDUQueue
      remove   : VCDUID, VCDUQueue -> VCDUQueue
      withdraw : VCDUID, VCDUQueue -> VCDUQueue
      expedite : VCDUID, VCDUQueue -> VCDU
      first    : VCDUQueue -> VCDU
      tail     : VCDUQueue -> VCDUQueue
      reverse  : VCDUQueue -> VCDUQueue
eqns  forall x, y : VCDUID, p, q : VCDU, z : VCDUQueue

      ofsort VCDU

      first(create)                       = NullVCDU ;
      first(AddBack(p, create))           = p ;
      first(AddBack(p, AddBack(q, z)))    = first(AddBack(q, z)) ;

      expedite(x, create)                 = NullVCDU ;

         GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     expedite(x, z)                       = first(z) ;

         GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     expedite(x, z) = expedite(x, remove(GetVCDUID(GetVCDUPrimaryHeader
                                            (first(z))), z)) ;

     ofsort VCDUQueue

     tail(create)                         = create ;
     tail(AddBack(p, create))             = create ;
     tail(AddBack(p, z))                  = AddBack(p, tail(z)) ;

     remove(x, create)                    = create ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     remove(x, z)                         = tail(z) ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     remove(x, z)                         = reverse(withdraw(x, z)) ;

     withdraw(x, create)                  = create ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     withdraw(x, z)                       = reverse(tail(z)) ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     withdraw(x, z)                       = AddBack(first(z),
                                                    withdraw(x, tail(z))) ;

     reverse(create)                      = create ;
     reverse(z)                           = AddBack(first(z), reverse(tail(z))) ;
endtype
```

```
behaviour

hide slap, int in

SpaceLinkARQProcedures [vca, man, slap]
|[slap]|
VirtualChannelProcedures [vca, vcdu, man, slap, int]
|[int]|
ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                        (InsertErrorChecked,
                         GenerateInsertLossFlag,
                         FillPattern)

where
```

The SLAP Procedures, although completely contained within the VCA sublayer, are sufficiently complex and detailed to put them outside the scope of the current Validation Programme. It is envisaged that the SLAP Procedural Entity will be specified as accepting VCA_SDUs on the GoVirtualChannel at the [vca] gate/generating SLAP_PDUs at the [slap] gate, and accepting SLAP_PDUs at the [slap] gate on the ReturnVirtualChannel/generating VCA_SDUs at the [vca] gate. The Virtual Channel Procedures are specified to support SLAP in this fashion.

```
process SpaceLinkARQProcedures [vca, man, slap] : noexit :=
(
     man ? GoVirtualChannel : VCDUID
         ? GoLink : LinkID
         ? ReturnVirtualChannel : VCDUID
         ? ReturnLink : LinkID
         ? GoVCPDULength : Nat
         ? ReturnVCPDULength : Nat ;
     (
          SpaceLinkARQProcedures [vca, man, slap]
          |||
          SpaceLinkARQProceduralEntity [vca, slap]
                                       (GoVirtualChannel,
                                        GoLink,
                                        ReturnVirtualChannel,
                                        ReturnLink,
                                        GoVCPDULength,
                                        ReturnVCPDULength)
      )
)

where

process SpaceLinkARQProceduralEntity [vca, slap]
                                    (GoVirtualChannel : VCDUID,
                                     GoLink : LinkID,
                                     ReturnVirtualChannel : VCDUID,
                                     ReturnLink : LinkID,
                                     GoVCPDULength : Nat,
                                     ReturnVCPDULength : Nat) : noexit :=
(
     stop
)
endproc SpaceLinkARQProceduralEntity

endproc SpaceLinkARQProcedures
```

In the following description, a value of all zeroes for the initial setting of the VCDUCounter is chosen for convenience only. An implementation may chose to initialise this count with some other value.

```
process VirtualChannelProcedures [vca, vcdu, man, slap, int] : noexit :=
(
     man ? VirtualChannel : VCDUID
         ? Direction : Direction
         ? RequiredLength : Nat
         ? UpperLayer : UpperLayer
         ? Links : LINKIDList ;
     (
          VirtualChannelProcedures [vca, vcdu, man, slap, int]
          |||
          VirtualChannelProceduralEntity [vca, vcdu, man, slap, int]
                                         (VirtualChannel,
                                          Direction,
                                          RequiredLength,
                                          MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                          Octet(0,0,0,0,0,0,0,0),
                                                         Octet(0,0,0,0,0,0,0,0)),
                                          UpperLayer,
                                          Links)
     )
)

where
```

The VirtualChannelProceduralEntity process covers the procedures specified in 5.4.9.1.1 of the AOS Blue Book (reference [2]). The process behaves either as a VCDUAssembly or a VCASDUExtraction process depending on the value of the Direction attribute passed by management. The interpretation of the RequiredLength attribute depends upon the value of the UpperLayer attribute: if the UpperLayer is the VCA UnitData service, then the RequiredLength refers to the VCA_SDU; if the UpperLayer is the SLAP service, then the RequiredLength refers to the SLAP_SDU; and, finally, if the UpperLayer is the Virtual Channel Data Unit service, then the RequiredLength refers to the VCDU/CVCDU.

```
process VirtualChannelProceduralEntity [vca, vcdu, man, slap, int]
                                       (VirtualChannel : VCDUID,
                                        ChannelDirection : Direction,
                                        RequiredLength : Nat,
                                        InitCounter : VCDUCounter,
                                        UpperLayer : UpperLayer,
                                        Links : LINKIDList) : noexit :=
(
       [ChannelDirection Eq Outgoing] ->
            man ! VirtualChannel
                   ? VCType : ReplayFlag ;
            VCDUAssembly [vca, vcdu, slap, int]
                  (VirtualChannel,
                   VCType,
                   RequiredLength,
                   InitCounter,
                   UpperLayer,
                   Links)
[]
[ChannelDirection Eq Incoming] ->
     VCASDUExtraction [vca, vcdu, slap, int]
                       (VirtualChannel,
                        RequiredLength,
                        InitCounter,
                        UpperLayer,
                        Head(Links))
)

where
```

```
      process VCDUAssembly [vca, vcdu, slap, int]
                   (VirtualChannel : VCDUID,
                    VCType : ReplayFlag,
                    RequiredLength : Nat,
                    InitCounter : VCDUCounter,
                    UpperLayer : UpperLayer,
                    Links : LINKIDList)
(
[UpperLayer Eq VCALayer] ->
     (
     vca ! VirtualChannel
         ? VCASDU : OctetString [LengthOf(VCASDU) Eq RequiredLength] ;
     DespatchVCDUs [int]
                   (MakeVCDU(MakeVCDUPrimaryHeader
                                    (Version2,
                                     VirtualChannel,
                                     Counter,
                                     MakeSignallingField(VCType,
                                                         VCDUSpare),
                                     NoVCDUHEC),
                              NullOS,
                              VCASDU,
                              NullTrailer,
                              NullOS),
                    VirtualChannel,
                    Links)
     >>
      VCDUAssembly [vca, vcdu, slap, int]
            (VirtualChannel,
             VCType,
             RequiredLength,
             Next(Counter),
             UpperLayer,
             Links)
     )
[]
[UpperLayer Eq VCDULayer] ->
     (
     vcdu ! VirtualChannel
          ? VCDU : VCDU [LengthOf(ConvertVCDUToOS(VCDU)) Eq RequiredLength] ;
     DespatchVCDUs [int] (VCDU, VirtualChannel, Links)
     >>
      VCDUAssembly [vca, vcdu, slap, int]
            (VirtualChannel,
             VCType,
             RequiredLength,
             Next(Counter),
             UpperLayer,
             Links)
     )

[]
[UpperLayer Eq SLAPLayer] ->
     (
     slap ! VirtualChannel
          ? SLAPPDU : OctetString [LengthOf(SLAPPDU) Eq RequiredLength] ;
     int ! Head(Links)
         ! VirtualChannel
         ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                                          VirtualChannel,
                                          Counter,
                                          MakeSignallingField(VCType,
                                                              VCDUSpare),
                                          NoVCDUHEC),
                    NullOS,
                    SLAPPDU,
                    NullTrailer,
                    NullOS);
      VCDUAssembly [vca, vcdu, slap, int]
            (VirtualChannel,
             VCType,
             RequiredLength,
             Next(Counter),
             UpperLayer,
             Links)
     )
)

where
```

```
process DespatchVCDUs [int] (VCDU : VCDU,
                             VirtualChannel : VCDUID,
                             Links : LINKIDList) : exit :=
(
[Head(Links) Eq NullLINKID] ->
     (
          exit
     )
[]
[Head(Links) Ne NullLINKID] ->
     (
          int ! Head(Links)
              ! VirtualChannel
              ! VCDU ;
          DespatchVCDUs [int] (VCDU, VirtualChannel, Tail(Links))
     )
)
endproc DespatchVCDUs

endproc VCDUAssembly
```

```
process VCASDUExtraction [vca, vcdu, slap, int]
                          (VirtualChannel : VCDUID,
                           RequiredLength : Nat,
                           Counter : VCDUCounter,
                           UpperLayer : UpperLayer,
                           Link : LinkID) : noexit :=
(
[UpperLayer Eq VCALayer] ->
     (
     int ! Link
         ! VirtualChannel
         ? VCDU : VCDU ;
          (
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Eq Counter)] ->
               (
               vca ! VirtualChannel
                   ! GetVCASDU(VCDU)
                   ! VCDUNotLost ;
               VCASDUExtraction [vca, vcdu, slap, int]
                                 (VirtualChannel,
                                  RequiredLength,
                                  Next(Counter),
                                  UpperLayer,
                                  Link)
               )
          []
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Ne Counter)] ->
               (
               vca ! VirtualChannel
                   ! GetVCASDU(VCDU)
                   ! VCDULost ;
               VCASDUExtraction [vca, vcdu, slap, int]
                                 (VirtualChannel,
                                  RequiredLength,
                                  Next(GetVCDUCounter
                                           (GetVCDUPrimaryHeader(VCDU))),
                                  UpperLayer,
                                  Link)
               )
          )
     )
[]
[UpperLayer Eq VCDULayer] ->
     (
     int ! Link
         ! VirtualChannel
         ? VCDU : VCDU ;
     vcdu ! VirtualChannel
          ! VCDU ;
     VCASDUExtraction [vca, vcdu, slap, int]
                       (VirtualChannel,
                        RequiredLength,
                        Counter,
                        UpperLayer,
                        Link)
     )
[]
[UpperLayer Eq SLAPLayer] ->
     (
     int ! Link
         ! VirtualChannel
         ? VCDU : VCDU ;
     slap ! VirtualChannel
          ! GetVCASDU(VCDU) ;
     VCASDUExtraction [vca, vcdu, slap, int]
                       (VirtualChannel,
                        RequiredLength,
                        Counter,
                        UpperLayer,
                        Link)
     )
)
endproc VCASDUExtraction

endproc VirtualChannelProceduralEntity

endproc VirtualChannelProcedures
```

```
process ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                                (InsertErrorChecked : Bool,
                                 GenerateInsertLossFlag : Bool,
                                 FillPattern : OctetString) : noexit :=
(
     man ? Link : LinkID
         ? VCPDUSize    : Nat
         ? DUZLength     : Nat
         ? InsertActive  : Bool
         ? InsertSize    : Nat
         ? BitTransReq : Bool
         ? Direction     : Direction ;
     (
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
     |||
     ChannelAccessProceduralEntity [insert, man, release, bitrate, phys, int]
                                   (Link,
                                    VCPDUSize,
                                    DUZLength,
                                    InsertActive,
                                    GenerateInsertLossFlag,
                                    InsertSize,
                                    InsertErrorChecked,
                                    BitTransReq,
                                    Direction,
                                    FillPattern)
     )
)

where
```

```
process ChannelAccessProceduralEntity [insert, man, release, bitrate, phys, int]
                                      (Link : LinkID,
                                       VCPDUSize : Nat,
                                       DUZLength : Nat,
                                       InsertActive : Bool,
                                       GenerateInsertLossFlag : Bool,
                                       InsertSize : Nat,
                                       InsertErrorChecked : Bool,
                                       BitTransReq : Bool,
                                       Direction : Direction,
                                       FillPattern : OctetString) : noexit :=
(
     [Direction Eq Outgoing] ->
     (
     hide ecef, delim in
          (
          VCDUCommutator [int, man, release, insert, ecef] (InsertActive,
                                                            InsertSize,
                                                            Link,
                                                            DUZLength,
                                                            FillPattern,
                                                            create)
          |[ecef]|
          ErrorControlEncoding [ecef, man, delim] (Link)
          |[delim]|
          Delimitor [delim, bitrate, phys] (Link,
                                            BitTransReq,
                                            NullBS)
          )
     )
     []
     [Direction Eq Incoming] ->
     (
     hide ecdf, synch in
          (
          VCDUDecommutator [int, ecdf] (Link)
          |[ecdf]|
          ErrorControlDecoding [ecdf, insert, man, synch] (Link,
                                                           InsertActive,
                                                           GenerateInsertLossFlag,
                                                           InsertSize,
                                                           InsertErrorChecked)
          |[synch]|
          Synchroniser [synch, phys] (Link,
                                      NullBS,
                                      BitTransReq,
                                      VCPDUSize,
                                      False)
          )
     )
)

where
```

The VCDUCommutator process broadly covers the function defined in the AOS Blue Book (reference [2]) under 5.4.9.1.2.1. The major difference is that the VCDUCommutator has no direct access to the external [vca] and [vcdu] service interfaces, and instead receives VCDUs/CVCDUs from the VCDUAssembly process over the [int] gate. The [release] gate represents an interface to the VC_PDU release algorithm which is not specified by CCSDS, but affects the order in which VC_PDUs are transmitted.

The VCDUCommutator process also contains the Insert injection function, this ensuring that the IN_SDU is in place before the CRC/Reed-Solomon encoding is carried out. Note that the process 'expects' the IN_SDU to be available at VC_PDU release time rather than accepting IN_SDUs asynchronously, this reflects 5.4.9.1.2.4.b in the AOS Blue Book.

```
process VCDUCommutator [int, man, release, insert, ecef]
                       (InsertActive : Bool,
                        InsertSize : Nat,
                        Link : LinkID,
                        DUZLength : Nat,
                        FillPattern : OctetString,
                        VCDUQueue : VCDUQueue) : noexit :=
(
     [Not(InsertActive)] ->
     (
          (
               int ! Link
                   ? VirtualChannel : VCDUID
                   ? VCDU : VCDU ;
               VCDUCommutator [int, man, release, insert, ecef]
                              (InsertActive,
                               InsertSize,
                               Link,
                               DUZLength,
                               FillPattern,
                               AddBack(VCDU, VCDUQueue))
          )
          []
(
               release ! Link
                       ? VCDUID : VCDUID ;
               (
               [expedite(VCDUID, VCDUQueue) Ne NullVCDU] ->
                    (
                    ecef ! Link
                         ! expedite(VCDUID, VCDUQueue) ;
                    VCDUCommutator [int, man, release, insert, ecef]
                                   (InsertActive,
                                    InsertSize,
                                    Link,
                                    DUZLength,
                                    FillPattern,
                                    Remove(VCDUID, VCDUQueue))
                    )
               []
               [expedite(VCDUID, VCDUQueue) Eq NullVCDU] ->
                    (
                    ecef ! Link
                         ! MakeFillVCDU(FillPattern, DUZLength) ;
                    VCDUCommutator [int, man, release, insert, ecef]
                                   (InsertActive,
                                    InsertSize,
                                    Link,
                                    DUZLength,
                                    FillPattern,
                                    VCDUQueue)
                    )
               )
          )
     )
```

```
[]
     [InsertActive] ->
     (
          (
               int ! Link
                   ? VirtualChannel : VCDUID
                   ? VCDU : VCDU ;
               VCDUCommutator [int, man, release, insert, ecef]
                              (InsertActive,
                               InsertSize,
                               Link,
                               DUZLength,
                               FillPattern,
                               AddBack(VCDU, VCDUQueue))
          )
          []
          (
               release ! Link
                       ? VCDUID : VCDUID ;
               (
               [expedite(VCDUID, VCDUQueue) Ne NullVCDU] ->
                    (
                    insert ! Link
                          ? INSDU : OctetString [LengthOf(INSDU) Eq InsertSize] ;
                    ecef ! Link
                         ! SetInsertZone(INSDU, expedite(VCDUID, VCDUQueue)) ;
                    VCDUCommutator [int, man, release, insert, ecef]
                                   (InsertActive,
                                    InsertSize,
                                    Link,
                                    DUZLength,
                                    FillPattern,
                                    Remove(VCDUID, VCDUQueue))
                    )
               []
               [expedite(VCDUID, VCDUQueue) Eq NullVCDU] ->
                    (
                    insert ! Link
                          ? INSDU : OctetString [LengthOf(INSDU) Eq InsertSize] ;
                    ecef ! Link
                         ! SetInsertZone(INSDU, MakeFillVCDU(FillPattern,
                                                             DUZLength)) ;
                    VCDUCommutator [int, man, release, insert, ecef]
                                   (InsertActive,
                                    InsertSize,
                                    Link,
                                    DUZLength,
                                    FillPattern,
                                    VCDUQueue)
                    )
               )
          )
     )
)
endproc VCDUCommutator
```

This process covers the procedures specified in section 5.4.9.1.2.3 of the AOS Blue Book (reference [2]). Certain combinations of error control encoding are not dealt with as they violate statements in the AOS Blue Book; for instance the VCDU Error Control Field is mandatory within Virtual Channels that are not Reed-Solomon encoded (5.4.9.2.1.4.2.b), and the VCDU Error Control and Operational Control Fields cannot be present if the VCDU Trailer is absent.

```
process ErrorControlEncoding [ecef, man, delim]
                             (Link : LinkID) : noexit :=
(
     ecef ! Link
          ? VCDU : VCDU ;
     man ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
         ? HeaderErrorControlRequired : Bool
         ? TrailerPresent : Bool
         ? VCDUErrorControlRequired : Bool
         ? OperationalControlFieldPresent : Bool
         ? ReedSolomonRequired : Bool
         ? ReedSolomonLength : Nat ;
     (
     AddOptionalFields [man] (VCDU,
                              HeaderErrorControlRequired,
                              TrailerPresent,
                              VCDUErrorControlRequired,
                              OperationalControlFieldPresent,
                              ReedSolomonRequired,
                              ReedSolomonLength)
     >>
     accept VCDU : VCDU in
          (
          delim ! Link
                ! VCDU ;
          ErrorControlEncoding [ecef, man, delim] (Link)
          )
     )
)

where
```

```
process AddOptionalFields [man]
                          (VCDU : VCDU,
                           HeaderErrorControlRequired : Bool,
                           TrailerPresent : Bool,
                           VCDUErrorControlRequired : Bool,
                           OperationalControlFieldPresent : Bool,
                           ReedSolomonRequired : Bool,
                           ReedSolomonLength : Nat) : exit (VCDU) :=
(
     AddHeaderErrorControl [man] (VCDU,
                                  HeaderErrorControlRequired)
     >>
     accept VCDU : VCDU in
     AddTrailer [man] (VCDU,
                       TrailerPresent,
                       VCDUErrorControlRequired,
                       OperationalControlFieldPresent)
     >>
     accept VCDU : VCDU in
     AddReedSolomon [man] (VCDU,
                           ReedSolomonRequired,
                           ReedSolomonLength)
     >>
     accept VCDU : VCDU in
     exit(VCDU)
)

where

process AddHeaderErrorControl [man] (VCDU : VCDU,
                                  HeaderErrorControlRequired : Bool) : exit (VCDU) :=
(
let PH : VCDUPrimaryHeader = GetVCDUPrimaryHeader(VCDU),
    IZ : OctetString = GetInsertZone(VCDU),
    DU : OctetString = GetVCASDU(VCDU),
    TR : VCDUTrailer = GetVCDUTrailer(VCDU),
    RS : OctetString = GetReedSolomon(VCDU)
in
     (
     [HeaderErrorControlRequired] ->
          exit(MakeVCDU(SetVCDUHEC(GenerateRS(2), PH),
                        IZ,
                        DU,
                        TR,
                        RS))
     []
     [Not(HeaderErrorControlRequired)] ->
          exit(MakeVCDU(SetVCDUHEC(NoVCDUHEC, PH),
                        IZ,
                        DU,
                        TR,
                        RS))
     )
)
endproc AddHeaderErrorControl
```

```
process AddTrailer [man] (VCDU : VCDU,
                          TrailerPresent : Bool,
                          VCDUErrorControlRequired : Bool,
                          OperationalControlFieldRequired : Bool) : exit (VCDU) :=
(
let PH : VCDUPrimaryHeader = GetVCDUPrimaryHeader(VCDU),
    IZ : OctetString = GetInsertZone(VCDU),
    DU : OctetString = GetVCASDU(VCDU),
    RS : OctetString = GetReedSolomon(VCDU)
in
     (
     [TrailerPresent] ->
          (
          [VCDUErrorControlRequired
           And
           OperationalControlFieldRequired] ->
               exit(MakeVCDU(PH,
                             IZ,
                             DU,
                             MakeVCDUTrailer(VCDUOCF,
                                             VCDUECF),
                             RS))
          []
          [Not(VCDUErrorControlRequired)
           And
           OperationalControlFieldRequired] ->
               exit(MakeVCDU(PH,
                             IZ,
                             DU,
                             MakeVCDUTrailer(VCDUOCF,
                                             NoECF),
                             RS))
          []
          [VCDUErrorControlRequired
           And
           Not(OperationalControlFieldRequired)] ->
               exit(MakeVCDU(PH,
                             IZ,
                             DU,
                             MakeVCDUTrailer(NoOCF,
                                             VCDUECF),
                             RS))
          )
     []
     [Not(TrailerPresent)] ->
          exit(MakeVCDU(PH,
                        IZ,
                        DU,
                        NullTrailer,
                        RS))
     )
)
endproc AddTrailer
```

```
process AddReedSolomon [man] (VCDU : VCDU,
                              ReedSolomonRequired : Bool,
                              ReedSolomonLength : Nat) : exit(VCDU) :=
(
let PH : VCDUPrimaryHeader = GetVCDUPrimaryHeader(VCDU),
    IZ : OctetString = GetInsertZone(VCDU),
    DU : OctetString = GetVCASDU(VCDU),
    TR : VCDUTrailer = GetVCDUTrailer(VCDU)
in
     (
     [ReedSolomonRequired] ->
          exit(MakeVCDU(PH,
                        IZ,
                        DU,
                        TR,
                        GenerateRS(ReedSolomonLength)))
     []
     [Not(ReedSolomonRequired)] ->
          exit(MakeVCDU(PH,
                        IZ,
                        DU,
                        TR,
                        NullOS))
     )
)
endproc AddReedSolomon

endproc AddOptionalFields

endproc ErrorControlEncoding
```

```
process Delimitor [delim, bitrate, phys]
                  (Link : LinkID,
                   BitTransReq : Bool,
                   SendBuffer : BitString) : noexit :=
(
     (
     delim ! Link
           ? VCDU : VCDU ;
          (
          [BitTransReq]->
               (
               BitScrambler[delim] (VCDU)
               >>
               accept VCDU : VCDU in
               Delimitor [delim, bitrate, phys]
                         (Link,
                          BitTransReq,
                          Append(Append(ConvertOSToBS(ConvertVCDUToOS(VCDU)),
                                        ConvertOSToBS(SyncMarker)),
                                 SendBuffer))
               )
          []
          [Not(BitTransReq)]->
               (
               Delimitor [delim, bitrate, phys]
                         (Link,
                          BitTransReq,
                          Append(Append(ConvertOSToBS(ConvertVCDUToOS(VCDU)),
                                        ConvertOSToBS(SyncMarker)),
                                 SendBuffer))
               )
          )
     )
     []
     (
     bitrate ;
     phys ! Link
          ! GetMSB(SendBuffer) ;
     Delimitor [delim, bitrate, phys] (Link,
                                       BitTransReq,
                                       RemoveMSB(SendBuffer))
     )
)

where
```

This process represents the optional Bit Transition Generation which may be applied to Channel Access Data Units (excluding the Synchronisation Marker). The application of the random sequence to the body of the CADU is not explicitly modelled; instead, the process merely passes the VCDU back to the calling process.

```
process BitScrambler [delim] (VCDU : VCDU) : exit (VCDU) :=
(
      exit(VCDU)
)
endproc BitScrambler

endproc Delimitor
```

```
process VCDUDecommutator [int, ecdf]
                         (Link : LinkID) : noexit :=
(
     ecdf ! Link
          ? VCDU : VCDU ;
     (
     [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU))) Ne FillVCID] ->
          (
          int ! Link
              ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
              ! VCDU ;
          VCDUDecommutator [int, ecdf] (Link)
          )
     []
     [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU))) Eq FillVCID] ->
          (
          VCDUDecommutator [int, ecdf] (Link)
          )
     )
)
endproc VCDUDecommutator
```

```
process ErrorControlDecoding [ecdf, insert, man, synch]
                             (Link : LinkID,
                              InsertActive : Bool,
                              GenerateInsertLossFlag : Bool,
                              InsertSize : Nat,
                              InsertErrorChecked : Bool) : noexit :=
(
     synch ! Link
           ? DataUnit : OctetString
           ? SynchLost : Bool ;
     (
          BuildVCDU [man] (DataUnit, InsertActive, InsertSize)
          >>
          accept VCDU : VCDU in
          (
          [InsertActive And InsertErrorChecked] ->
               (
                    (
                         (
                         [GenerateInsertLossFlag And SynchLost] ->
                              (
                              insert ! Link
                                     ! GetInsertZone(VCDU)
                                     ! INSDULost ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                         []
                         [GenerateInsertLossFlag And (Not(SynchLost))] ->
                              (
                              insert ! Link
                                     ! GetInsertZone(VCDU)
                                     ! INSDUNotLost ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                         []
                         [Not(GenerateInsertLossFlag)] ->
                              (
                              insert ! Link
                                     ! GetInsertZone(VCDU) ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                         )
                    )
                    []
                    ErrorControlDecoding [ecdf, insert, man, synch]
                                         (Link, InsertActive,
                                          GenerateInsertLossFlag,
                                          InsertSize,
                                          InsertErrorChecked)
               )
          []
          [InsertActive And (Not(InsertErrorChecked))] ->
               (
                    (
                    [GenerateInsertLossFlag And SynchLost] ->
                         (
                              (
                              Insert ! Link
                                     ! GetInsertZone(VCDU)
                                     ! INSDULost ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                              []
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                         )
                    []
                    [GenerateInsertLossFlag And (Not(SynchLost))] ->
                         (
                              (
                              Insert ! Link
                                     ! GetInsertZone(VCDU)
                                     ! INSDUNotLost ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                              []
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                         )
[]
                     [Not(GenerateInsertLossFlag)] ->
                         (
                              (
                              Insert ! Link
                                     ! GetInsertZone(VCDU) ;
                              ecdf ! Link
                                   ! VCDU ;
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link,InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                              )
                              []
                              ErrorControlDecoding [ecdf, insert, man, synch]
                                                   (Link, InsertActive,
                                                    GenerateInsertLossFlag,
                                                    InsertSize,
                                                    InsertErrorChecked)
                         )
                    )
               )
          []
          [Not(InsertActive)] ->
               (
                    (
                    ecdf ! Link
                         ! VCDU ;
                    ErrorControlDecoding [ecdf, insert, man, synch]
                                         (Link, InsertActive,
                                          GenerateInsertLossFlag,
                                          InsertSize,
                                          InsertErrorChecked)
                    )
                    []
                    ErrorControlDecoding [ecdf, insert, man, synch]
                                         (Link, InsertActive,
                                          GenerateInsertLossFlag,
                                          InsertSize,
                                          InsertErrorChecked)
               )
          )
     )
)

where
```

```
process BuildVCDU [man] (DataUnit : OctetString,
                         InsertActive : Bool,
                         InsertSize : Nat) : exit (VCDU) :=
(
     StripHeader [man] (DataUnit)
     >>
     accept PH : VCDUPrimaryHeader,
            DataUnit : OctetString,
            DuzSize : Nat,
            TrailerPresent : Bool,
            OCFPresent : Bool,
            ECFPresent : Bool in
     StripInsert [man] (PH, DataUnit,
                        InsertActive, InsertSize,
                        DUZSize, TrailerPresent,
                        OCFPresent, ECFPresent)
     >>
     accept PH : VCDUPrimaryHeader,
            InsertZone : OctetString,
            DataUnit : OctetString,
            DUZSize : Nat,
            TrailerPresent : Bool,
            OCFPresent : Bool,
            ECFPresent : Bool in
     StripDataZone [man] (PH, InsertZone,
                          DataUnit, DUZSize,
                          TrailerPresent, OCFPresent,
                          ECFPresent)
     >>
     accept PH : VCDUPrimaryHeader,
            InsertZone : OctetString,
            DataUnitZone : OctetString,
            DataUnit : OctetString,
            TrailerPresent : Bool,
            OCFPresent : Bool,
            ECFPresent : Bool in
     StripTrailer [man] (PH, InsertZone,
                         DataUnitZone, DataUnit,
                         TrailerPresent, OCFPresent,
                         ECFPresent)
     >>
     accept PH : VCDUPrimaryHeader,
            InsertZone : OctetString,
            DataUnitZone : OctetString,
            Trailer : VCDUTrailer,
            RS : OctetString in
     exit (MakeVCDU(PH, InsertZone, DataUnitZone, Trailer, RS))
)

where
```

```
process StripHeader [man]
                    (DataUnit : OctetString) : exit (VCDUPrimaryHeader,
                                                     OctetString,
                                                     Nat, Bool, Bool, Bool) :=
(
let VN : VersionNumber = MakeVersionNumber(Bit1(First(DataUnit)),
                                Bit2(First(DataUnit))),
    VCDUID : VCDUID =  MakeVCDUID(MakeSCID(Bit3(First(DataUnit)),
                                   Bit4(First(DataUnit)),
                                   Bit5(First(DataUnit)),
                                   Bit6(First(DataUnit)),
                                   Bit7(First(DataUnit)),
                                   Bit8(First(DataUnit)),
                                   Bit1(Nth(DataUnit, 2)),
                                   Bit2(Nth(DataUnit, 2))),
                          MakeVCID(Bit3(Nth(DataUnit, 2)),
                                   Bit4(Nth(DataUnit, 2)),
                                   Bit5(Nth(DataUnit, 2)),
                                   Bit6(Nth(DataUnit, 2)),
                                   Bit7(Nth(DataUnit, 2)),
                                   Bit8(Nth(DataUnit, 2)))),
    VC : VCDUCounter = MakeVCDUCounter(Nth(DataUnit, 3),
                              Nth(DataUnit, 4),
                              Nth(DataUnit, 5)),
    SF : SignallingField =
MakeSignallingField(MakeReplayFlag(Bit1(Nth(DataUnit, 6))),
                                  MakeVCDUSpare(Bit2(Nth(DataUnit, 6)),
                                                Bit3(Nth(DataUnit, 6)),
                                                Bit4(Nth(DataUnit, 6)),
                                                Bit5(Nth(DataUnit, 6)),
                                                Bit6(Nth(DataUnit, 6)),
                                                Bit7(Nth(DataUnit, 6)),
                                                Bit8(Nth(DataUnit, 6)))),
   HEC : OctetString = AddFront(Nth(DataUnit, 7),
                        AddFront(Nth(DataUnit, 8), NullOS))
in
(
          man ! VCDUID
              ? HECPresent : Bool
              ? DUZSize : Nat
              ? TrailerPresent : Bool
              ? OCFPresent : Bool
              ? ECFPresent : Bool ;
          (
          [HECPresent] ->
               (
                    exit(MakeVCDUPrimaryHeader(VN,VCDUID,VC,SF,HEC),
                         StripOctets(DataUnit, 8),
                         DUZSize,
                         TrailerPresent,
                         OCFPresent,
                         ECFPresent)
               )
          []
          [Not(HECPresent)] ->
               (
                    exit(MakeVCDUPrimaryHeader(VN,VCDUID,VC,SF,NullOS),
                         StripOctets(DataUnit, 6),
                         DUZSize,
                         TrailerPresent,
                         OCFPresent,
                         ECFPresent)
               )
          )
     )
)
endproc StripHeader
```

```
process StripInsert [man]
                    (PH : VCDUPrimaryHeader,
                     DataUnit : OctetString,
                     InsertActive : Bool,
                     InsertSize : Nat,
                     DUZSize : Nat,
                     TrailerPresent : Bool,
                     OCFPresent : Bool,
                     ECFPresent : Bool) : exit (VCDUPrimaryHeader,
                                                OctetString,
                                                OctetString,
                                                Nat, Bool, Bool, Bool) :=
(
[InsertActive] ->
     exit(PH,
          RetainOctets(DataUnit, InsertSize),
          StripOctets(DataUnit, InsertSize),
          DUZSize,
          TrailerPresent,
          OCFPresent,
          ECFPresent)
[]
[Not(InsertActive)] ->
     exit(PH, NullOS, DataUnit,
          DUZSize,
          TrailerPresent,
          OCFPresent,
          ECFPresent)
)
endproc StripInsert
```

```
process StripDataZone [man]
                      (PH : VCDUPrimaryHeader,
                       InsertZone : OctetString,
                       DataUnit : OctetString,
                       DUZSize : Nat,
                       TrailerPresent : Bool,
                       OCFPresent : Bool,
                       ECFPresent : Bool) : exit (VCDUPrimaryHeader,
                                                  OctetString,
                                                  OCtetString,
                                                  OctetString,
                                                  Bool, Bool, Bool) :=
(
     exit(PH,
          InsertZone,
          RetainOctets(DataUnit, DUZSize),
          StripOctets(DataUnit, DUZSize),
          TrailerPresent,
          OCFPresent,
          ECFPresent)
)
endproc StripDataZone
```

```
process StripTrailer [man] (PH : VCDUPrimaryHeader,
                            InsertZone : OctetString,
                            DataUnitZone : OctetString,
                            DataUnit : OctetString,
                            TrailerPresent : Bool,
                            OCFPresent : Bool,
                            ECFPresent : Bool) : exit(VCDUPrimaryHeader,
                                                      OctetString,
                                                      OctetString,
                                                      VCDUTrailer,
                                                      OctetString) :=
(
[TrailerPresent] ->
     (
     [OCFPresent And ECFPresent] ->
          exit(PH,
               InsertZone,
               DataUnitZone,
               MakeVCDUTrailer(AddFront(First(DataUnit),
                               AddFront(Nth(DataUnit, 2),
                               AddFront(Nth(DataUnit, 3),
                               AddFront(Nth(DataUnit, 4), NullOS)))),
                               AddFront(Nth(DataUnit, 5),
                               AddFront(Nth(DataUnit, 6), NullOS))),
               StripOctets(DataUnit, 6))
     []
     [OCFPresent And Not(ECFPresent)] ->
          exit(PH,
               InsertZone,
               DataUnitZone,
               MakeVCDUTrailer(AddFront(First(DataUnit),
                               AddFront(Nth(DataUnit, 2),
                               AddFront(Nth(DataUnit, 3),
                               AddFront(Nth(DataUnit, 4), NullOS)))),
                               NullOS),
               StripOctets(DataUnit, 4))
     []
     [Not(OCFPresent) And ECFPresent] ->
          exit(PH,
               InsertZone,
               DataUnitZone,
               MakeVCDUTrailer(NullOS,
                               AddFront(First(DataUnit),
                               AddFront(Nth(DataUnit, 2), NullOS))),
               StripOctets(DataUnit, 2))
     )
[]
[Not(TrailerPresent)] ->
     exit(PH, InsertZone, DataUnitZone, NullTrailer, DataUnit)
)
endproc StripTrailer

endproc BuildVCDU

endproc ErrorControlDecoding
```

```
process Synchroniser [synch, phys] (Link : LinkID,
                                    SyncSearch : BitString,
                                    BitTransReq : Bool,
                                    VCAPDUSize : Nat,
                                    SynchWasLost : Bool) : noexit :=
(
     phys ! Link
          ? BitIn : Bit ;
     (
     [AddLSB(BitIn, SyncSearch) Eq ConvertOSToBS(SyncMarker)] ->
          (
          GetVCAPDU [synch, phys] (Link,
                                   VCAPDUSize,
                                   NullOS,
                                   SynchWasLost,
                                   BitTransReq)
          >>
          Synchroniser [synch, phys] (Link,
                                      NullBS,
                                      BitTransReq,
                                      VCAPDUSize,
                                      False)
          )
     []
     [AddLSB(BitIn, SyncSearch) Ne ConvertOSToBS(SyncMarker)] ->
          (
          [LengthOf(SyncSearch) Lt Pred(8*4)] ->
               (
               Synchroniser [phys, synch]
                            (Link,
                             AddLSB(BitIn, SyncSearch),
                             BitTransReq,
                             VCAPDUSize,
                             False)
               )
          []
          [LengthOf(SyncSearch) Eq Pred(8*4)] ->
               (
               Synchroniser [phys, synch]
                            (Link,
                             AddLSB(BitIn, RemoveMSB(SyncSearch)),
                             BitTransReq,
                             VCAPDUSize,
                             True)
               )
          )
     )
)

where
```

```
process GetVCAPDU [synch, phys] (Link : LinkID,
                                 VCAPDUSize : Nat,
                                 OctetBuffer : OctetString,
                                 SynchWasLost : Bool,
                                 BitTransReq : Bool) : exit :=
(

[VCAPDUSize Eq 0] ->
     (
     [BitTransReq]->
          (
          BitDeScrambler[synch] (OctetBuffer)
          >>
          accept Buffer : OctetString in
          synch ! Link
                ! Buffer
                ! SynchWasLost ;
          exit
          )
     []
     [Not(BitTransReq)]->
          (
          synch ! Link
                ! OctetBuffer
                ! SynchWasLost ;
          exit
          )
     )
[]
[VCAPDUSize Gt 0] ->
     (
     ReadOctet [phys] (NatNum(0),
                       Link,
                       Octet(0,0,0,0,0,0,0,0))
     >>
     accept Octet : Octet in
     GetVCAPDU [synch, phys] (Link,
                              Pred(VCAPDUSize),
                              Append(AddFront(Octet, NullOS), OctetBuffer),
                              SynchWasLost,
                              BitTransReq)
     )
)

where
```

```
process ReadOctet [phys] (BitCounter : Nat,
                          Link : LinkID,
                          BitBuffer : Octet) : exit (Octet) :=
(
[BitCounter Eq 0] ->
     (
     phys  ! Link
           ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(BitIn, 0, 0, 0, 0, 0, 0, 0))
     )
[]
[BitCounter Eq 1] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), BitIn, 0, 0, 0, 0, 0, 0))

     )
[]
[BitCounter Eq 2] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), BitIn,
                             0, 0, 0, 0, 0))
     )
[]
[BitCounter Eq 3] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), Bit3(BitBuffer),
                             BitIn, 0, 0, 0, 0))
     )
[]
[BitCounter Eq 4] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), Bit3(BitBuffer),
                             Bit4(BitBuffer), BitIn, 0, 0, 0))
     )
[]
[BitCounter Eq 5] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), Bit3(BitBuffer),
                             Bit4(BitBuffer), Bit5(BitBuffer), BitIn, 0, 0))
     )
[]
[BitCounter Eq 6] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), Bit3(BitBuffer),
                             Bit4(BitBuffer), Bit5(BitBuffer), Bit6(BitBuffer),
                             BitIn, 0))
     )
[]
[BitCounter Eq 7] ->
     (
     phys ! Link
          ? BitIn : Bit ;
     ReadOctet [phys] (Succ(BitCounter), Link,
                       Octet(Bit1(BitBuffer), Bit2(BitBuffer), Bit3(BitBuffer),
                             Bit4(BitBuffer), Bit5(BitBuffer), Bit6(BitBuffer),
                             Bit7(BitBuffer), BitIn))
     )
[]
[BitCounter Eq 8] ->
     (
     exit (BitBuffer)
     )
)

endproc ReadOctet
```

This process represents the optional Bit Transition Removal which is necessary if Bit Transition Generation has been applied to Channel Access Data Units (excluding the Synchronisation Marker). The application of the random sequence to the body of the CADU is not explicitly modelled; instead, the process merely passes the VCDU back to the calling process.

```
process BitDeScrambler [synch] (Buffer : OctetString) : exit (OctetString) :=
(
      exit (Buffer)
)

endproc BitDeScrambler

endproc GetVCAPDU

endproc Synchroniser

endproc ChannelAccessProceduralEntity

endproc ChannelAccessProcedures

endspec
```

3 VIRTUAL CHANNEL ACCESS SERVICE

```
specification VCAService [vca, vcdu, insert, man, release]
                         (InsertLossFlags : Bool) : noexit

library VCDU, OctetString, VCDUCounter, VCDUHeaderErrorControl,
        VCDUID, Boolean, Bit, NaturalNumber, VCDULossFlag,
        InsertLossFlag  endlib
```

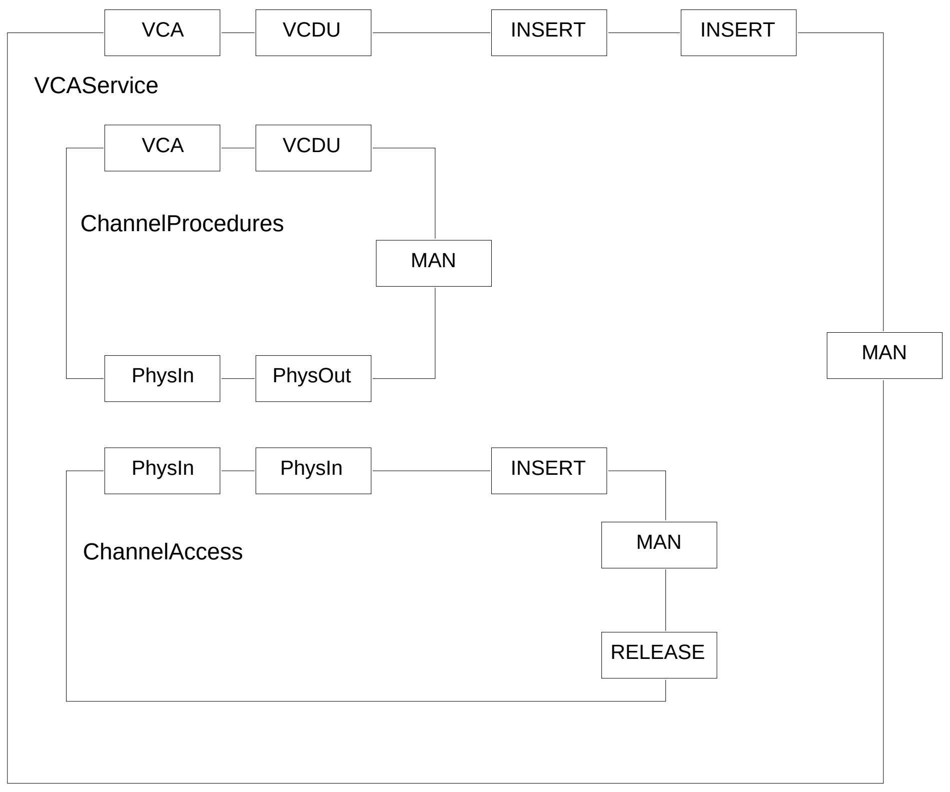

Figure 3-1: Internal Breakdown of LOTOS Specification

```
type UpperLayer is Boolean
sorts UpperLayer
opns  VCALayer    : -> UpperLayer
      VCDULayer   : -> UpperLayer
      SLAPLayer   : -> UpperLayer
      _Eq_        : UpperLayer, UpperLayer -> Bool
      _Ne_        : UpperLayer, UpperLayer -> Bool
eqns  forall x, y : UpperLayer

      ofsort Bool

      VCALayer  Eq VCALayer         = True ;
      VCALayer  Eq VCDULayer        = False ;
      VCALayer  Eq SLAPLayer        = False ;
      VCDULayer Eq VCALayer         = False ;
      VCDULayer Eq VCDULayer        = True ;
      VCDULayer Eq SLAPLayer        = False ;
      SLAPLayer Eq VCALayer         = False ;
      SLAPLayer Eq VCDULayer        = False ;
      SLAPLayer Eq SLAPLayer        = True ;

      x Ne y                        = Not(x Eq y) ;
endtype


type  LinkID is Bit, Boolean
sorts LinkID
opns  NullLINKID           : -> LinkID
      Add                 : Bit, LinkID -> LinkID
      _Eq_                : LinkID, LinkID -> Bool
      _Ne_                : LinkID, LinkID -> Bool
eqns  forall b1, b2 : Bit, LINKID1, LINKID2 : LinkID

      ofsort Bool

      NullLINKID Eq NullLINKID             = True ;
      NullLINKID Eq Add(b1, LINKID1)       = False ;
      Add(b1, LINKID1) Eq NullLINKID       = False ;
      Add(b1, LINKID1) Eq Add(b2, LINKID2) = (b1 Eq b2) And (LINKID1 Eq LINKID2) ;
      LINKID1 Ne LINKID2                   = Not(LINKID1 Eq LINKID2) ;
endtype


type  LINKIDList is LinkID
sorts LINKIDList
opns  NullLINKIDList    : -> LINKIDList
      Add             : LinkID, LINKIDList -> LINKIDList
      Tail            : LINKIDList -> LINKIDList
      Head            : LINKIDList -> LinkID
eqns  forall x, y : LinkID, l : LINKIDList

      ofsort LinkID

      Head(NullLINKIDList)         = NullLINKID ;
      Head(Add(x, NullLINKIDList)) = x ;
      Head(Add(x, Add(y, l)))    = Head(Add(y, l)) ;

      ofsort LINKIDList

      Tail(NullLINKIDList)         = NullLINKIDList ;
      Tail(Add(x, NullLINKIDList)) = NullLINKIDList ;
      Tail(Add(x, Add(y, l)))    = Add(x, Tail(Add(y, l))) ;
endtype
```

```
type  VCDUQueue is VCDU, VCDUID
sorts VCDUQueue
opns  create   : -> VCDUQueue
      AddBack : VCDU, VCDUQueue -> VCDUQueue
      remove   : VCDUID, VCDUQueue -> VCDUQueue
      withdraw : VCDUID, VCDUQueue -> VCDUQueue
      expedite : VCDUID, VCDUQueue -> VCDU
      first    : VCDUQueue -> VCDU
      tail     : VCDUQueue -> VCDUQueue
      reverse  : VCDUQueue -> VCDUQueue
eqns  forall x, y : VCDUID, p, q : VCDU, z : VCDUQueue

      ofsort VCDU

      first(create)                     = NullVCDU ;
      first(AddBack(p, create))         = p ;
      first(AddBack(p, AddBack(q, z)))  = first(AddBack(q, z)) ;

      expedite(x, create)               = NullVCDU ;

         GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     expedite(x, z)                     = first(z) ;

         GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     expedite(x, z)                     = expedite(x,
                         remove(GetVCDUID(GetVCDUPrimaryHeader(first(z))), z)) ;
     ofsort VCDUQueue

     tail(create)                       = create ;
     tail(AddBack(p, create))           = create ;
     tail(AddBack(p, z))                = AddBack(p, tail(z)) ;

     remove(x, create)                  = create ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     remove(x, z)                       = tail(z) ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     remove(x, z)                       = reverse(withdraw(x, z)) ;

     withdraw(x, create)                 = create ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Eq x =>
     withdraw(x, z)                     = reverse(tail(z)) ;

        GetVCDUID(GetVCDUPrimaryHeader(first(z))) Ne x =>
     withdraw(x, z)                     = AddBack(first(z),
                                                  withdraw(x, tail(z))) ;

     reverse(create)                    = create ;
     reverse(z)                         = AddBack(first(z), reverse(tail(z))) ;
endtype
```

```
behaviour

     VCAService [vca, vcdu, insert, man, release]
                (InsertLossFlags)
where
```

```
process VCAService [vca, vcdu, insert, man, release]
                   (InsertLossFlags : Bool) : noexit :=
(
     hide ChanIn, ChanOut in

     ChannelProcedures [vca, vcdu, insert, man, ChanIn, ChanOut]
     |[ChanIn, ChanOut]|
     ChannelAccess [ChanIn, ChanOut, insert, man, release] (InsertLossFlags)
)

where
```

```
process ChannelProcedures [vca, vcdu, insert, man, ChanIn, ChanOut] : noexit :=
(

     man ? VCDUID : VCDUID
         ? UpperLayer : UpperLayer
         ? RequiredLength : Nat
         ? InsertActive : Bool
         ? InsertErrorChecked : Bool
         ? Links : LINKIDList ;
     (
     [UpperLayer Eq VCDULayer] ->
          (
               VCDUChannel [vcdu, ChanIn, ChanOut]
                           (VCDUID,
                            RequiredLength,
                            Links)
               |||
               ChannelProcedures [vca, vcdu, insert, man, ChanIn, ChanOut]
          )
     []
     [UpperLayer Eq VCALayer] ->
          (
               VCAChannel [vca, insert, ChanIn, ChanOut]
                           (VCDUID,
                           RequiredLength,
                           InsertActive,
                           InsertErrorChecked,
                           Links,
                           MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                           Octet(0,0,0,0,0,0,0,0),
                                           Octet(0,0,0,0,0,0,0,0)))
               |||
               ChannelProcedures [vca, vcdu, insert, man, ChanIn, ChanOut]
          )
     )
)

where
```

```
process VCDUChannel [vcdu, ChanIn, ChanOut]
                    (VCDUID : VCDUID,
                     RequiredLength : Nat,
                     Links : LINKIDList) : noexit :=
(
     VCDUSource [vcdu, ChanIn] (VCDUID,
                                RequiredLength,
                                Links)
     |||
     VCDUSink [ChanOut, vcdu] (VCDUID)
)

where
```

```
process VCDUSource [vcdu, ChanIn]
                   (VCDUID : VCDUID,
                    RequiredLength : Nat,
                    Links : LINKIDList) : noexit :=
(
     vcdu ! VCDUID
          ? VCDU : VCDU [LengthOf(ConvertVCDUToOS(VCDU)) Eq RequiredLength] ;
     DespatchVCDUs [ChanIn] (VCDU, VCDUID, Links)
     >>
     VCDUSource [vcdu, ChanIn]
                (VCDUID,
                 RequiredLength,
                 Links)
)

endproc VCDUSource
```

```
process VCDUSink [ChanOut, vcdu]
                 (VCDUID : VCDUID) : noexit :=
(
     ChanOut ? Link : LinkID
             ! VCDUID
             ? VCDU : VCDU ;
     vcdu ! VCDUID
          ! VCDU ;
     VCDUSink [ChanOut, vcdu] (VCDUID)
)
endproc VCDUSink

endproc VCDUChannel
```

```
process VCAChannel [vca, insert, ChanIn, ChanOut]
                    (VCDUID             : VCDUID,
                    RequiredLength     : Nat,
                    InsertActive : Bool,
                    InsertErrorChecked : Bool,
                    Links   : LINKIDList,
                    Counter : VCDUCounter) : noexit :=
(
     VCASource [vca, ChanIn]
               (VCDUID,
                RequiredLength,
                Links,
                Counter)
     |||
     VCASink [ChanOut, vca, insert]
              (VCDUID,
               InsertActive,
               InsertErrorChecked,
               Counter)
)

where
```

```
process VCASource [vca, ChanIn]
                  (VCDUID : VCDUID,
                   RequiredLength : Nat,
                   Links : LINKIDList,
                   Counter : VCDUCounter) : noexit :=
(
     vca ! VCDUID
         ? VCASDU : OctetString [LengthOf(VCASDU) Eq RequiredLength] ;
     DespatchVCDUs [ChanIn]
              (MakeVCDU(MakeVCDUPrimaryHeader
                                (Version2,
                                 VCDUID,
                                 Counter,
                                 MakeSignallingField(RealTimeVCDU,
                                                     VCDUSpare),
                                 NoVCDUHEC),
                          NullOS,
                          VCASDU,
                          NullTrailer,
                          NullOS),
               VCDUID,
               Links)
     >>
     VCASource [vca, ChanIn]
               (VCDUID,
                RequiredLength,
                Links,
                Next(Counter))
)
endproc VCASource
```

```
process VCASink [ChanOut, vca, insert]
                (VCDUID             : VCDUID,
                 InsertActive       : Bool,
                 InsertErrorChecked : Bool,
                 Counter            : VCDUCounter) : noexit :=
(
     ChanOut ? Link : LinkID
             ! VCDUID
             ? VCDU : VCDU ;
     (
     [InsertActive And InsertErrorChecked]->
          (
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Eq Counter)] ->
               (
               vca ! VCDUID
                   ! GetVCASDU(VCDU)
                   ! VCDUNotLost ;
               insert ! Link
                      ! GetInsertZone(VCDU) ;
               VCASink [ChanOut, vca, insert]
                        (VCDUID,
                         InsertActive,
                         InsertErrorChecked,
                         Next(Counter))
               )
          []
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Ne Counter)] ->
               (
               vca ! VCDUID
                   ! GetVCASDU(VCDU)
                   ! VCDULost ;
               insert ! Link
                      ! GetInsertZone(VCDU) ;
               VCASink [ChanOut, vca, insert]
                        (VCDUID,
                         InsertActive,
                         InsertErrorChecked,
                         Next(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU))))
               )
          )
     []
     [Not(InsertActive) or Not(InsertErrorChecked)]->
          (
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Eq Counter)] ->
               (
               vca ! VCDUID
                   ! GetVCASDU(VCDU)
                   ! VCDUNotLost ;
               VCASink [ChanOut, vca, insert]
                        (VCDUID,
                         InsertActive,
                         InsertErrorChecked,
                         Next(Counter))
               )
          []
          [(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU)) Ne Counter)] ->
               (
               vca ! VCDUID
                   ! GetVCASDU(VCDU)
                   ! VCDULost ;
               VCASink [ChanOut, vca, insert]
                        (VCDUID,
                         InsertActive,
                         InsertErrorChecked,
                         Next(GetVCDUCounter(GetVCDUPrimaryHeader(VCDU))))
               )
          )
     )
)
endproc VCASink

endproc VCAChannel
```

```
process DespatchVCDUs [ChanIn] (VCDU : VCDU,
                                VirtualChannel : VCDUID,
                                Links : LINKIDList) : exit :=
(
[Head(Links) Eq NullLINKID] ->
     (
          exit
     )
[]
[Head(Links) Ne NullLINKID] ->
     (
          ChanIn ! Head(Links)
                 ! VirtualChannel
                 ! VCDU ;
          DespatchVCDUs [ChanIn] (VCDU, VirtualChannel, Tail(Links))
     )
)
endproc DespatchVCDUs


endproc ChannelProcedures
```

```
process ChannelAccess [ChanIn, ChanOut, insert, man, release]
                      (InsertLossFlags : Bool) : noexit :=
(
     man ? Link : LinkID
         ? InsertActive : Bool
         ? InsertSize : Nat
         ? InsertErrorChecked : Bool ;
     (
          ChannelAccessEntity [ChanIn, ChanOut, insert, release]
                              (InsertLossFlags,
                               Link,
                               InsertActive,
                               InsertSize,
                               InsertErrorChecked,
                               create,
                               False)
          |||
          ChannelAccess [ChanIn, ChanOut, insert, man, release]
                        (InsertLossFlags)
     )
)

where
```

```
process ChannelAccessEntity [ChanIn, ChanOut, insert, release]
                            (InsertLossFlags : Bool,
                             Link : LinkID,
                             InsertActive : Bool,
                             InsertSize : Nat,
                             InsertErrorChecked : Bool,
                             Queue : VCDUQueue,
                             SyncWasLost : Bool) : noexit :=
(
     (
          ChanIn ! Link
                 ? VCDUID : VCDUID
                 ? VCDU : VCDU ;
          ChannelAccessEntity [ChanIn, ChanOut, insert, release]
                              (InsertLossFlags,
                               Link,
                               InsertActive,
                               InsertSize,
                               InsertErrorChecked,
                               AddBack(VCDU, Queue),
                               SyncWasLost)
     )
     []
     (
          release ! Link
                  ? VCDUID : VCDUID ;
          (
          [expedite(VCDUID, Queue) Ne NullVCDU] ->
               (
               [InsertActive] ->
                    (
                         insert ! Link
                                ? INSDU : OctetString [LengthOf(INSDU) Eq
                                                       InsertSize] ;
                         ChannelErrors [ChanOut, insert]
                                       (SetInsertZone(INSDU,
                                                      expedite(VCDUID, Queue)),
                                        Link,
                                        InsertActive,
                                        InsertLossFlags,
                                        InsertErrorChecked,
                                        SyncWasLost)
                         >> accept SyncState : Bool in
                         ChannelAccessEntity [ChanIn, ChanOut,
                                              insert, release]
                                             (InsertLossFlags,
                                              Link,
                                              InsertActive,
                                              InsertSize,
                                              InsertErrorChecked,
                                              remove(VCDUID, Queue),
                                              SyncState)
                    )
               []
               [Not(InsertActive)] ->
                    (
                         ChannelErrors [ChanOut, insert]
                                       (expedite(VCDUID, Queue),
                                        Link,
                                        InsertActive,
                                        InsertLossFlags,
                                        InsertErrorChecked,
                                        SyncWasLost)
                         >> accept SyncState : Bool in
ChannelAccessEntity [ChanIn, ChanOut,
                                              insert, release]
                                             (InsertLossFlags,
                                              Link,
                                              InsertActive,
                                              InsertSize,
                                              InsertErrorChecked,
                                              remove(VCDUID, Queue),
                                              SyncState)
                    )
               )
          []
          [expedite(VCDUID, Queue) Eq NullVCDU] ->
               (
               [InsertActive] ->
                   (
                        insert ! Link
                               ? INSDU : OctetString [LengthOf(INSDU) Eq
                                                       InsertSize] ;
                         ChannelErrors [ChanOut, insert]
                                       (SetInsertZone
                                        (INSDU,
                                         MakeFillVCDU(AddFront
                                                  (Octet(0,0,0,0,0,0,0,0),
                                                  NullOS), succ(0))),
                                        Link,
                                        InsertActive,
                                        InsertLossFlags,
                                        InsertErrorChecked,
                                        SyncWasLost)
                         >> accept SyncState : Bool in
                         ChannelAccessEntity [ChanIn, ChanOut,
                                              insert, release]
                                             (InsertLossFlags,
                                              Link,
                                              InsertActive,
                                              InsertSize,
                                              InsertErrorChecked,
                                              Queue,
                                              SyncState)
                   )
               []
               [Not(InsertActive)] ->
                    ChannelAccessEntity [ChanIn, ChanOut,
                                         insert, release]
                                        (InsertLossFlags,
                                         Link,
                                         InsertActive,
                                         InsertSize,
                                         InsertErrorChecked,
                                         Queue,
                                         SyncWasLost)
               )
          )
     )
)

where
```

```
process ChannelErrors [ChanOut, insert]
                      (VCDU : VCDU,
                       Link : LinkID,
                       InsertActive : Bool,
                       InsertLossFlags,
                       InsertErrorChecked : Bool,
                       SyncWasLost : Bool) : exit (Bool) :=
(
[Not(InsertErrorChecked) And InsertActive] ->
     (
          (
          [SyncWasLost and InsertLossFlags]->
               (
                    insert ! Link
                           ! GetInsertZone(VCDU)
                           ! INSDULost;
                    (
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Ne FillVCID] ->
                         (
                         ChanOut ! Link
                                 ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
                                 ! VCDU ;
                         exit (False)
                         )
                    []
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Eq FillVCID] ->
                         (
                         exit (False)
                         )
                    )
               )
          []
          [Not(SyncWasLost) And InsertLossFlags]->
               (
                    insert ! Link
                           ! GetInsertZone(VCDU)
                           ! INSDUNotLost ;
                    (
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Ne FillVCID] ->
                         (
                         ChanOut ! Link
                                 ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
                                 ! VCDU ;
                         exit (False)
                         )
                    []
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Eq FillVCID] ->
                         (
                         exit (False)
                         )
                    )
               )
          []
          [Not(InsertLossFlags)]->
               (
                    insert ! Link
                           ! GetInsertZone(VCDU) ;
                    (
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Ne FillVCID] ->
                         (
                         ChanOut ! Link
                                 ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
                                 ! VCDU ;
                         exit (False)
                         )
                    []
                    [GetVCID(GetVCDUID(GetVCDUPrimaryHeader(VCDU)))
                     Eq FillVCID] ->
                         (
                         exit (False)
                         )
                    )
               )
          )
     []
          (
          [SyncWasLost And InsertLossFlags]->
               (
               insert ! Link
                      ! GetInsertZone(VCDU)
                      ! INSDULost ;
               exit (False)
               )
          []
          [Not(SyncWasLost) And InsertLossFlags]->
               (
               insert ! Link
                      ! GetInsertZone(VCDU)
                      ! INSDUNotLost ;
               exit (False)
               )
          []
          [Not(InsertLossFlags)]->
               (
               insert ! Link
                      ! GetInsertZone(VCDU) ;
               exit (False)
               )
          )
     []
          (
          exit (True)
          )
     )
[]
[InsertErrorChecked or Not(InsertActive)] ->
     (
          (
          ChanOut ! Link
                  ! GetVCDUID(GetVCDUPrimaryHeader(VCDU))
                  ! VCDU ;
          exit (False )
          )
     []
          (
          exit (True)
          )
     )
)
```

```
endproc ChannelErrors

endproc ChannelAccessEntity

endproc ChannelAccess

endproc VCAService

endspec
```

ANNEX A

VCA PROTOCOL AND SERVICE TESTS

(THIS ANNEX **IS NOT** PART OF THE RECOMMENDATION)

Purpose:

This Annex details procedures for testing the VCA protocol and service.

Virtual Channel Access Protocol Test 1

Name: VCA_SDU Length Check.

Description: The Channel Access Data Units (CADUs) for a Physical Channel must be of a fixed length over the lifetime of that channel [5.4.10.1]. The VCA Sublayer must therefore only accept VCA_SDUs which exactly fit the data zone of the VCDU/CVCDU to be transmitted over the Physical Channel in question.

Inputs: A VCA_UNITDATA.request with a VCA_SDU which exceeds a certain length X.

Configuration: Management information such that the required length of the VCDU/CVCDU data zone is Y, where Y<X.

Expected Results: This input should be rejected.

```
process vcapt1 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! OutGoing
         ! Succ(Succ(8))
         ! VCALayer
         ! NullLINKIDList ;

     (
          (
          vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ! AddFront(Octet(0,0,0,0,0,0,0,0), NullOS)  ;
          failure ;
          exit
          )
     []
          (
          success ;
          exit
          )
     )
)

endproc test

endproc vcapt1
```

Virtual Channel Access Protocol Test 2a

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2a (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2a
```

Virtual Channel Access Protocol Test 2b

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2b (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! False (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2b
```

Virtual Channel Access Protocol Test 2c

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2c (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! False (* Operational Control Field Present *)
         ! False (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2c
```

Virtual Channel Access Protocol Test 2d

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2d (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! False (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2d
```

Virtual Channel Access Protocol Test 2e

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2e (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! False (* Trailer Present *)
         ! False (* VCDUErrorControl Required *)
         ! False (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2e
```

Virtual Channel Access Protocol Test 2f

Name: VCDU/CVCDU format.

Description: Each VCDU-ID has associated with it a number of parameters defining (among other things) presence or absence of Reed-Solomon check symbols, presence or absence of SLAP protocol fields, presence or absence of VCDU Header Error field, presence or absence of Insert Zone (and length) and the presence or absence of the VCDU Trailer field (and its contents). Each of these parameters affect the VCDU/CVCDU format.

Inputs: Correctly formatted VCA_UNITDATA.request.

Configuration: The configuration data should be arranged to test the production of each field in turn, i.e., each field should be set to both true and then to false, and the outputs compared.

Expected Results: The generated VCDU/CVCDUs should be formatted as per the configuration data.

```
process vcapt2f (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8        (* VCPDUSize *)
         ! 4          (* DUZLength *)
         ! False      (* InsertActive *)
         ! 2          (* InsertSize *)
         ! False      (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! False (* HEC Required *)
         ! False (* Trailer Present *)
         ! False (* VCDUErrorControl Required *)
         ! False (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 4     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt2f
```

Virtual Channel Access Protocol Test 3

Name: VCDU/CVCDU Discontinuities.

Description: The VCA protocol should, if required, generate a VCDU Data Loss Flag derived from the VCDU/CVCDU Counter [5.4.7.2.e].

Inputs: Three VCDUs/CVCDUs on the same VCDU-ID, the first with a Counter field set to X, the second with a Counter field set to Y, where Y<>X+1, the third with a Counter field set to W, where W=Y+1.

Configuration: The VCA protocol should be configured to pass Data Loss Flags, and to expect a VCDU/CVCDU with a Counter set to X.

Expected Results: Three VCA_UNITDATA.indications should be generated, the first with the a Data Loss Flag set to false, the second with a Data Loss Flag set to true, the third with a Data Loss flag set to false.

```
process vcapt3 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! Incoming
         ! Succ(Succ(8))
         ! VCALayer
         ! Add(Add(1, NullLINKID), NullLINKIDList) ;

     (
          int ! Add(1, NullLINKID)
              ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                           MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                      MakeVCID(0,0,0,0,1,1)),
                           MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                           Octet(0,0,0,0,0,0,0,0),
                                           Octet(0,0,0,0,0,0,0,0)),
                                             MakeSignallingField(RealTimeVCDU,
                                                                 VCDUSpare),
                                             VCDUHEC),
                         NullOS,
                         AddFront(Octet(0,0,0,0,0,0,0,1), NUllOS),
                         NullTrailer,
                         NullOS) ;

          vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ? DataZone : OctetString
              ? DLF : VCDULossFlag [DLF Eq VCDUNotLost] ;
          int ! Add(1, NullLINKID)
              ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                                  MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                             MakeVCID(0,0,0,0,1,1)),
                                  MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                  Octet(0,0,0,0,0,0,0,0),
                                                  Octet(0,0,0,0,0,1,0,1)),
                                  MakeSignallingField(RealTimeVCDU,
                                                      VCDUSpare),
                                  VCDUHEC),
                         NullOS,
                         AddFront(Octet(0,0,0,0,0,0,0,1), NUllOS),
                         NullTrailer,
                         NullOS) ;
vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                            MakeVCID(0,0,0,0,1,1))
              ? DataZone : OctetString
              ? DLF : VCDULossFlag [DLF Eq VCDULost] ;

          int ! Add(1, NullLINKID)
              ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                            MakeVCID(0,0,0,0,1,1))
              ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                                          MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                                     MakeVCID(0,0,0,0,1,1)),
                                          MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                          Octet(0,0,0,0,0,0,0,0),
                                                          Octet(0,0,0,0,0,1,1,0)),
                                          MakeSignallingField(RealTimeVCDU,
                                                               VCDUSpare),
                                          VCDUHEC),
                         NullOS,
                         AddFront(Octet(0,0,0,0,0,0,0,1), NUllOS),
                         NullTrailer,
                         NullOS) ;

          vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                            MakeVCID(0,0,0,0,1,1))
              ? DataZone : OctetString
              ? DLF : VCDULossFlag [DLF Eq VCDUNotLost] ;

          success ;
          exit
     )
     []
     (
          failure ;
          exit
     )
)
endproc test

endproc vcapt3
```

Virtual Channel Access Protocol Test 4

Name: Inappropriate VCDU-ID.

Description: Each VCDU-ID is associated with an upper layer service interface [5.5.1.1.c]. Use of the VCA service should not be allowed on a VCDU-ID which is associated with the VCDU service.

Inputs: A correctly formatted VCA_UNITDATA.request on VCDU-ID X.

Configuration: VCDU-ID X should be associated with the VCDU service.

Expected Results: This input should result in an error condition.

```
process vcapt4 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! OutGoing
         ! Succ(Succ(8))
         ! VCDULayer
         ! NullLINKIDList ;

     (
          (
          vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ! AddFront(Octet(0,0,0,0,0,0,0,0),
                 AddFront(Octet(0,0,0,0,0,0,0,1),
                  AddFront(Octet(0,0,0,0,0,0,1,0),
                   AddFront(Octet(0,0,0,0,0,0,1,1),
                    AddFront(Octet(0,0,0,0,0,1,0,0),
                     AddFront(Octet(0,0,0,0,0,1,0,1),
                      AddFront(Octet(0,0,0,0,0,1,1,0),
                       AddFront(Octet(0,0,0,0,0,1,1,1),
                        AddFront(Octet(0,0,0,0,1,0,0,0),
                         AddFront(Octet(0,0,0,0,1,0,0,1), NullOS))))))))))  ;
          failure ;
          exit
          )
     []
          (
          success ;
          exit
          )
     )
)

endproc test

endproc vcapt4
```

Virtual Channel Access Protocol Test 5

Name: Unknown VCDU-ID.

Description: A VCDU-ID forms part of the VCA_UNITDATA.request.

Inputs: A correctly formatted VCA_UNITDATA.request using VCDU-ID X.

Configuration: Any Management Information which excludes VCDU-ID X.

Expected Results: This input should result in an error condition.

```
process vcapt5 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,1),
                      MakeVCID(0,0,0,0,1,1))
         ! OutGoing
         ! Succ(Succ(8))
         ! VCALayer
         ! NullLINKIDList ;

     (
          (
          vca ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                           MakeVCID(0,0,0,0,1,1))
              ! AddFront(Octet(0,0,0,0,0,0,0,0),
                 AddFront(Octet(0,0,0,0,0,0,0,1),
                  AddFront(Octet(0,0,0,0,0,0,1,0),
                   AddFront(Octet(0,0,0,0,0,0,1,1),
                    AddFront(Octet(0,0,0,0,0,1,0,0),
                     AddFront(Octet(0,0,0,0,0,1,0,1),
                      AddFront(Octet(0,0,0,0,0,1,1,0),
                       AddFront(Octet(0,0,0,0,0,1,1,1),
                        AddFront(Octet(0,0,0,0,1,0,0,0),
                         AddFront(Octet(0,0,0,0,1,0,0,1), NullOS))))))))))  ;
          failure ;
          exit
          )
     []
          (
          success ;
          exit
          )
     )
)

endproc test

endproc vcapt5
```

Virtual Channel Access Protocol Test 6

Name: IN_SDU Length Check.

Description: The Channel Access Data Units (CADUs) for a Physical Channel must be of a fixed length over the lifetime of that channel [5.4.10.1]. The Insert Layer must therefore only accept IN_SDUs which exactly fit the Insert Data Zones of the VCDUs/CVCDUs to be transmitted over the Physical Channel in question.

Inputs: An INSERT.request with an IN_SDU of a certain length X. An INSERT.request with an IN_SDU of a certain length Y.

Configuration: Management information such that the required length of the VCDU/CVCDU Insert Data Zone is W, where $Y<W<X$.

Expected Results: Both inputs should result in error conditions.

```
process vcapt6 (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8
         ! 4
         ! True
         ! 2
         ! True
         ! OutGoing ;

     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;
     (
          (
          insert ! Add(1, NullLINKID)
                 ! AddFront(Octet(0,0,0,0,0,0,0,1), NullOS) ;
          failure ;
          exit
          )
     []
          (
          insert ! Add(1, NullLINKID)
                 ! AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1), NullOS))) ;
          failure ;
          exit
          )
     []
          (
          insert ! Add(1, NullLINKID)
                 ! AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0), NullOS)) ;
          success ;
          exit
          )
     )
)
endproc test

endproc vcapt6
```

Virtual Channel Access Test 7

Name: Insert SDU Discontinuities.

Description: The Insert protocol should, if required, generate a VCDU Data Loss Flag derived from the VCDU/CVCDU Counter [5.4.7.4.e].

Inputs: Three VCDUs/CVCDUs on the same VCDU-ID, the first with a Counter field set to X, the second with a Counter field set to Y, where Y<>X+1, the third with a Counter field set to W, where W=Y+1.

Configuration: The Insert protocol should be configured to pass Data Loss Flags, and to expect a VCDU/CVCDU with a Counter set to X.

Expected Results: Three INSERT.indications should be generated, the first with the a Data Loss Flag set to false, the second with a Data Loss Flag set to true, the third with a Data Loss flag set to false.

Unfortunately, the ESPRIT LOTOS toolset can not cope with the number of Physical Channel indications necessary to build up VCDUs, so this test can not be executed. It should, however, form part of the test plan for the implementations.

Virtual Channel Access Protocol Test 8

Name: Correct Function of the VC_PDU Release Parameters.

Description: Each VCDU-ID has associated with it parameters which define when the VC_PDU must be released for transmission.

Inputs: Correctly formatted VCA_VCDU.requests with known content.

Configuration: The VCDU Protocol should be configured to release VC_PDUs at known intervals.

Expected Results: CADUs should be transmitted at the required rate.

```
process vcapt8 (InsertErrorChecked : Bool,
                GenerateInsertLossFlag : Bool,
                FillPattern : OctetString) : noexit :=
hide insert, man, release, bitrate, phys, int in
(
     test [insert, man, release, bitrate, phys, int]
     |[insert, man, release, bitrate, phys, int]|
     ChannelAccessProcedures [insert, man, release, bitrate, phys, int]
                             (InsertErrorChecked,
                              GenerateInsertLossFlag,
                              FillPattern)
)

where

process test [insert, man, release, bitrate, phys, int] : exit :=

hide success, failure in
(
     man ! Add(1, NullLINKID)
         ! 2*8         (* VCPDUSize *)
         ! 4           (* DUZLength *)
         ! False       (* InsertActive *)
         ! 2           (* InsertSize *)
         ! True        (* BitTransReq *)
         ! OutGoing ;

     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,0,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,0,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                    NullOS,
                    AddFront(Octet(0,0,0,0,0,0,0,1),
                     AddFront(Octet(0,0,0,0,0,0,1,0),
                      AddFront(Octet(0,0,0,0,0,0,1,1),
                       AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                    NullTrailer,
                    NullOS) ;
     int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,0))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                 MakeVCID(0,0,0,0,1,0)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                    NullOS,
                    AddFront(Octet(0,0,0,0,0,0,0,1),
                     AddFront(Octet(0,0,0,0,0,0,1,0),
                      AddFront(Octet(0,0,0,0,0,0,1,1),
                       AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                    NullTrailer,
                    NullOS) ;
```

```
int ! Add(1, NullLINKID)
         ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! MakeVCDU(MakeVCDUPrimaryHeader
                     (Version2,
                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                  MakeVCID(0,0,0,0,1,1)),
                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0),
                                      Octet(0,0,0,0,0,0,0,0)),
                      MakeSignallingField(RealTimeVCDU,
                                          VCDUSpare),
                      VCDUHEC),
                   NullOS,
                   AddFront(Octet(0,0,0,0,0,0,0,1),
                    AddFront(Octet(0,0,0,0,0,0,1,0),
                     AddFront(Octet(0,0,0,0,0,0,1,1),
                      AddFront(Octet(0,0,0,0,0,1,0,0), NullOS)))),
                   NullTrailer,
                   NullOS) ;
     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,1)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,0,0,0),
                      MakeVCID(1,1,1,1,1,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,0,1)) ;
```

```
man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,0,1))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     release ! Add(1, NullLINKID)
             ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                          MakeVCID(0,0,0,0,1,0)) ;

     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,0))
         ! True  (* HEC Required *)
         ! True  (* Trailer Present *)
         ! True  (* VCDUErrorControl Required *)
         ! True  (* Operational Control Field Present *)
         ! True  (* Reed Solomon Required *)
         ! 2     (* Reed Solomon Length *) ;

     success ;
     exit

)

endproc test

endproc vcapt8
```

Virtual Channel Access Protocol Test 9

Name: Inappropriate VCDU-ID.

Description: Each VCDU-ID is associated with an upper layer service interface [5.5.1.1.c]. Use of the VCDU service should not be allowed on a VCDU-ID which is associated with the VCA service.

Inputs: A correctly formatted VCA_VCDU.request on VCDU-ID X.

Configuration: VCDU-ID X should be associated with the VCA service.

Expected Results: This input should result in an error condition.

```
process vcapt9 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                      MakeVCID(0,0,0,0,1,1))
         ! OutGoing
         ! Succ(Succ(8))
         ! VCALayer
         ! NullLINKIDList ;

     (
          (
          vcdu ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                            MakeVCID(0,0,0,0,1,1))
               ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                                      MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                                 MakeVCID(0,0,0,0,1,1)),
                                      MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                      Octet(0,0,0,0,0,0,0,0),
                                                      Octet(0,0,0,0,0,0,0,0)),
                                      MakeSignallingField(RealTimeVCDU,
                                                          VCDUSpare),
                                      VCDUHEC),
                         NullOS,
                         AddFront(Octet(0,0,0,0,0,0,0,1), NUllOS),
                         NullTrailer,
                         NullOS) ;
          failure ;
          exit
          )
     []
          (
          success ;
          exit
          )
     )
)

endproc test

endproc vcapt9
```

Virtual Channel Access Protocol Test 10

Name: Unknown VCDU-ID.

Description: A VCDU-ID forms part of the VCA_VCDU.request.

Inputs: A correctly formatted VCA_VCDU.request using VCDU-ID X.

Configuration: Any Management Information which excludes VCDU-ID X.

Expected Results: This input should result in an error condition.

```
process vcapt10 : noexit :=
hide vca, vcdu, man, slap, int in
(
     test [vca, vcdu, man, slap, int]
     |[vca, vcdu, man, slap, int]|
     VirtualChannelProcedures [vca, vcdu, man, slap, int]
)

where

process test [vca, vcdu, man, slap, int] : exit :=

hide success, failure in
(
     man ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,1),
                      MakeVCID(0,0,0,0,1,1))
         ! OutGoing
         ! Succ(Succ(8))
         ! VCDULayer
         ! NullLINKIDList ;

     (
          (
          vcdu ! MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                            MakeVCID(0,0,0,0,1,1))
               ! MakeVCDU(MakeVCDUPrimaryHeader(Version2,
                                        MakeVCDUID(MakeSCID(0,0,0,0,0,1,1,0),
                                                   MakeVCID(0,0,0,0,1,1)),
                                        MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                        Octet(0,0,0,0,0,0,0,0),
                                                        Octet(0,0,0,0,0,0,0,0)),
                                        MakeSignallingField(RealTimeVCDU,
                                                            VCDUSpare),
                                        VCDUHEC),
                         NullOS,
                         AddFront(Octet(0,0,0,0,0,0,0,1),
                          AddFront(Octet(0,0,0,0,0,0,1,0), NullOS)),
                         NullTrailer,
                         NullOS) ;
          failure ;
          exit
          )
     []
          (
          success ;
          exit
          )
     )
)

endproc test

endproc vcapt10
```

Virtual Channel Access Service Test 1

Name: Data Transfer

Description: The Virtual Channel Access Service is used to transfer VCA_SDUs across the Space Link Subnetwork.

Inputs: A string of VCA_UNITDATA.requests with known data content.

Configuration: No specific configuration is required.

Expected Results: A string of VCA_UNITDATA.indications with SDUs containing the same known data as input.

```
process vcast1 (GenerateInsertLossFlag : Bool) : noexit :=
hide vca, vcdu, insert, man, release in
(
     test [vca, vcdu, insert, man, release]
     |[vca, vcdu, insert, man, release]|
     VCAService [vca, vcdu, insert, man, release]
                (False)
)

where

process test [vca, vcdu, insert, man, release] : exit :=

hide success, failure in
(

man ! Add(1, NullLINKID)
    ! False
    ! 2
    ! False ;

man ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
    ! VCALayer
    ! 8
    ! False
    ! False
    ! Add(Add(1, NullLINKID), NullLINKIDList) ;
     (
          (
          vca ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
              ! AddFront(Octet(0,0,0,0,0,0,0,0),
                 AddFront(Octet(0,0,0,0,0,0,0,1),
                  AddFront(Octet(0,0,0,0,0,0,1,0),
                   AddFront(Octet(0,0,0,0,0,0,1,1),
                    AddFront(Octet(0,0,0,0,0,1,0,0),
                     AddFront(Octet(0,0,0,0,0,1,0,1),
                      AddFront(Octet(0,0,0,0,0,1,1,0),
                       AddFront(Octet(0,0,0,0,0,1,1,1), nullOS)))))))) ;

          release ! Add(1, NullLINKID)
                  ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1),
                    MakeVCID(0,1,0,0,1,1)) ;

          vca ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
              ? VCASDU : OctetString
                [VCASDU Eq
                 AddFront(Octet(0,0,0,0,0,0,0,0),
                  AddFront(Octet(0,0,0,0,0,0,0,1),
                   AddFront(Octet(0,0,0,0,0,0,1,0),
                    AddFront(Octet(0,0,0,0,0,0,1,1),
                     AddFront(Octet(0,0,0,0,0,1,0,0),
                      AddFront(Octet(0,0,0,0,0,1,0,1),
                       AddFront(Octet(0,0,0,0,0,1,1,0),
                        AddFront(Octet(0,0,0,0,0,1,1,1), nullOS))))))))] ;
          success ; exit
          )
          []
          (
          failure ; exit
          )
     )
)
endproc test

endproc vcast1
```

<u>Virtual Channel Access Service Test 2</u>

Name: Data Transfer

Description: The Virtual Channel Access Service is used to transfer IN_SDUs across the Space Link Subnetwork.

Inputs: A release event, without any previous VCA_UNITDATA.requests

Configuration: No specific configuration is required.

Expected Results: An INSERT indication containing the same known data as input.

```
process vcast2 (GenerateInsertLossFlag : Bool) : noexit :=
hide vca, vcdu, insert, man, release in
(
     test [vca, vcdu, insert, man, release]
     |[vca, vcdu, insert, man, release]|
     VCAService [vca, vcdu, insert, man, release]
                (False)
)

where

process test [vca, vcdu, insert, man, release] : exit :=

hide success, failure in
(

man ! Add(1, NullLINKID)
    ! True
    ! 2
    ! False ;

     (
          (
          release ! Add(1, NullLINKID)
                  ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1),
                    MakeVCID(0,1,0,0,1,1)) ;

          insert ! Add(1, NullLINKID)
                 ! AddFront(Octet(0,1,0,1,0,1,0,1),
                    AddFront(Octet(1,0,1,0,1,0,1,0), NullOS)) ;

          insert ! Add(1, NullLINKID)
                 ? INSDU : OctetString
                 [INSDU Eq
                  AddFront(Octet(0,1,0,1,0,1,0,1),
                    AddFront(Octet(1,0,1,0,1,0,1,0), NullOS))] ;

          success ; exit
          )
          []
          (
          failure ; exit
          )
     )
)
endproc test

endproc vcast2
```

Virtual Channel Access Service Test 3

Name: Data Transfer

Description: The Virtual Channel Access Service is used to transfer VCDUs across the Space Link Subnetwork.

Inputs: A release event, without any previous VCA_VCDU.requests

Configuration: No specific configuration is required.

Expected Results: A VCA_VCDU.indication containing the same known data as input.

```
process vcast3 (GenerateInsertLossFlag : Bool) : noexit :=
hide vca, vcdu, insert, man, release in
(
     test [vca, vcdu, insert, man, release]
     |[vca, vcdu, insert, man, release]|
     VCAService [vca, vcdu, insert, man, release]
                (False)
)

where

process test [vca, vcdu, insert, man, release] : exit :=

hide success, failure in
(

man ! Add(1, NullLINKID)
    ! False
    ! 2
    ! False ;

man ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
    ! VCDULayer
    ! 8
    ! False
    ! False
    ! Add(Add(1, NullLINKID), NullLINKIDList) ;
     (
          (
          vcdu ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
               ! MakeVCDU(MakeVCDUPrimaryHeader
                                    (Version2,
                                     MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1),
                                                MakeVCID(0,1,0,0,1,1)) ,
                                     MakeVCDUCounter(Octet(0,0,0,0,0,0,0,0),
                                                     Octet(0,0,0,0,0,0,0,0),
                                                     Octet(0,0,0,0,0,0,0,0)),
                                     MakeSignallingField(RealTimeVCDU,
                                                         VCDUSpare),
                                     NoVCDUHEC),
                              NullOS,
                              AddFront(Octet(0,1,0,1,0,1,0,1),
                               AddFront(Octet(1,0,1,0,1,0,1,0), NullOS)),
                              NullTrailer,
                              NullOS) ;

          release ! Add(1, NullLINKID)
                  ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1),
                    MakeVCID(0,1,0,0,1,1)) ;

          vcdu ! MakeVCDUID(MakeSCID(0,0,0,0,1,0,1,1), MakeVCID(0,1,0,0,1,1))
               ? VCDU : VCDU
                 [GetVCASDU(VCDU) Eq
                  AddFront(Octet(0,1,0,1,0,1,0,1),
                   AddFront(Octet(1,0,1,0,1,0,1,0), NullOS))] ;

          success ; exit
          )
          []
          (
          failure ; exit
          )
     )
)
endproc test

endproc vcast3
```