PROTOCOLS SPECIFICATION AND VALIDATION FOR THE MOVEMENT OF GRADES 2 AND 3 BITSTREAM DATA THROUGH THE VIRTUAL CHANNEL LINK CONTROL LAYER OF THE RETURN LINK OF THE CCSDS PRINCIPAL NETWORK (CPN)

By

Quoc The Nguyen

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

APPROVED:

[illegible]
Frederick J. Ricci

[illegible]
Daniel Schaefer

[illegible]
Burgess Allison

May, 1989

Blacksburg, Virginia

PROTOCOLS SPECIFICATION AND VALIDATION

FOR THE MOVEMENT OF GRADES 2 AND 3 BITSTREAM DATA

THROUGH

THE VIRTUAL CHANNEL LINK CONTROL LAYER OF THE RETURN LINK

OF THE CCSDS PRINCIPAL NETWORK (CPN)

by

Quoc The Nguyen

Committee Chairman: Frederick J. Ricci

Electrical Engineering

(ABSTRACT)

Specification of data communication protocols requirements requires a formal approach to ensure that the requirements are correctly and unambiguously specified. This research examines a proposed protocols specification for the movement of bitstream data through space segment by applying a formal definition technique known as the Language of Temporal Ordering System (LOTOS). Successful generation of the LOTOS specification to detail sequence of events and their internal structures in an implementation independent manner clarifies the requirements and provides a framework from which possible cases or events in each process can be tested.

In addition, a LOTOS software tool called HIPPO is used in the research. HIPPO identifies any deadlock that could happen in the protocols and allows sequence of events to be interactively simulated to ascertain of the specification consistency.

ACKNOWLEDGEMENTS

The author wishes to acknowledge The MITRE Corporation for its contribution and assistance. Specifically, the author wishes to acknowledge the contributions of Mr. Burgess Allison for providing his guidance throughout the entire research. The author also wishes to acknowlege the contributions of Mr. James Noles and Mr. Kent Lockhart for their assistance in providing and verifying the protocols concepts that this research specifies.

Finally, the author would like to acknowledge the support of Dr. Fred Ricci and Dr. Daniel Schaefer for being supportive and encouraging during the pursuit of the degree and the development of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

1.0 BACKGROUND

The National Aeronautics and Space Administration (NASA) is developing the Space Station Information System (SSIS) to provide data storage, processing, and communications facilities for the operation of the Space Station Manned Base, platforms, and ground support systems. A Network Services Reference Configuration has been developed to document a logical configuration of the SSIS network with the emphasis on the logical functions required for supplying network services.

Figure 1-1 gives an overview of the SSIS functions. The SSIS network consists of the Data Management System (DMS) and the Commmunications and Tracking (C&T) on the space segment. On the ground segment, the SSIS consists of the Data Interface Facility (DIF), Virtual Channel Gateway (VCGW), and Ground Distribution.

The SSIS functions are further divided into two levels: a return and a forward link level. The return link level consists of SSIS components that facilitate the movement of data from the space segment to the ground segment. These components are shown in Figure 1-2. The forward link level specifies movement of data from the ground segment to the space segment. The components of the forward link level are shown in Figure 1-3.

The Consultative Committee for Space Data Systems (CCSDS), which is established by the management of member space agencies, is responsible for formulating standards and recommendations which address common data

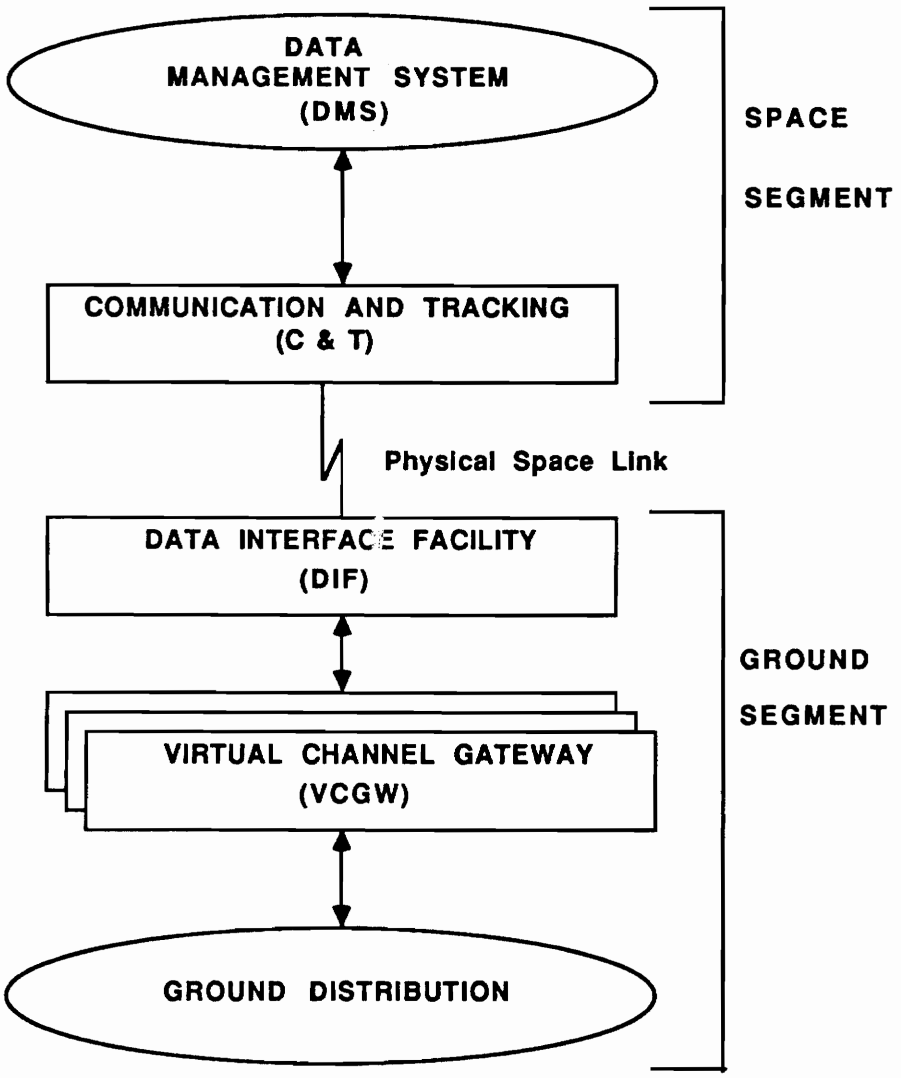

DATA MANAGEMENT SYSTEM (DMS)
COMMUNICATION AND TRACKING (C & T)
Physical Space Link
DATA INTERFACE FACILITY (DIF)
VIRTUAL CHANNEL GATEWAY (VCGW)
GROUND DISTRIBUTION
SPACE SEGMENT
GROUND SEGMENT

Figure 1-1 SSIS Functions

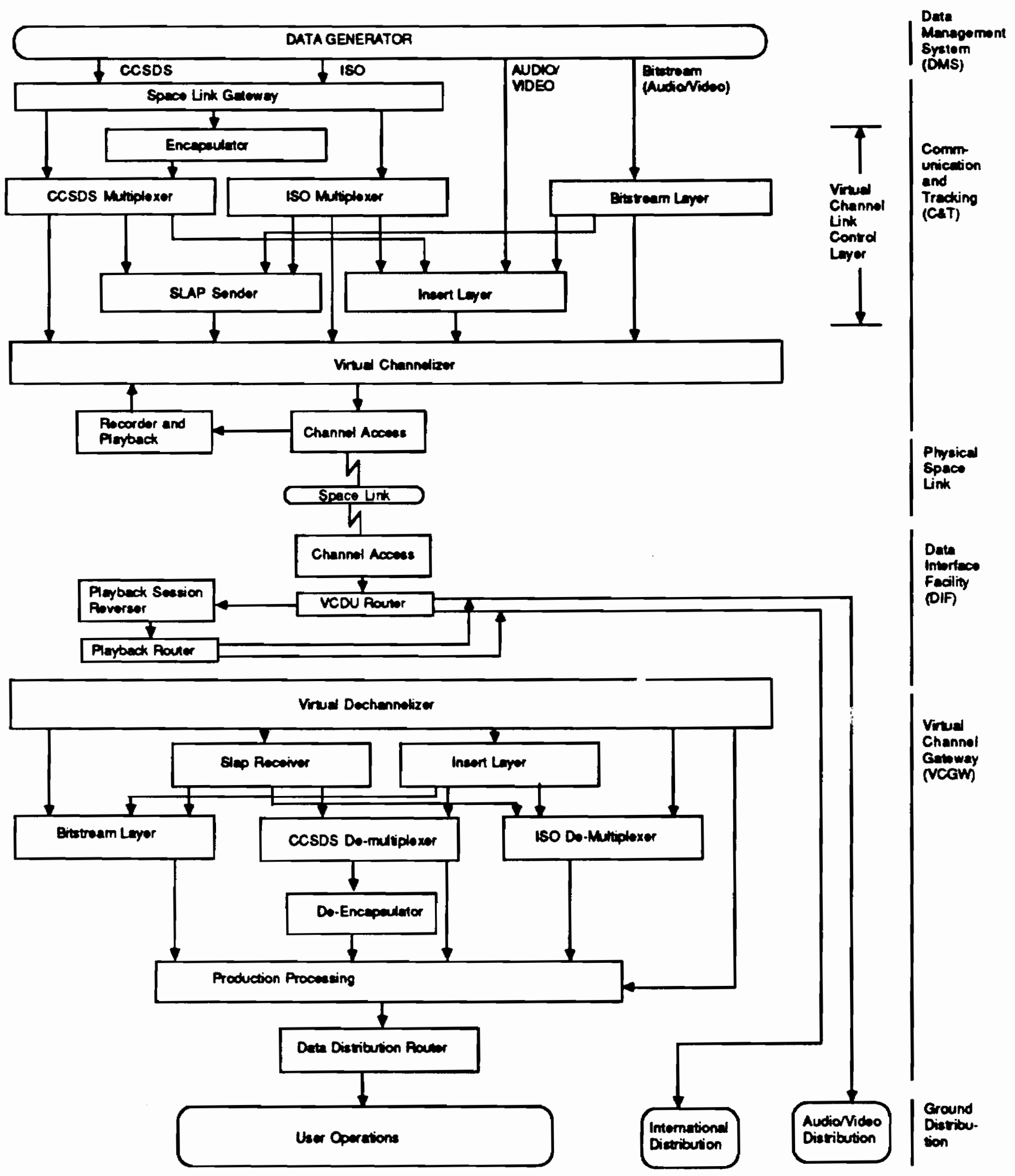


Figure 1-2 SSIS Return Link Components

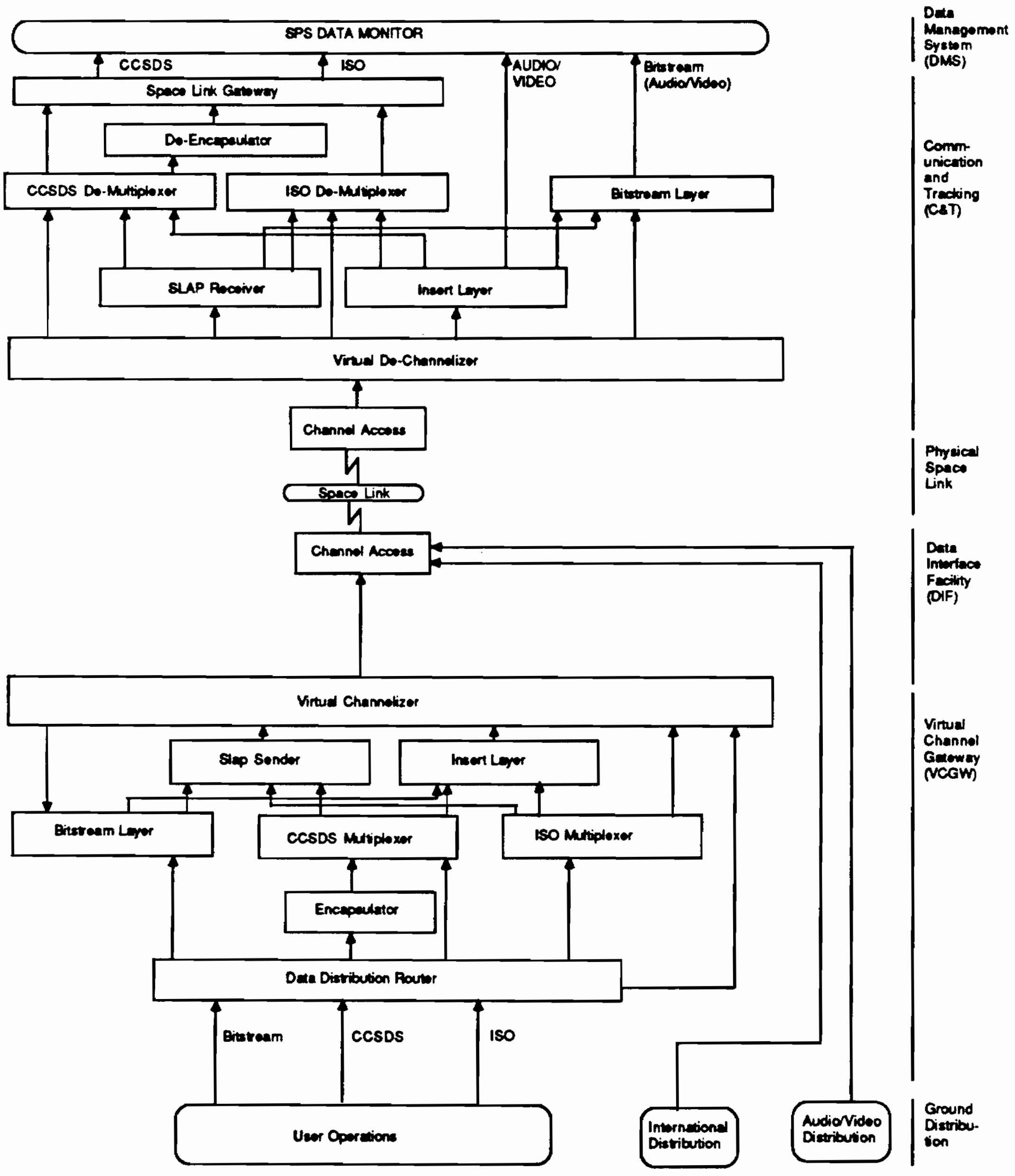


Figure 1-3 SSIS Forward Link Components

systems problems. The member agencies of the CCSDS include:

- British National Space Centre (BNSC)/United Kingdom.
- Centre National D'Etudes Spatiales (CNES)/France
- Deutsche Forschungs-u Versuchsanstalt fuer Luft und Raumfahrt e.V (DFVLR)/West Germany.
- European Space Agency (ESA)/Europe.
- Indian Space Research Organization (ISRO)/India.
- Insitituto de Pesquisas Espacialis (INPE)/Brazil.
- National Aeronautics and Space Administration (NASA)/USA.
- National Space Development Agency of Japan (NASDA)/Japan.
- Chinese Academy of Space Technology (CAST)/People's Republic of China.
- Department of Communications, Communications Research Centre (DOC-CRC)/Canada.

The CCSDS is currently examining the Network and Data Link Architecture for the SSIS. Specifically, the CCSDS is developing a set of protocols which facilitates the movement of data throughout the SSIS Space Link Network (SLN). The CCSDS Architecture of the SSIS Network is slightly different from the SSIS Network Architecture. This research will primarily deal with the CCSDS Architecture.

2.0 SCOPE

This thesis will examine the procedures required to move Grade 2 and 3 Bitstream data through the Virtual Channel Link Control (VCLC) layer of the space segment of the Return Link path as shown in Figure 1-2. As a result, only the Bitstream Service and the Insert Service will be addressed in this research. For each service, the research specifies the exposed service interfaces at which various services can be obtained, the corresponding interface primitives to be implemented to access them, their associated data structures, and the services to be provided. The protocols will be specified in both narrative and in Language of Temporal Ordering System (LOTOS) formats for Bitstream data movement from a Space Bitstream Data Source (e.g., Bitstream Data Generator) to the Virtual Channelizer. LOTOS is a Formal Description Technique (FDT) which provides a mathematical model that is implementation independent and suitable for the design, analysis, and specification of information processing systems. LOTOS defines the behavior of a system in terms of processes in a language with a formal syntaxs and semantics, instead of natural language such as English.

3.0 OBJECTIVES

This thesis aims to accomplish the following objectives:

- To specify the communication protocols and data format for the bitstream data using mathematical modeling techniques from the Language of Temporal Ordering System (LOTOS). Successful completion of this process serves as a validation for the correctness of the space link protocols requirements.
- To validate the completeness and accuracy of the specification against the services (or functions) to be provided by the bitstream service. A software tool called HIPPO will be used to assist in this validation process.

4.0 DOCUMENT ORGANIZATION

<u>Section 1 - Background</u>: provides necessary information to familiarize the reader with the Space Station networking environment.

<u>Section 2 - Scope</u>: discusses the thesis area of research.

<u>Section 3 - Objectives</u>: gives the objectives to be accomplished by the research.

<u>Section 4 - Document Organization</u>: provides a briefing summary of the content of each section in the document.

<u>Section 5 - Overview and General Definition</u>: provides relative overview information and definitions to clarify terms that are used throughout the document.

<u>Section 6 - Bitstream Procedures</u>: discusses the protocols for the Bitstream Procedures process.

<u>Section 7 - Insert Procedures</u>: provides the protocols for the Insert Procedure process.

<u>Section 8 - LOTOS Specification</u>: provides the LOTOS specification for both the Bitstream Service and the Insert Service.

<u>Section 9 - HIPPO</u>: documents the result of the validation of the protocols specification using HIPPO as a simulation tool.

<u>Section 10 - Conclusions</u>: summarizes the research results and makes recommendations for areas which need further research if applicable.

Appendix A: gives detailed LOTOS specification for both Bitstream and Insert procedures.

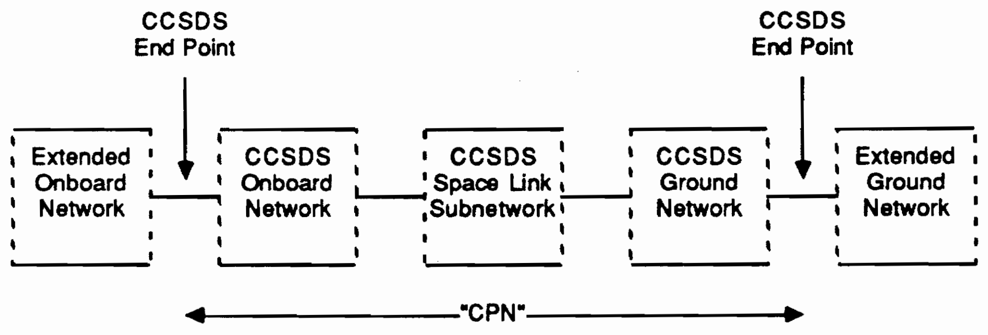
Appendix B: gives the symbolic execution result of the LOTOS specification for both Bitstream and Insert procedure.

5.0 OVERVIEW AND GENERAL DEFINITIONS

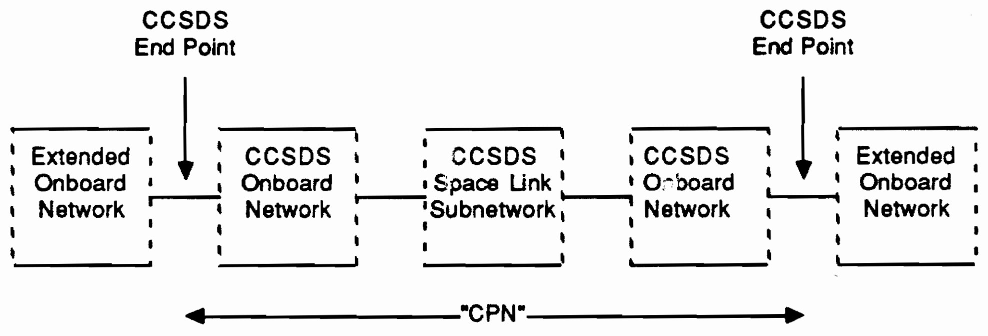
5.1 CCSDS Principal Network (CPN) Overview.

The CCSDS Principal Network (CPN) serves as, or is embedded within, the data handling networks which provide end-to-end data flow in support of space mission users. A CPN consists of an "Onboard Network" in an orbiting segment connected via a CCSDS "Space Link Subnetwork" either to a "Ground Network", or to an Onboard Network in another orbiting segment. Figure 5-1 provides the conceptual view of a CPN in both a space-to-ground and a space-to-space configuration.

The CCSDS is trying to formulate a suite of services and protocols operating within the Space Link Subnetwork (SLS) to support various unique data types being transferred through space data channels. Specifically, the CCSDS seeks to provide a communication systems architecture that facilitates opportunities for inter-Agency "cross-support", which is defined as the capability for one space Agency to bidirectionally transfer another Agency's data between ground and space systems, using its own transmission resources. Cross support takes place at the level of data structures that are created by one Agency and handed over to another Agency for transmission. This is achieved by defining standard service interface to the communications infrastructure which is provided by the cooperating Agencies. The address of such service interfaces on the CPN are called "Service Access Points" (SAPs). Data coming to a SAP is called a Service Data Unit (SDU). Some Protocol



(A) Space-to-Ground Configuration



(B) Space-to-Space Configuration

Figure 5-1 CCSDS Principal Network Elements

Control Information (PCI) may or may not be added at the SAP. The resulting data coming out of a SAP and going to a service (or process) is called a Protocol Data Unit (PDU). A "Data Unit" may consist of a data field and a header field, which gives identification information.

A key feature of the CCSDS protocols provided for the SLS is the concept of Virtual Channel (VC). The VC facility allows one physical space channel to be shared among multiple higher layer traffic streams, each of which may have different service requirements. A single physical space channel may have different service requirements. A single physical space channel may therefore be divided into several separate logical data channels, each known as a "Virtual Channel". Each VC is individually identified by a Virtual Channel Data Unit Identifier (VCDU-ID) and each VCDU-ID supports a single "Grade of Service". The Grade of Service concept will be discussed in the following paragraphs.

5.2 Architecture of the Space Link Subnetwork (SLS)

A conceptual model for the Bitstream data movement from the Data Management System (DMS) to the Physical Space Link is provided in Figure 5-2. The Bitstream data source is modeled as the Bitstream Data Generator. Bitstream data are serial strings of bits, whose internal structure and boundaries are unknown to the CPN. The Bitstream data is transferred across the SLS. The transfer is sequence preserving and may be either "asynchronous" or "isochronous". Isochronous transfer

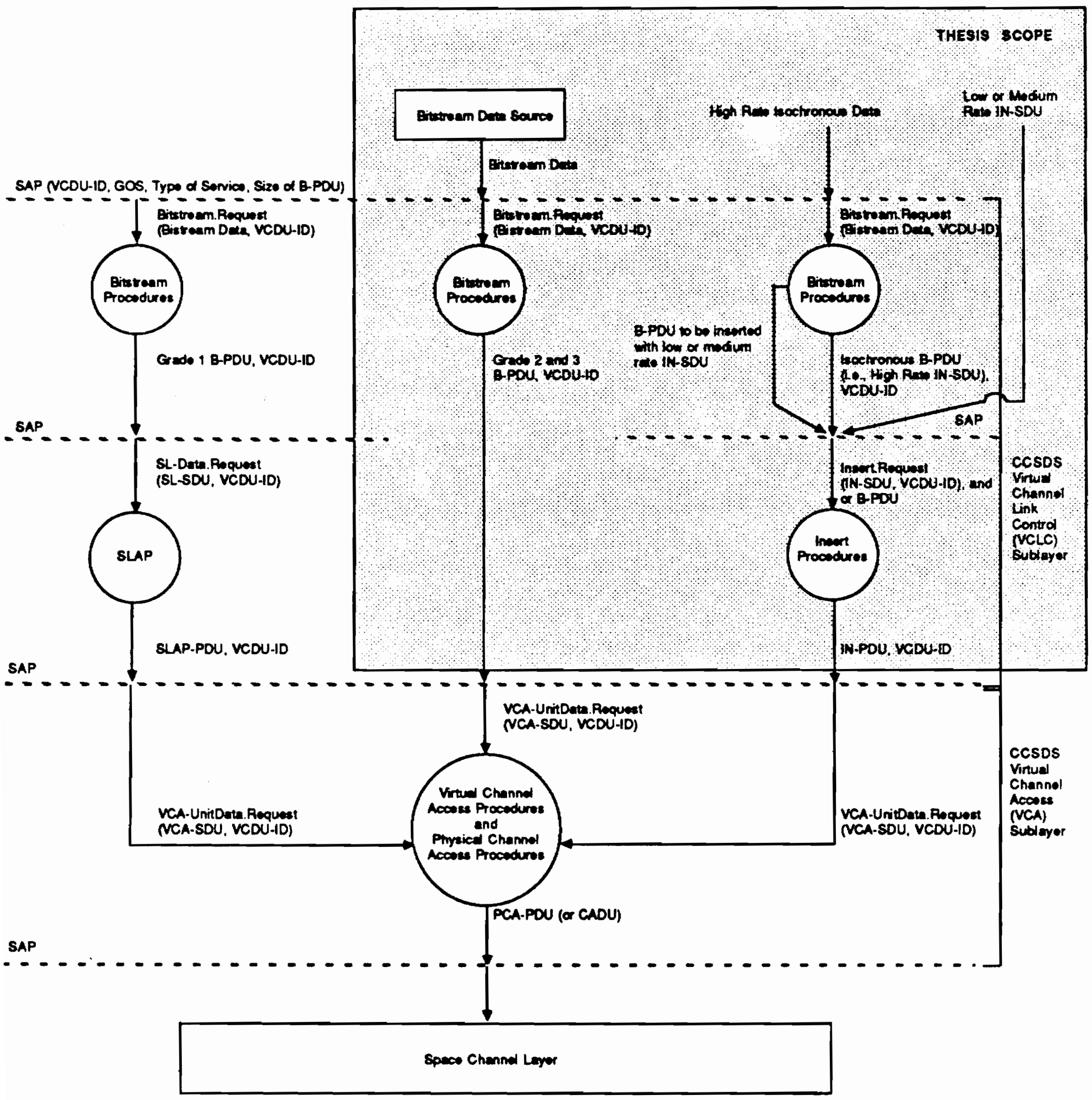

THESIS SCOPE
Bitstream Data Source
High Rate Isochronous Data
Low or Medium Rate IN-SDU
Bitstream Data
SAP (VCDU-ID, GOS, Type of Service, Size of B-PDU)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream Procedures
Bitstream Procedures
Bitstream Procedures
Grade 1 B-PDU, VCDU-ID
Grade 2 and 3 B-PDU, VCDU-ID
B-PDU to be inserted with low or medium rate IN-SDU
Isochronous B-PDU (i.e., High Rate IN-SDU), VCDU-ID
SAP
SAP
SL-Data.Request (SL-SDU, VCDU-ID)
Insert.Request (IN-SDU, VCDU-ID), and/or B-PDU
CCSDS Virtual Channel Link Control (VCLC) Sublayer
SLAP
Insert Procedures
SLAP-PDU, VCDU-ID
IN-PDU, VCDU-ID
SAP
VCA-UnitData.Request (VCA-SDU, VCDU-ID)
VCA-UnitData.Request (VCA-SDU, VCDU-ID)
VCA-UnitData.Request (VCA-SDU, VCDU-ID)
Virtual Channel Access Procedures and Physical Channel Access Procedures
CCSDS Virtual Channel Access (VCA) Sublayer
PCA-PDU (or CADU)
SAP
Space Channel Layer

Figure 5-2 Bitstream Data Flow Diagram - High Level

from service interface to service interface is provided with a specified maximum delay and a specified maximum jitter at the service interface. High rate video data may use the isochronous Bitstream service.

Data to be transferred across the SLS are classified by the Grade of Service (GOS). Three GOSs are offered within the SLS:

- Grade-1: SLS service data units are delivered through the SLS complete, with their sequence preserved, and with a very high probability of containing no errors induced by the SLS.
- Grade-2: SLS service data units are delivered through the SLS possibly incomplete, but with their sequence preserved and with a very high probability of containing no errors induced in the SLS.
- Grade-3: SLS service data units are delivered through the SLS possibly incomplete with a moderate probability that they contain errors induced by the SLS, but with their sequence preserved.

The GOS are not explicitly signaled by protocol, but are set up by CPN Management procedures. This research will only address the data movement procedures for Grade 2 and Grade 3 Bitstream data.

In Figure 5-2 a BITSTREAM.Request primitive is generated by the receiving layer upon receiving the Bitstream from the Bitstream data source. The BITSTREAM.Request primitive is passed to the Virtual Channel Link Control sublayer to request that Bitstream data be sent.

The BITSTREAM.Request contains the VCDU-ID and the Bitstream data.

Assuming that the VC is enabled by the CPN management and data are allowed to be transferred in the VC, the Bitstream data will be moved to the Bitstream Procedure which breaks Bitstream data into fixed and pre-determined Bitstream PDU (B-PDU) depending on the GOS and the type of service (i.e., the length of the B-PDU for regular Bitstream service is different from the length of the B-PDU for the Insert Service). Depending on the GOS and the type of service required, B-PDUs coming out from the Bitstream procedure are either routed to the SLAP procedure, the VCA procedure, or the Insert procedure.

Data coming out from the SLAP procedure are called SLAP-PDU. Similarly, data coming out from the Insert procedure are called IN-PDU. These data will be submitted to the VCA procedure, which is a part of the Virtual Channelizer, via the VCA-UnitData.Request primitive. The VCA-UnitData.Request primitive consists of a VCA-SDU and the VCDU-ID. The VCA-SDU could be a SLAP-PDU (i.e. containing Grade 1 Bitstream data), an IN-PDU (i.e. containing high rate, isochronous Bitstream data), or a Grade 2 or 3 B-PDU. Data coming out from the VCA procedure are called Physical Control Access PDU (PCA-PDU). This is also known as the Channel Access Data Unit (CADU). The CADU is routed to the Space Channel Layer which facilitates the transmission of data from space to ground or from space to space.

For each Bitstream data source a dedicated VC is pre-configured by the Network Management (i.e., each VC is dedicated to one source of

Bitstream data). A Virtual Channel Data Unit Identification (VCDU-ID) consisting of a Spacecraft Identifier(SCID) and a Virtual Channel Identifier (VCID) is assigned to a VC to uniquely identify the spacecraft with which the VCDU is associated, and to identify the VC in use. One spacecraft may have multiple SCIDs assigned to it. The format of the VCDU-ID is discussed in details in the later sections.

There are two possible ways to implement a dedicated VC. The first method is to have a one-to-one connection between a Bitstream data source and a dedicated receiving port (i.e., VC). The receiving port is assigned a fixed VCDU-ID which uniquely supports that particular Bitstream data source. As a result, this particular port will only support one type of service dictated by one particular VCDU-ID only. This implementation is deemed to be rigid since it does not allow CPN Management to change the configuration of the port dynamically in the future to support different types of service. The second implementation approach is to connect the bitstream data source to an arbitrary port from the receiving end. A management look up table can be established to assign a particular configuration to a port. As a result, to change a configuration of a port CPN Management only has to change the port ID in the look up table for a particular configuration. This implementation approach is considered to be more robust since it allows flexibility in VC configuration management. Figures 5-3 and 5-4 detail these two implementation approaches.

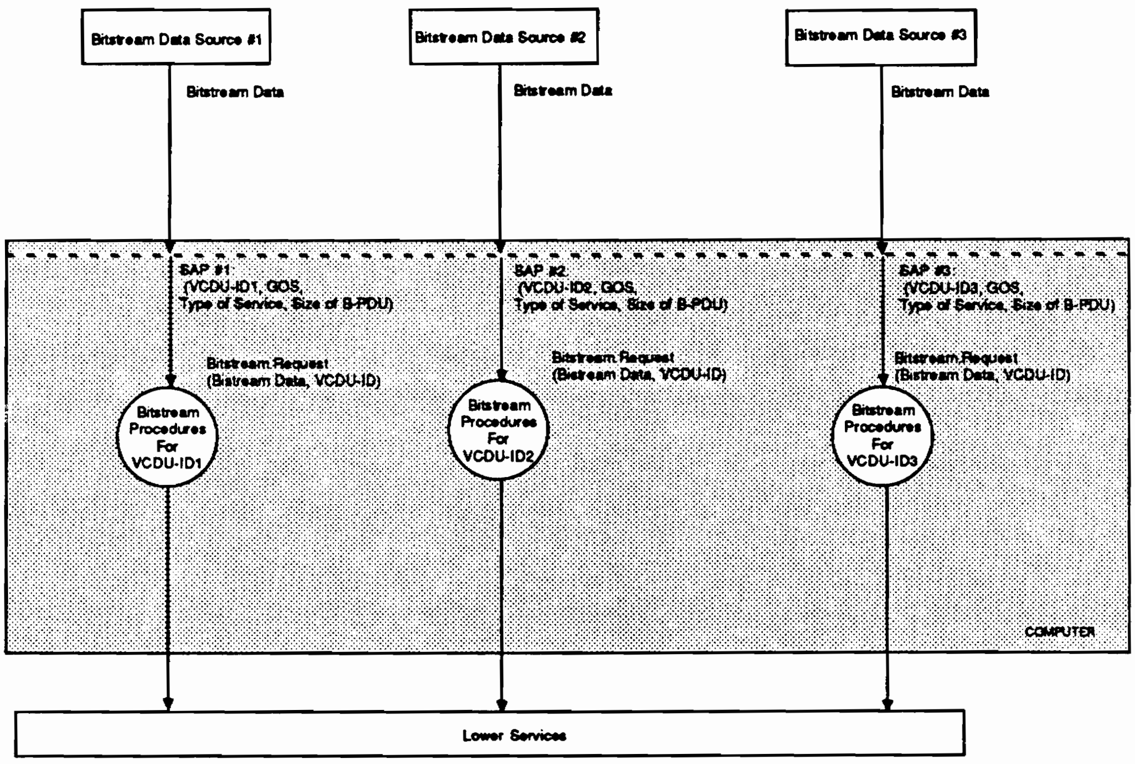

Bitstream Data Source #1
Bitstream Data Source #2
Bitstream Data Source #3
Bitstream Data
Bitstream Data
Bitstream Data
SAP #1: (VCDU-ID1, GOS, Type of Service, Size of B-PDU)
SAP #2: (VCDU-ID2, GOS, Type of Service, Size of B-PDU)
SAP #3: (VCDU-ID3, GOS, Type of Service, Size of B-PDU)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream.Request (Bitstream Data, VCDU-ID)
Bitstream Procedures For VCDU-ID1
Bitstream Procedures For VCDU-ID2
Bitstream Procedures For VCDU-ID3
COMPUTER
Lower Services

Figure 5-3 One-to-One Implementation of Virtual Channel

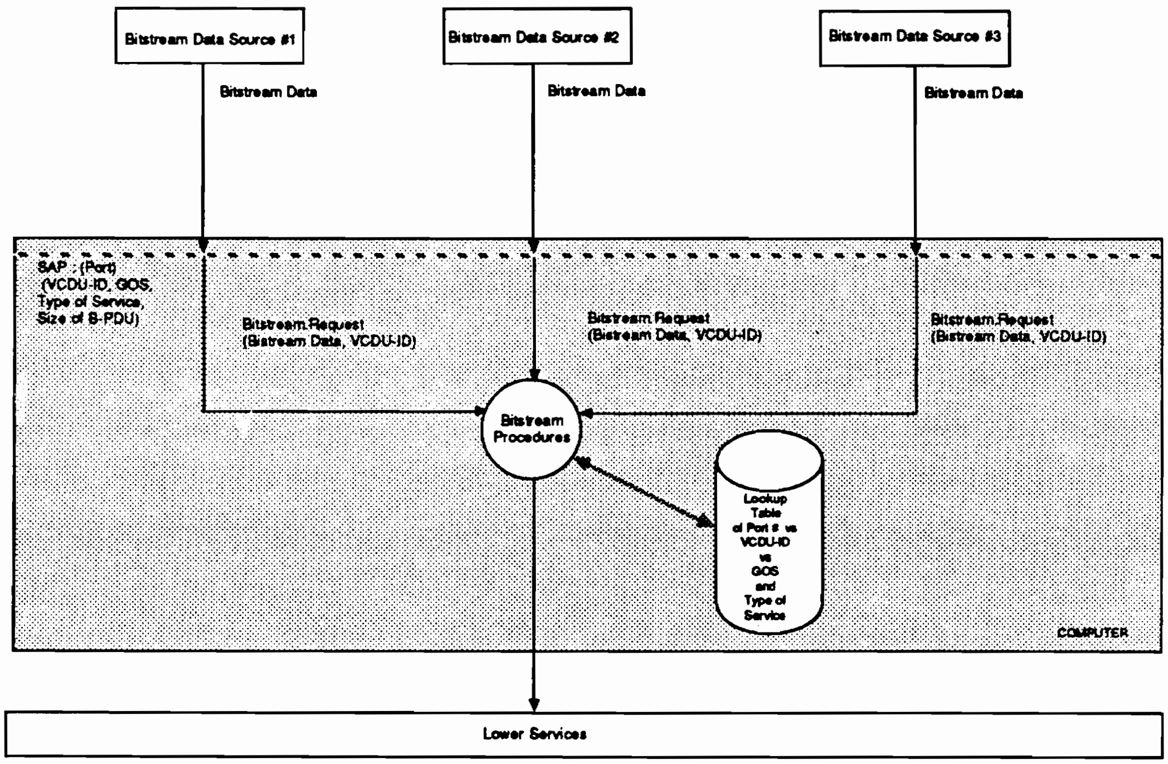

Bitstream Data Source #1
Bitstream Data Source #2
Bitstream Data Source #3
Bitstream Data
Bitstream Data
Bitstream Data
SAP : (Port)
(VCDU-ID, GOS, Type of Service, Size of B-PDU)
Bitstream.Request (Bistream Data, VCDU-ID)
Bitstream.Request (Bistream Data, VCDU-ID)
Bitstream.Request (Bistream Data, VCDU-ID)
Bitstream Procedures
Lookup Table of Port # vs VCDU-ID vs GOS and Type of Service
COMPUTER
Lower Services

Figure 5-4 Table Lookup Implementation of Virtual Channel

5.3 Validation

As used in this research, "Validation" of the protocols means that the protocols' requirements are reviewed, analyzed, specified in an unambiguous and internally consistent manner, and tested under specific cases for their logics, their internal structure of sequences of events, and their completeness and accuracy within the cases considered. "Validation" of the protocols, as used in this research, does not mean that an exhaustive independent validation and verification of all possible implementation approaches and test cases has been performed. As a result, the "Validation" of this research only pertains to the high level requirements specification and not to the detailed design specification.

For this research, the validation process consists of two stages. The first stage is to review and analyze the requirements. Upon completion of this stage, areas which are vague or not specifically clarified in the requirements are identified. A coherent and consistent LOTOS specification is generated as a result of the analysis. The LOTOS specification of the protocols details the procedures containing the sequence of events that satisfy the protocols' requirements. It provides a framework from which possible cases or events in each process can be tested. The second stage of the validation process is to dynamically step through each case specified in the LOTOS specification, using several example cases, to test for its consistency with the requirements and completeness within the specification itself. HIPPO is used for this process. Any deadlock that could happen in the protocols is identified and fixed. The sequence of events are interactively simulated to ascertain of their consistency. As a result, the dynamic behavior of the protocols can be viewed and analyzed. Successful completion of the HIPPO simulation session implies that the protocols as specified contains no deadlock and that each sequence of events specified is complete and consistent with others. The HIPPO simulation process does not provide any insight to any design approach for a system which would satisfy the protocols' requirements.

6.0 BITSTREAM PROCEDURE

Figure 6-1 depicts the Bitstream Procedure in detail. The Bitstream data generated by a Bitstream data source (i.e., Bitstream Data Generator) are sent to a SAP (i.e., the interface point between two services) where VC connection is established. The VCDU-ID is determined by the point of entrance of Bitstream data (i.e., the SAP). Hence, the VCDU-ID gives the address of the SAP. From the VCDU-ID the GOS and the VC to be used for the data from this particular source are determined. Once the VC to be used is determined, the size of the Bitstream Protocol Data Unit (B-PDU) to be supported and the type of service to be used (i.e., Bitstream Service or Insert Service) can be determined. These information are pre-defined and provided by the CPN Management for each VC at the receiving port where the Bitstream data enter.

An implementation approach for the scenario described above is to have a look up table using the VCDU-ID as the key field. The CPN Management stores these information in table format which could be searched by the VCDU-ID. A conceptual model for this implementation approach is provided in Figure 6-2.

Bitstream data generated from a Bitstream source is sent to the Virtual Channel Link Control (VCLC) sublayer via a service primitive called BITSTREAM.Request.

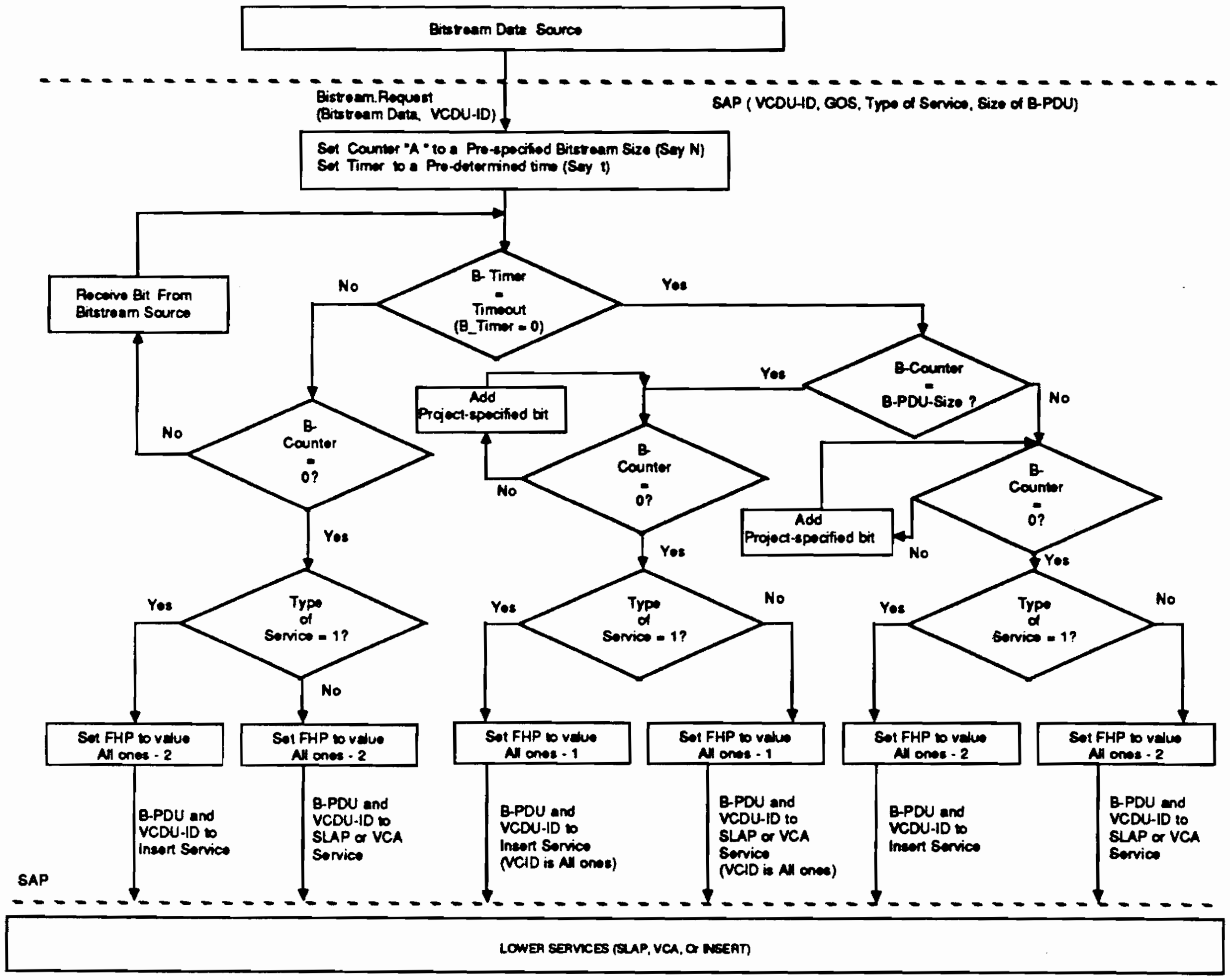

Bitstream Data Source
Bistream.Request (Bitstream Data, VCDU-ID)
SAP (VCDU-ID, GOS, Type of Service, Size of B-PDU)
Set Counter "A" to a Pre-specified Bitstream Size (Say N)
Set Timer to a Pre-determined time (Say t)
Receive Bit From Bitstream Source
B- Timer = Timeout (B_Timer = 0)
No
Yes
B-Counter = B-PDU-Size ?
Yes
No
Add Project-specified bit
B- Counter = 0?
No
Yes
Add Project-specified bit
B- Counter = 0?
No
Yes
B- Counter = 0?
No
Yes
Type of Service = 1?
Yes
No
Type of Service = 1?
Yes
No
Type of Service = 1?
Yes
No
Set FHP to value All ones - 2
Set FHP to value All ones - 2
Set FHP to value All ones - 1
Set FHP to value All ones - 1
Set FHP to value All ones - 2
Set FHP to value All ones - 2
B-PDU and VCDU-ID to Insert Service
B-PDU and VCDU-ID to SLAP or VCA Service
B-PDU and VCDU-ID to Insert Service (VCID is All ones)
B-PDU and VCDU-ID to SLAP or VCA Service (VCID is All ones)
B-PDU and VCDU-ID to Insert Service
B-PDU and VCDU-ID to SLAP or VCA Service
SAP
LOWER SERVICES (SLAP, VCA, Or INSERT)

Figure 6-1 Bitstream Procedure Detailed Description

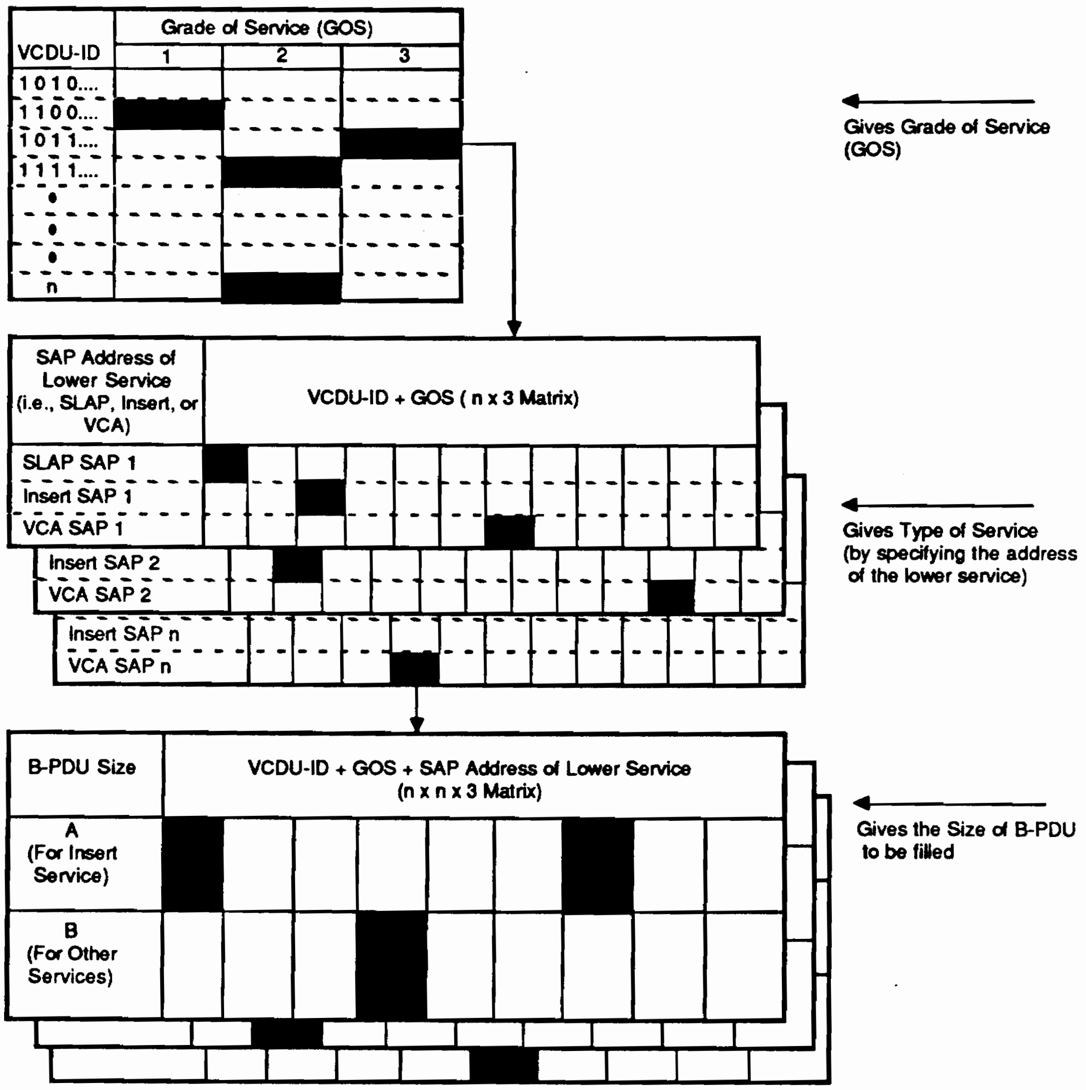

Grade of Service (GOS)
VCDU-ID
1
2
3
1 0 1 0....
1 1 0 0....
1 0 1 1....
1 1 1 1....
n
Gives Grade of Service (GOS)
SAP Address of Lower Service (i.e., SLAP, Insert, or VCA)
VCDU-ID + GOS (n x 3 Matrix)
SLAP SAP 1
Insert SAP 1
VCA SAP 1
Insert SAP 2
VCA SAP 2
Insert SAP n
VCA SAP n
Gives Type of Service (by specifying the address of the lower service)
B-PDU Size
VCDU-ID + GOS + SAP Address of Lower Service (n x n x 3 Matrix)
A (For Insert Service)
B (For Other Services)
Gives the Size of B-PDU to be filled

Figure 6-2 Conceptual Model for Table Lookup of the VCDU-ID vs GOS and Virtual Channel

The BITSTREAM.Request primitive is defined as follows:

a. Function

This primitive is the service request primitive for the Bitstream Service (provided by the Bitstream procedure).

b. Semantics

The primitive contains the following parameters:

BITSTREAM.Request (Bitstream data, VCDU-ID)

c. When generated

This primitive is generated by the receiving "entity" at the SAP and passed to the Virtual Channel Link Control (VCLC) sublayer to request it to send the Bitstream data.

d. Effect on Receipt

Receipt of this primitive causes the VCLC sublayer to attempt to process and send the Bitstream data.

e. Additional Comments

Since the service interface specification is an abstract specification, the implementation of the Bitstream data parameter is not constrained, i.e., it may be continuous Bitstream, delimited Bitstream or individual bits.

A conceptual data flow model for the BITSTREAM.Request service primitive generation is provided in Figure 6-3

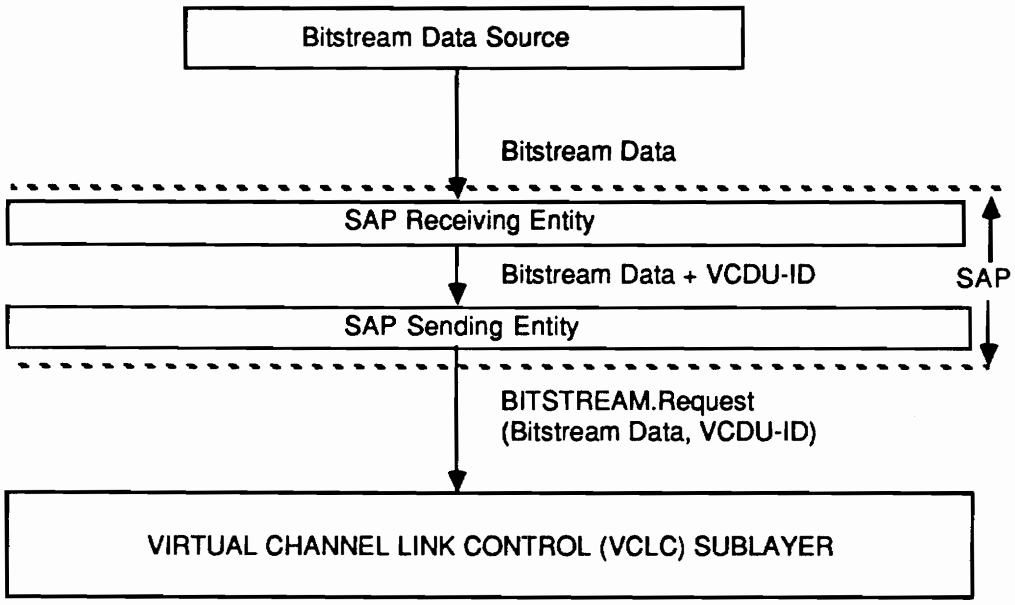


Figure 6-3 Conceptual Data Flow Model for the Bitstream.Request

6.2.1 BITSTREAM.Request Format Specification.

The BITSTREAM.Request contains a VCDU-ID and Bitstream Data. The VCDU-ID identified the SAP to the VCLC (i.e.,the interface point between two services). The VCDU-ID consists of the Spacecraft Identifier (SCID) concatenated with the Virtual Channel Identifier (VCID). A spacecraft may have multiple SCIDs. The format of the VCDU-ID is specified in Figure 6-4.

The SCID is an 8-bit identifier which provides positive identification of the spacecraft. For complex international group of spacecraft, several SCIDs may be present on the same space channel. Different SCIDs will be assigned for flight vehicles, for development vehicles which are using ground networks during prelaunch operations, and for simulated streams. The Secretariat of the CCSDS assigns SCIDs.

The VCID is a 6-bit identifier which enables up to 64 VCs to be run concurrently for each assigned SCID on a particular physical space channel. If only one VC is used, these bits are set permanently to value "all zeros". If a Project wishes to reserve a VC for transmission of "fill" data, this is indicated by setting these bits to value "all ones": a VC so identified may not contain any valid user data.

Upon receiving the BITSTREAM.Request service primitive, the Bitstream procedure provides a "B-PDU Construction Function".

VCDU-ID	
Spacecraft ID (SCID)	Virtual Channel Identifier (VCID)
8 Bits	6 Bits

Figure 6-4 VCDU-ID Format

6.2.2 B-PDU Construction Function.

The B-PDU Construction Function is used to fill the data field of the B-PDU with the Bitstream data supplied in the BITSTREAM.Request primitives. Each B-PDU contains data for only one VC, identified by the VCDU-ID descriptor. Each bit is placed sequentially, and unchanged, into the B-PDU data field.

The B-PDU is fixed length for any VC: its length is specified by the CPN Management to match the data carrying space of the VCA-PDU. When the Bitstream data have filled one particular B-PDU, the continuation of the Bitstream data is placed in the next B-PDU on the same VC (i.e., same VCDU-ID).

There are four scenarios that could happen during the B-PDU Construction Function process:

1. A B-PDU is filled with all valid Bitstream data. This is a normal operation. The B-PDU will then be sent to the lower service procedures.
2. If, due to the constraints of the PDU release algorithm (e.g., a timer is set to release PDUs after a certain time interval), a B-PDU is not completely filled with Bitstream data at release time, the B-PDU Construction Function may fill the remainder of the B-PDU with a Project-specified fill pattern. CCSDS does not currently provide a mechanism for identifying the boundary between the valid user data and fill data. It is currently

assumed then that the receiver at the other end knows which data is valid. It is conceivable that a pointer (or counter) could be set to specify the number of valid Bitstream data bits contained in the B-PDU. This information could be stored in the Bitstream Data Pointer field in the B-PDU header. If such implementation is chosen, 14 bits may be needed to specify the number of valid Bitstream data since the maximum number of bits a B-PDU data field can contain is 10,200.

3. If the Bitstream data source continues to send invalid Bitstream data after it finishes sending all valid Bitstream data, a B-PDU will contain a mix of valid and invalid Bitstream data. In this case, the B-PDU Construction Function does not know that it is receiving invalid data. There are two possible alternatives to address this scenario:
    - The Bitstream data source will have to provide an extra pointer to specify the number of valid Bitstream data and sends it along with the Bitstream data to the Bitstream procedure to be included in the spare field or in the Bitstream Data Pointer (i.e., also called First Header Pointer (FHP)) the B-PDU header.
    - It will be left to the receiver to interpret the received data and assume that the receiver will know which are the valid data.

4. If the Bitstream data source stops sending Bitstream data after sending its last valid bits and the B-PDU is not completely filled by valid Bitstream data, the B-PDU Construction Function could perform similar functions as in Scenario 2. The alternatives in Scenario 3 could also be applied in this case.

After the B-PDUs have been constructed, they will be routed to the SLAP, the Insert procedure, or the VCA procedure depending on the type of service and the GOS required. The service primitive to be used for the interface between the Bitstream procedure and the SLAP is the SL-DATA.Request. The service primitive for the interface between the Bitstream procedure and the VCA procedure is the VCA-UnitData.Request. The service primitive for the interface between the Bitstream procedure and the Insert procedure is the Insert.Request. Format of these primitives will be discussed in each applicable procedure specification in the following subsections.

6.2.3 Format of the B-PDU

Figure 6-5 depicts the format of the B-PDU. The B-PDU Header also known as the VCDU Data Field Status field consists of a Spare field (5 bits) and a Bitstream Data Pointer (also known as the First Header Pointer (FHP))(11 bits). These fields are used as follows:

1. Spare: This field is currently undefined by the CCSDS. It

VCDU Data Field Status		Data Unit Zone
Spare	Bitstream Data Pointer (or First Header Pointer)	Bitstream Data
5 bits	11 bits	Varies
2 octets		Varies

Figure 6-5 B-PDU Format

shall therefore be set to the reserved value "00000". In future, this spare field could have two possible application:

- Use this field to provide an integrated PDU header identifier to discriminate between the Multiplexing PDU (M-PDU), the B-PDU and the IN-PDU, and the possible extension of the Bitstream Data Pointer to provide bit resolution. To specify the type of PDU, a 2-bit code will be adequate.
- Use this field and the Bitstream Data Pointer field to provide a pointer which specify the number of valid Bitstream bits in the B-PDU. This will require 14 bits in total.

2. Bitstream Data Pointer: CCSDS is currently studying the potential of using this pointer to support internal Bitstream Data delimiting (same as item 2 of the spare). At present, it shall be set to the reserved value of "all ones minus two" if any valid user Bitstream data are contained within the B-PDU, or to the reserved value " all ones minus one" if the B-PDU contains only a Project-specified fill pattern. If this Bitstream Data Pointer is used with the Spare field to specify the number of valid bits, the total bits required is 14. Then, there is no need to specified whether the B-PDU contains filled or partially filled valid data. The remaining 2 bits of the VCDU Data Field Status field could be used as identified in #1.

3. Bitstream Data Zone: the Bitstream Data Zone contains either a fixed length block of the user Bitstream data (possibly containing

fill data) or a fixed length Project-specified fill pattern.

Note that a B-PDU must be filled before being transmitted to the lower layer or service.

7.0 INSERT PROCEDURES

7.1 Insert Procedure Overview

Data which are labeled "isochronous" data are transferred through the SLS via the Insert Service. Isochronous data transferring from service interface to service interface is characterized by having maximum delay and a specified maximum jitter at the service interface. High rate video data may use the isochronous Bitstream service.

The Insert service provides a facility for fixed length, octet aligned isochronous SDUs to be transferred across the SLS in a mode which efficiently utilizes the space channel transmission resource. The service is sequence preserving and is provided with a specified maximum delay and a specified maximum jitter at the service interface.

The Insert data usually consist of small samples, inserted into SDUs (i.e., B-PDUs) whose fixed length is pre-determined by the CPN Management. These Insert data are put into an "Insert Zone" which allows them to share the data unit zone of the VCDU with B-PDUs (e.g., for low and medium rate insert data), or to occupy the entire unit zone (e.g., for high rate insert data). The procedure for handling the high rate insert data may be considered as a special case of the Bitstream Procedure (i.e., the outcome is called IN_PDU instead of B_PDU). As a result, it will not be considered as the nucleus of the Insert Service. The Insert Service Data Unit is equal in length at both the return and forward service interfaces. The length of the Insert "packet" must be constant so that the data-carrying space available for other types of SDUs (i.e., B-PDUs) is known.

Since the Insert service shares the same VCs as the other services (i.e., Bitstream service), the SAP is addressed by the VCDU-ID together with a parameter which distinguishes the Insert SDU from other SDUs. This parameter is called the "Service Qualifier". The Service Qualifier is not a service parameter since it is inherent in the name of the service. Therefore, the SAP for the Bitstream service will be identified by the VCDU-ID with the Service Qualifier set to the opposite sense of the Insert service.

7.2 Insert Procedure Specification

Figure 7-1 provides a conceptual overview of the Insert service. B-PDUs which has isochronous characteristics (i.e., isochronous B-PDUs) are sent from the Bitstream procedure from the upper layer to a SAP between the Bitstream service and the Insert service. Upon receiving the B-PDUs and VCDU-ID from the Bitstream procedure, an INSERT.Request is generated at the SAP to request Insert service. Figure 7-2 provides a simplistic view of such data flow.

7.2.1 Insert Procedure/Upper Layer Interface Specification

The INSERT.Request service primitive is defined as follows:

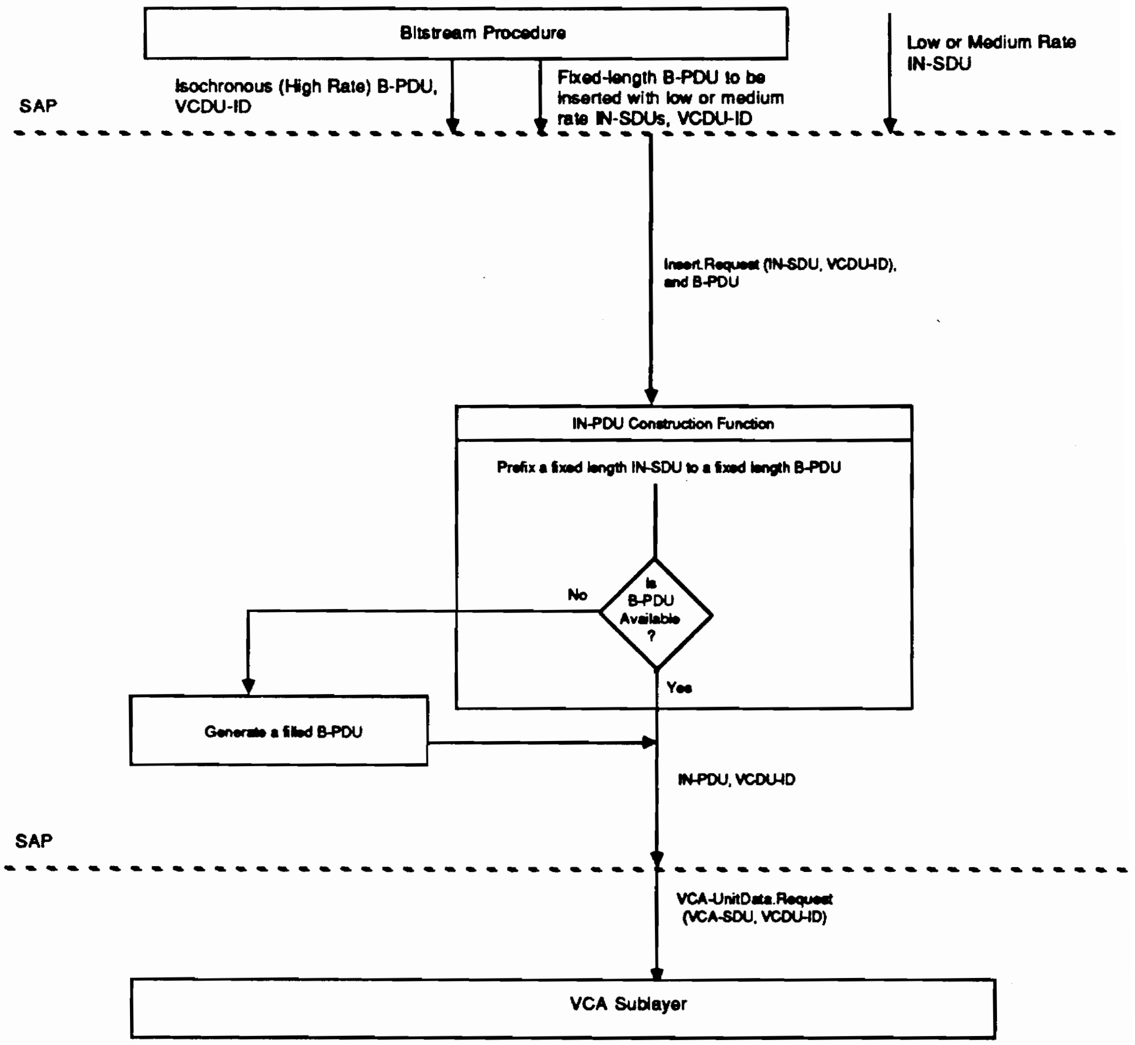

Bitstream Procedure
Low or Medium Rate IN-SDU
Isochronous (High Rate) B-PDU, VCDU-ID
Fixed-length B-PDU to be inserted with low or medium rate IN-SDUs, VCDU-ID
SAP
Insert.Request (IN-SDU, VCDU-ID), and B-PDU
IN-PDU Construction Function
Prefix a fixed length IN-SDU to a fixed length B-PDU
No
Is B-PDU Available ?
Yes
Generate a filled B-PDU
IN-PDU, VCDU-ID
SAP
VCA-UnitData.Request (VCA-SDU, VCDU-ID)
VCA Sublayer

Figure 7-1 Conceptual Overview of the Insert Service

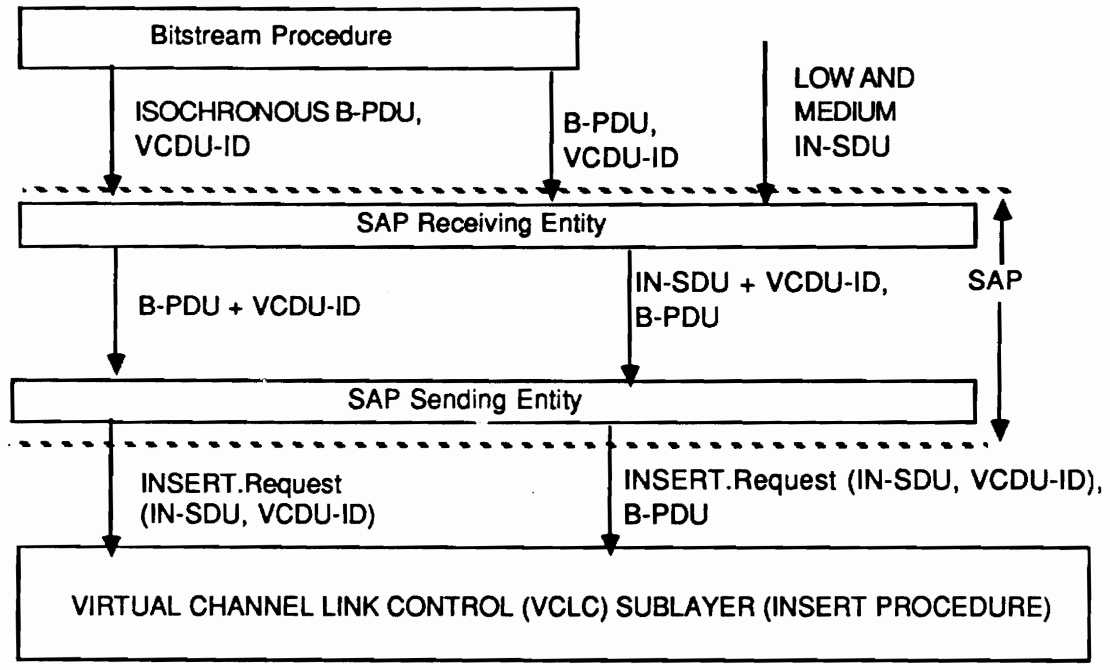

Bitstream Procedure
ISOCHRONOUS B-PDU, VCDU-ID
B-PDU, VCDU-ID
LOW AND MEDIUM IN-SDU
SAP Receiving Entity
B-PDU + VCDU-ID
IN-SDU + VCDU-ID, B-PDU
SAP
SAP Sending Entity
INSERT.Request (IN-SDU, VCDU-ID)
INSERT.Request (IN-SDU, VCDU-ID), B-PDU
VIRTUAL CHANNEL LINK CONTROL (VCLC) SUBLAYER (INSERT PROCEDURE)

Figure 7-2 Conceptual Data flow for the INSERT.Request Service Primitive

1. Function

This primitive is used to request the Insert service provided by the Insert procedure.

2. Semantics

The primitive provides the following parameters:

INSERT.Request (IN-SDU, VCDU-ID)

3. When generated

This service primitive is generated at the SAP and passed to the Insert procedure (in the VCLC sublayer) to request it to send the IN-SDU.

4. Effect on Receipt

Receipt of this primitive causes the Insert procedure (i.e., part of the VCLC sublayer) to attempt to send the IN-SDU.

5. Additional comments

The IN-SDU is provided in the INSERT.Request generated at the SAP. As a result, the IN-SDU must be supplied from the upper layer to this SAP. If a high rate IN-SDU is supplied with the INSERT.Request, the "upper layer" is the Bitstream procedure which was used to block isochronous data into IN-SDU. These are sometimes called Isochronous B-PDUs. These IN-SDU will occupy the whole data unit zone of a Virtual Channel Data Unit (VCDU) supplied by the VCA sublayer.

A separate Insert "facility" will be used to block and provide low or medium rate IN-SDUs. This "facility" (i.e., different from the Bitstream procedure) has not been clearly defined. It is conceivable that a "Isochronous Data Generator" will be used to generate blocks of IN-SDU. The procedures used to do this may be left to the implementor. These low or medium rate IN-SDU will be carried in the INSERT.Request primitive and combined with the B-PDUs from Bitstream procedure (using Insert procedure) to generate IN-PDUs. The internal interface between the Bitstream procedure and the Insert procedure is not clearly defined.

7.2.2 Insert Procedure Specification

Upon receiving the INSERT.Request service primitive the Insert procedure provides an "IN-PDU Construction Function". The IN-PDU Construction prefixed a fixed length isochronous IN-SDU (i.e., low or medium IN-SDU) to a fixed length B-PDU (received from the Bitstream procedure via an internal interface between Bitstream procedure and Insert procedure) to construct a fixed length Insert PDU (IN-PDU).

IN-SDUs are received from the Bitstream procedure in INSERT.Request primitives. There are two types of IN-SDU: high rate IN-SDU and low or medium rate IN-SDU. The high rate IN-SDUs are Isochronous B-PDUs generated by the Bitstream procedure. These IN-SDUs (which will become IN-PDUs) will occupy the whole data unit zone of the VCA-PDU provided by the VCA sublayer. The low or medium rate IN-SDUs (shorter in length) are generated by an undefined separate facility and supplied to the Insert procedure via the service primitive. These fixed-length IN-SDUs will be coupled with the regular fixed-length B-PDUs received from the Bitstream Service to form the IN-PDUs.

If due to the constraints of the IN-PDU release algorithm, a B-PDU containing valid user data is not available at release time, the IN-PDU Construction Function will internally generate a fill B-PDU.

7.2.3 Format of the IN-PDU

Figure 7-3 gives the format of the IN-PDU. The fields within the IN-PDU are defined as follows:

1. Insert Header

The length and format of this field is currently to be determined (TBD). The CCSDS is considering (as an item of study) the possibility of using this field to support an integrated PDU header identifier to discriminate between different types of PDU, and also to support the identification of IN-SDUs containing audio and video data (i.e., high, low, or medium rate IN-SDUs).

2. Insert Zone

The Insert Zone contains one IN-SDU. the length of this field is set by the CPN Management procedures.

3. B-PDU Zone

This field contains a B-PDU received from the upper layer.

7.2.4 Insert Procedure/Lower Layer Interface

The Insert procedure interfaces with the lower layer (i.e., the VCA

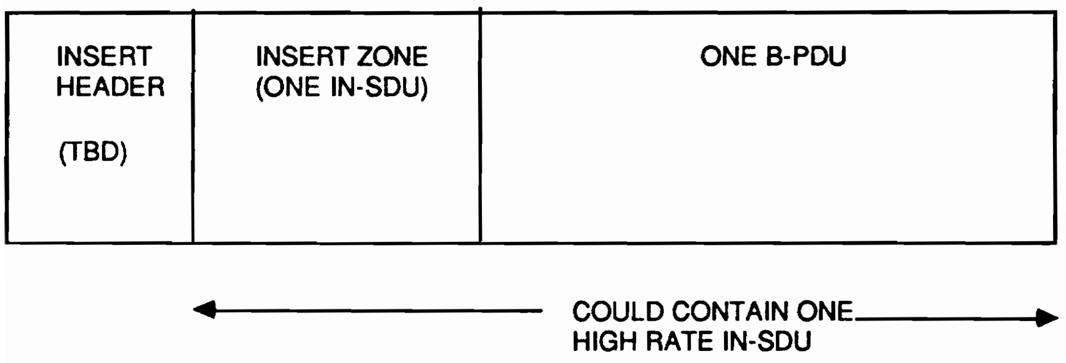


Figure 7-3 IN-PDU Format

sublayer) via the VCA-UNITDATA.Request service primitive. The VCA-UNITDATA.Request primitive consists of the VCA-PDU (i.e., IN-PDU from the Insert procedure) and the VCDU-ID. The VCDU-ID (unchanged throughout the SLS) identifies the SAP between the Insert procedure in the VCLC sublayer and the VCA procedure in the VCA sublayer.

8.0 LOTOS SPECIFICATION

Appendix A provides the LOTOS specification for the Bitstream Service and the Insert Service protocols. The specification first specifies the type of data and the related data operations used in the protocol. Upon completion of data definition, the specification specifies the processes involved in the protocol.

Throughout the specification, comments are provided to serve the following purposes:

1. To clarify the language, the specification, and the procedure used.

2. To point out places where assumptions were made. Assumptions were usually made when the requirements were not clear (i.e., validating the requirements) and needed to be studied further, or when it was necessary to simplify the procedure in order for the specification to work with HIPPO. In the latter case, those assumptions do not change the meaning of the procedures nor the requirements.

The LOTOS specification along with the comments define a protocol which meets the stated requirements. The protocol can be traced by following through the LOTOS specification. As a result, the specification acts as the intermediate step between requirements and actual implementation of the protocol.

In generating this LOTOS specification, it is found that the requirements (i.e., protocols) contain areas that need to be further defined or explicitly stated. These are documented as comments in the LOTOS specification.

9.0 HIPPO

A software tool called HIPPO is used in this research to analyze the completeness and the accuracy of the LOTOS specification. HIPPO contains a syntax checker component and a simulator. The syntax checker is used to verify that the data are correctly defined and the specication is syntactically correct. As a simulator, HIPPO offers interactive symbolic execution of the LOTOS specification. HIPPO builds a communication tree from a given LOTOS specification. The communication tree is built by interactively stepping through a specification, while selecting events from the menu of possible events in each state. At each moment during simulation the state of simulation can be displayed. While stepping through a LOTOS specfication deadlock properties of the specfication can be investigated, test sequences can be analyzed, or dynamic behavior in generally can be checked.

Appendix B provides the simulation sessions for the Bitstream and Insert procedures. For each procedure, a finite number of cases that will impact the state of the protocol are considered. Successful completion of these cases without deadlocks verifies that within the scope of the specification and the cases considered, the LOTOS specification is correct and internally consistent. The following subsections describe the cases considered for each procedure in details.

9.1 Bitstream Procedure Cases Consideration

Figure 9-1 gives the state table for the Bitstream procedure. The state table details all possible cases as specified in the specification. There are eight possible cases identified for the Bitstream procedure. These eight cases are further divided into sub-cases based upon the type of service and GOS. The cases are discussed in the following paragraphs.

Cases 1 and 2 in the state table are simulated as Case 1 in Appendix B for the Bitstream procedure. For this scenario, the timer is on and the Data Unit Zone is filled with all valid bits. Case 1 of the state table is merely an instance in the process. The simulation output is sent to the SLAP gate (sub-case 2.2) since the type of service is not Insert and the GOS is 1.

Cases 3 and 4 in the state table will never happen in the procedure since the Data Unit Zone will always contains valid bit while the timer is on.

Cases 5 and 6 in the state table are simulated as Case 2 in Appendix B for the Bitstream procedure. For this scenario, the timer goes off when the Data Unit Zone is half filled with valid bits. Case 5 of the state table is merely an instance in the process, where Project-filled data will be inserted to the remaining part of the Data Unit Zone. The simulation output is sent to the Insert gate (sub-case 6.0)

Case	Timer	Data Unit Zone Contains Valid Bits ?	Counter	B-Req Gate (Valid)	Project (Internal) (Filled)	First Header Pointer	Type of Service	GOS	SLAP Gate	VCA Gate	Insert Gate	Comments
1.0	1	Yes	Not 0	Receive	N/A	N/A	1	2	-	-	-	No output
1.1	1	Yes	Not 0	Receive	N/A	N/A	1	3	-	-	-	No output
1.2	1	Yes	Not 0	Receive	N/A	N/A	Not 1	1	-	-	-	No output
1.3	1	Yes	Not 0	Receive	N/A	N/A	Not 1	2	-	-	-	No output
1.4	1	Yes	Not 0	Receive	N/A	N/A	Not 1	3	-	-	-	No output
2.0	1	Yes	0	Stop	N/A	Allones -2	1	2	-	-	output	output
2.1	1	Yes	0	Stop	N/A	Allones -2	1	3	-	-	output	all valid
2.2	1	Yes	0	Stop	N/A	Allones -2	Not 1	1	output	-	-	bitstream
2.3	1	Yes	0	Stop	N/A	Allones -2	Not 1	2	-	output	-	data
2.4	1	Yes	0	Stop	N/A	Allones -2	Not 1	3	-	output	-	
3.0	1	No	Not 0									Never happen
4.0	1	No	0									Never happen
5.0	0	Yes	Not 0	N/A	Receive	N/A	1	2	-	-	-	No output
5.1	0	Yes	Not 0	N/A	Receive	N/A	1	3	-	-	-	No output
5.2	0	Yes	Not 0	N/A	Receive	N/A	Not 1	1	-	-	-	No output
5.3	0	Yes	Not 0	N/A	Receive	N/A	Not 1	2	-	-	-	No output
5.4	0	Yes	Not 0	N/A	Receive	N/A	Not 1	3	-	-	-	No output
6.0	0	Yes	0	N/A	Stop	Allones -2	1	2	-	-	output	output with a
6.1	0	Yes	0	N/A	Stop	Allones -2	1	3	-	-	output	mix of valid
6.2	0	Yes	0	N/A	Stop	Allones -2	Not 1	1	output	-	-	and project-
6.3	0	Yes	0	N/A	Stop	Allones -2	Not 1	2	-	output	-	specified
6.4	0	Yes	0	N/A	Stop	Allones -2	Not 1	3	-	output	-	filled bits
7.0	0	No	Not 0	N/A	Receive	N/A	1	2	-	-	-	No output
7.1	0	No	Not 0	N/A	Receive	N/A	1	3	-	-	-	No output
7.2	0	No	Not 0	N/A	Receive	N/A	Not 1	1	-	-	-	No output
7.3	0	No	Not 0	N/A	Receive	N/A	Not 1	2	-	-	-	No output
7.4	0	No	Not 0	N/A	Receive	N/A	Not 1	3	-	-	-	No output
8.0	0	No	0	N/A	Stop	Allones -1	1	2	-	-	output	output with all
8.1	0	No	0	N/A	Stop	Allones -1	1	3	-	-	output	project-specified
8.2	0	No	0	N/A	Stop	Allones -1	Not 1	1	output	-	-	filled data
8.3	0	No	0	N/A	Stop	Allones -1	Not 1	2	-	output	-	
8.4	0	No	0	N/A	Stop	Allones -1	Not 1	3	-	output	-	

Figure 9-1 Bitstream Procedure State Table

since the type of service is Insert and the GOS is 2.

Cases 7 and 8 in the state table are simulated as Case 3 in Appendix B for the Bitstream procedure. For this scenario, the timer goes off while the Data Unit Zone is empty. Case 7 of the state table is merely and instance in the process, where Project-filled data will be inserted to the whole Data Unit Zone. The simulation output is sent to the SLAP gate since the type of service is not Insert and the GOS is 1 (sub-case 8.2).

Appendix B also provides an extra case, Case 4, for the Bitstream procedure. This case is similar to Case 2 in Appendix B, except that the output is sent to the VCA gate instead (sub-case 6.3 in the state table).

All other sub-cases for the Bitstream procedure are not simulated since the results will be the same as other simulated cases except the output will be routed to different gates.

9.2 Insert Procedure Cases Consideration

Figure 9-2 gives the state table for the Insert procedure. The state table details all possible cases as specified in the specification. There are five possible cases identified for the Insert procedure. The cases are discussed in the following paragraphs.

Case	B-PDU Receiving Status	IN-SDU Receiving Status	VCDU-IDs Matched ?	Output to VCA Gate	Comments
1	Yes	Yes	No	IN-SDU + Filled B-PDU	(Case 1 in Appendix B)
2	Yes	Yes	Yes	IN-SDU + B-PDU	(Case 2 in Appendix B)
3	No	No	N/A	No Output	(Case 4 in Appendix B)
4	No	Yes	N/A	IN-SDU + Filled B-PDU	(Similar to Case 1)
5	Yes	No	N/A	No Output (Loop Back)	(Case 3 in Appendix B)

Figure 9-2 Insert Procedure State Table

Case 1 in the state table considers the state where both B-PDU and IN-SDU are available from upper layer and their VCDU-IDs are not matched. As specified in the procedure, the output will contain the IN-SDU and a filled B-PDU, which is generated internally. This case is simulated as Case 1 in Appendix B for the Insert procedure.

Case 2 in the state table considers the state where both B-PDU and IN-SDU are available from upper layer and their VCDU-IDs are matched. As specified in the procedure, the IN-SDU will be prefixed to the B-PDU and sent to the lower layer. This case is simulated as Case 2 in Appendix B for the Insert procedure.

Case 3 in the state table considers the state where no data being received from the upper layer. The procedure will produce no output and go back to the initial state. This case is simulated as Case 4 in Appendix B for the Insert procedure.

Case 4 in the state table considers the state where a B-PDU is not available from upper layer and an IN-SDU is being received from the upper layer. The procedure is similar to Case 1 in the state table, and therefore is not simulated.

Case 5 in the state table considers the state where a B-PDU is available and an IN-SDU is not. The procedure will produce no output and will go back to the initial state and wait until an IN-SDU is received. This case is simulated as Case 3 in Appendix B for the Insert procedure.

10.0 CONCLUSIONS

This research takes an existing requirements specification for the movement of Grade 2 and Grade 3 data through the space segment of the return link of the CCSDS Principal Network, reviews and analyzes the requirements, specifies a protocol to meet those requirements using LOTOS, and develop a high level requirements validation of the LOTOS specification (i.e., the protocols) through the use of HIPPO. The research offers possible solutions to certain scenarios in the protocols such as the implementation of dedicated Virtual Channel and the management lookup table. It also identifies areas which the requirements are not completely stated or vague. Specifically, these areas include: the identification of invalid bits (i.e., Project-specified bit) within a Bitstream Packet Data Unit (B-PDU), the timing requirement for the case when only B-PDUs are received and the matched IN-PDU is not available, and the possible use of a queue for the release of B-PDUs or IN-PDUs. Sucessful specification of the protocol in LOTOS (as verified through use of the HIPPO toolset) demonstrates that the protocol, as specified, is unambiguous and internally consistent. Further, by executing each process defined in the LOTOS specification against a finite set of cases that impact the protocols, the research proves that the specified protocol is complete (there are no deadlock conditions produced by the specification) and, at the requirements level, operationally correct.

In summary, the research provides an unambiguous procedural specification for the Bitstream and Insert protocols. It also tests the procedures specified by interactively stepping through the specification, while selecting possible events, and providing the dynamic behavior of the specification. The specification is provided in Appendix A. The symbolic execution of the LOTOS specification which details the tested cases is provided in Appendix B.

APPENDIX A

LOTOS SPECIFICATION FOR

THE BITSTREAM AND INSERT SERVICES

```
specification BITSTREAM [B_Req, man, slap, vca, insert, int]
                        (VCDU_ID:  VCDU_ID,
                        B_PDU:  B_PDU,
                        B_Timer: Nat,
                        B_Counter: Nat,
                        Service_Table: Service_Table): noexit
```

(* This specification represents the functional requirements of the Bitstream Procedures protocol, for use in the Return Link of the Space Link Subnetwork (SLS), as described in "Advanced Orbiting Systems, Networks and Data Links: Architectural Specification," Consultative Committee for Space Data Systems, CCSDS 701.0-R-2A, January 1989, RED BOOK Issue-2, Volume A.

The Bitstream Procedure is represented in this specification with six basic interface points: B_Req, man, slap, vca, insert, I. The I interface point is hidden within the Bitstream Procedure. As a result, only five interface points are available to external layers.

```
                                                        UPPER LAYER
+------------------------[B_Req]----------------+-------------+
| BITSTREAM_PROC                                |   CPN       |
| +---------------------------------------------| MANAGEMENT  |
| | STARTPRO                                    |             |
| | +-----------+      +------------------------|             |
| | | ADD       |      | ADDFILL                |             |
| | |           |      | +-----------+ +--------|             |
| | |           |      | | FILLALL   | | FILLHALF [man]       |
| | |           |      | |           | |        |             |
| | |           |      | +-----------+ +--------|             |
| | +-----------+      +------------------------|             |
| +---------------------------------------------|             |
+-----[slap]----[vca]------[insert]-------------+-------------+
                                                        LOWER LAYER
```

[man] represents the interface with CPN management, which is charged with the responsibility and authority to exercise control over certain Bitstream Procedure operations. Specifically, [man] will provides the service table to look up Grade of Service (GOS), B-PDU-type, and B-PDU Size based upon a given VCDU-ID.

[vca or slap or insert] represents the virtual channel access point -- the lower layer through which Bitstream Procedure communicates to send B-PDUs and to receive SLAP-PDU, VCA-PDU, or INSERT-PDU.

[int] represents internal interface between operations within Bitstream Procedure. Since [int] is a hidden interface within Bitstream Procedure, it does not represent a functional requirements; rather, it is a LOTOS-based mechanism which helps separate and define the non-deterministic operations within the Bitstream Procedure.

[B_Req] represents the upper layer access point, through which the Bitstream Procedure communicates to receive Bitstream. It is assumed that there are two type of Bitstream: valid and invalid Bitstream. The valid Bitstream is a normal and meaningful bit sent by a Bitstream source. An invalid could be an error bit or a garbage bit. It is assumed in this specification that if the procedure receives an error bit or a garbage bit, the procedure will have no way of knowing that it is receiving an invalid bit. It is assumed that the receiver will have to detect it.

BITSTREAM_PROC represents the overall Bitstream process of the Bitstream Procedure. BITSTREAM_PROC contains STARTPRO as its child process.

STARTPRO is a child process of the BITSTREAM_PROC process. STARTPRO contains 2 child processes: ADD and ADDFILL.

ADD process is used to add valid Bitstream data to the B-PDU data unit zone. The Bitstream data are received from the B_Req access point.

ADDFILL process is an internal procedure used to fill Project-specified bits into the B-PDU data unit zone. The process contains two child processes: FILLALL and FILLHALF.

FILLALL process is used when B_timer equals 0 (i.e., timeout) and the data unit zone is empty. The process fills the entire B-PDU data unit zone with Project-specified bits, sets the FHP to appropriate value, and sends the B-PDU.

FILLHALF process is used when B_timer equals 0 while receiving the valid bits from the B_Req access point. The process will fill the remaining part of the B-PDU data unit zone with Project-specified bits. *)

```
(*
     The following data type definition defines the Boolean operations
used throughout this specification.
*)
type    Boolean is
sorts   Bool
opns    true, false                 : -> Bool
        not                         : Bool -> Bool
        _and_, _or_, _xor_,
        _implies_, _iff_, _eq_,
        _ne_                        : Bool, Bool -> Bool
eqns    forall x, y: Bool
        ofsort Bool
        not(true)                   = False              ;
        not(false)                  = True               ;
        x and true                  = x                  ;
        x and false                 = false              ;
        x or true                   = true               ;
        x or false                  = x                  ;
        x xor y                     = (x and not(y)) or
                                      (y and not(x))     ;
        x implies y                 = y or not(x)        ;
        x iff y                     = (x implies y) and
```

```
                                         (y implies x)            ;
        x eq y                         = x iff y                  ;
        x ne y                         = x xor y                  ;
endtype
```

```
(*
     The following data type definition defines Basic Natural Numbers
and Natural Numbers.  Natural Numbers definition is a combination of the
Basic Natural Numbers and Boolean definitions.
     For simulation purpose, only natural numbers from 1 to 3 are
defined in the following data type definitions.  As a result, values
simulated in the specification cannot be greater than 3.
*)
type    BasicNaturalNumber is
sorts   Nat
opns    0,1,2,3                         : -> Nat
        pred                            : Nat -> Nat
        succ                            : Nat -> Nat
        _+_, _*_, _**_                  : Nat, Nat -> Nat
eqns    forall m, n : Nat
        ofsort Nat
        m + 0                           = m                     ;
        m + succ(n)                     = succ(m) + n           ;
        m * 0                           = 0                     ;
        m * succ(n)                     = m + (m * n)           ;
        m ** 0                          = succ(0)               ;
        m ** succ(n)                    = m * (m ** n)          ;
        pred(0)                         = 0                     ;
        pred(succ(m))                  = m                  ;
        succ(0)                        = 1                  ;
        succ(1)                        = 2                  ;
        succ(2)                        = 3                  ;
        pred(3)                        = 2                  ;
        pred(2)                        = 1                  ;
        pred(1)                        = 0                  ;
endtype

type    NaturalNumber is BasicNaturalNumber, Boolean
opns    GOS, B_PDU_Size, Type_of_Service: -> Nat
        _eq_, _ne_, _lt_, _le_,
        _ge_, _gt_                     : Nat, Nat -> Bool
eqns    forall m, n : Nat
        ofsort Bool
        0 eq 0                         = true               ;
        0 eq succ(m)                   = false              ;
        succ(m) eq 0                   = false              ;
        succ(m) eq succ(n)             = m eq n             ;
        m ne n                         = not(m eq n)        ;
        0 lt 0                         = false              ;
        0 lt succ(n)                   = true               ;
        succ(n) lt 0                   = false              ;
        succ(m) lt succ(n)              = m lt n                 ;
        m le n                          = (m lt n) or (m eq n)   ;
        m ge n                          = not(m lt n)            ;
        m gt n                          = not(m le n)            ;
        0 eq 1                          = false                  ;
        0 eq 2                          = false                  ;
        0 eq 3                          = false                  ;
        1 eq 1                          = true                   ;
        2 eq 2                          = true                   ;
        3 eq 3                          = true                   ;
        2 eq 1                          = false                  ;
        3 eq 1                          = false                  ;
        3 eq 2                          = false                  ;
        1 eq 2                          = false                  ;
        1 eq 3                          = false                  ;
        2 eq 3                          = false                  ;
        1 eq 0                          = false                  ;
        2 eq 0                          = false                  ;
        3 eq 0                          = false                  ;
        succ(m) eq succ(succ(m))        = false                  ;
endtype
```

```
(*
     The following definition define Bit as used in the specification.
*)
type    Bit is NaturalNumber, Boolean
sorts   Bit
opns    0, 1                             : -> Bit
        Bitstream_bit                    : -> Bit
        Proj_bit                         : -> Bit
        _eq_, _ne_, _lt_,
        _le_, _ge_, _gt_                 : Bit, Bit -> Bool
        NatNum                           : Bit -> Nat
eqns    forall x, y: Bit
        ofsort Nat
        NatNum(0)                        = 0                    ;
        NatNum(1)                        = Succ(0)              ;
        ofsort Bool
        x eq y                           = NatNum(x) eq NatNum(y);
        x ne y                           = NatNum(x) ne NatNum(y);
        x lt y                           = NatNum(x) lt NatNum(y);
        x le y                           = NatNum(x) le NatNum(y);
        x ge y                           = NatNum(x) ge NatNum(y);
        x gt y                           = NatNum(x) gt NatNum(y);
endtype
```

```
(*
     The following definition defines Bitstream as used in this
specification.  Bitstream is a set of data which contains bits.  As a
result, bits can be considered as elements in Bitstream.
*)
type    Bitstream is NaturalNumber, Bit, Boolean
sorts   Bitstream
opns    CreateBS                    : -> Bitstream
        Add                         : Bit, Bitstream -> Bitstream
        LSB                         : Bitstream -> Bit
        Tail                        : Bitstream -> Bitstream
        Append                      : Bitstream, Bitstream ->
                                      Bitstream
        LengthOf                    : Bitstream -> Nat
        NatNum                      : Bitstream -> Nat
        _eq_, _ne_, _lt_,
        _le_, _ge_, _gt_            : Bitstream, Bitstream -> Bool
eqns    forall BitA, BitB: Bit,
               BSA, BSB: Bitstream
        ofsort Bit
        LSB(CreateBS)              = 0                 ;
        LSB(Add(BitA, CreateBS))   = BitA              ;
        LSB(Add(BitA, BSA))        = BitA              ;
```

```
ofsort Bitstream

Tail(CreateBS)                    = CreateBS              ;

Tail(Add(BitA, CreateBS))         = CreateBS              ;

Tail(Add(BitA, BSA))              = Add(BitA, Tail(BSA))  ;

Append(CreateBS, BSA)             = BSA                   ;

Append(BSA, CreateBS)             = BSA                   ;

Append(Add(BitA, CreateBS), BSA)= Add(BitA, BSA)          ;

Append(Add(BitA, BSB), BSA)       = Add(BitA, Append(BSB,

                                    BSA))                 ;

Append(Add(BitA, BSB),

Add(BitB, BSA))                   = Add(BitA, Append(BSB,

                                    Add(BitB, BSA)));

Append(Add(BitA, BSB),

Add(BitB, CreateBS))              = Add(BitA, Append(BSB,

                                    Add(BitB, CreateBS))) ;

ofsort Nat

LengthOf(CreateBS)                = 0                     ;

LengthOf(Add(BitA, BSA))          = Succ(LengthOf(BSA))   ;

NatNum(CreateBS)                  = 0                     ;

NatNum(Add(BitA, CreateBS))       = NatNum(BitA)          ;

NatNum(Add(BitA, BSA))            = (NatNum(BitA) *

                                    (Succ(NatNum(1)) **

                                     LengthOf(BSA))) +
                                    NatNum(BSA)          ;
      ofsort Bool
      CreateBS eq CreateBS        = True                 ;
      BSA eq BSB                  = (LSB(BSA) eq LSB(BSB)) and
                                    (Tail(BSA) eq Tail(BSB));
      BSA lt BSB                  = NatNum(BSA) lt NatNum(BSB);
      BSA le BSB                  = NatNum(BSA) le NatNum(BSB);
      BSA gt BSB                  = NatNum(BSA) gt NatNum(BSB);
      BSA ge BSB                  = NatNum(BSA) ge NatNum(BSB);
endtype
```

```
(*
     The following definition defines Bitstream_data as used in this
specification.  Bitstream_data is a set of data which contains string of
bits.
*)
type    Bitstream_data is Bit, BasicNaturalNumber, Boolean
sorts   Bitstream_data
opns    Bitstream_data                   : -> Bitstream_data
        {}                               : -> Bitstream_data
        Add_bit, Remove                  : Bitstream_data, Bit
                                           -> Bitstream_data
        _Isin_, _Notin_                  : Bit, Bitstream_data
                                           -> Bool
        _Union_, _Ints_, _Minus_         : Bitstream_data,Bitstream_data
                                           ->Bitstream_data
        _eq_, _ne_, _Includes_,
        _IsSubsetOf_                     : Bitstream_data, Bitstream_data
                                           -> Bool
        _and_                            : Bit, Bool -> Bool
        _Fill_ , _Unfill_                : Bit, Bitstream_data
                                           -> Bool
eqns    forall x,y: Bit,
               s,t,u: Bitstream_data
```

```
ofsort Bitstream_data

Add_bit(Add_bit(s,x), x)          = Add_bit(s,x)          ;

Add_bit(Add_bit(s,y), x)          = Add_bit(Add_bit(s,x),y);

x Notin s =>

Remove(Add_bit(s,x), x)           = s                     ;

x Notin s =>

Remove (s,x)                      = s                     ;

{} Union s                        = s                     ;

s Union t                         = t Union s             ;

Add_bit (s,x) Union t             = Add_bit(s Union t, x) ;

{} Ints s                         = {}                    ;

s Ints t                          = t Ints s              ;

x Isin t =>

Add_bit(s,x) Ints t               = Add_bit (s Ints t, x) ;

x Notin t =>

Add_bit (s, x) Ints t             = s Ints t              ;

s Minus {}                        = s                     ;

s Minus Add_bit (t,x)             = Remove(s,x) Minus

                                    Remove (t,x)          ;

ofsort Bool

x Isin {}                         = false                 ;

x Isin Add_bit(s,y)               = (x eq y) or (x Isin s);

x Notin s                         = not(x Isin s)         ;
```

```
        t Minus s                       = {} =>

        s includes t                    = true                ;

        t Minus s = Add_bit (u,x) =>

        s includes t                    = false               ;

        s IsSubsetOf t                  = t includes s        ;

        s eq t                          = (s includes t) and

                                          (t includes s)      ;

        s ne t                          = not(s eq t)         ;

        x and true                      = true                ;

        x and false                     = false               ;

        y Fill Add_bit(s,x)             = (x Isin s) and

                                          (y Notin s)         ;

        y Unfill Add_bit(s,x)           = (x Isin s) and

                                          (y Isin s)          ;

endtype
```

```
(*
     The following definition defines the Spacecraft Identifier which is
part of the VCDU_ID.  The SCID consists of 8 bits.
*)
type    SCID is Bit, Boolean
sorts   SCID
opns    SCID                              : Bit, Bit, Bit, Bit,
                                            Bit, Bit, Bit, Bit
                                            -> SCID
        Get_Bit1, Get_Bit2, Get_Bit3,
        Get_Bit4, Get_Bit5, Get_Bit6,
        Get_Bit7, Get_Bit8                : SCID -> Bit
        NullSCID                          : -> SCID
        _eq_, _ne_                        : SCID, SCID
                                            -> Bool
eqns    forall b1, b2, b3, b4, b5, b6, b7, b8: bit,
               SCID1, SCID2 : SCID
        ofsort Bit
        Get_Bit1(SCID(b1, b2, b3, b4,
        b5, b6, b7, b8))                  = b1                    ;
        Get_Bit2(SCID(b1, b2, b3, b4,
        b5, b6, b7, b8))                  = b2                    ;
        Get_Bit3(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b3                        ;
Get_Bit4(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b4                        ;
Get_Bit5(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b5                        ;
Get_Bit6(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b6                        ;
Get_Bit7(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b7                        ;
Get_Bit8(SCID(b1, b2, b3, b4,
b5, b6, b7, b8))                = b8                        ;
ofsort Bool
NullSCID eq NullSCID            = True                      ;
NullSCID eq SCID(b1, b2, b3, b4,
b5, b6, b7, b8)                 = False                     ;
SCID(b1, b2, b3, b4,
b5, b6, b7, b8) eq NullSCID     = False                     ;
(Get_SCID(VCDU_ID1) ne Get_SCID(VCDU_ID2)) and
(Get_VCID(VCDU_ID1) eq Get_VCID(VCDU_ID2)) =>
VCDU_ID1 eq VCDU_ID2            = False                     ;
(SCID1 ne NullSCID) and (SCID2 ne NullSCID) =>
SCID1 eq SCID2                  = (Get_Bit1(SCID1) eq
                                   Get_Bit1(SCID2)) and
```

```
                                        (Get_Bit2(SCID1) eq
                                        Get_Bit2(SCID2)) and
                                        (Get_Bit3(SCID1) eq
                                        Get_Bit3(SCID2)) and
                                        (Get_Bit4(SCID1) eq
                                        Get_Bit4(SCID2)) and
                                        (Get_Bit5(SCID1) eq
                                        Get_Bit5(SCID2)) and
                                        (Get_Bit6(SCID1) eq
                                        Get_Bit6(SCID2)) and
                                        (Get_Bit7(SCID1) eq
                                        Get_Bit7(SCID2)) and
                                        (Get_Bit8(SCID1) eq
                                        Get_Bit8(SCID2))       ;
      SCID1 ne SCID2                  = Not(SCID1 eq SCID2)    ;
endtype
```

```
(*
     The following definition defines the Virtual Channel Identifier
(VCID) which is a part of the VCDU_ID used in this specification.  The
VCID consists of 6 bits.
*)
type    VCID is Bit, Boolean
sorts   VCID
opns    VCID                           : Bit, Bit, Bit, Bit,
                                         Bit, Bit  -> VCID
        Get_Bit1, Get_Bit2, Get_Bit3,
        Get_Bit4, Get_Bit5, Get_Bit6   : VCID -> Bit
        NullVCID                       : -> VCID
        _eq_, _ne_                     : VCID, VCID
                                         -> Bool
eqns    forall b1, b2, b3, b4, b5, b6: bit,
        VCID1, VCID2 : VCID
        ofsort Bit
        Get_Bit1(VCID(b1, b2, b3, b4,
        b5, b6))                       = b1                 ;
        Get_Bit2(VCID(b1, b2, b3, b4,
        b5, b6))                       = b2                 ;
        Get_Bit3(VCID(b1, b2, b3, b4,
        b5, b6))                       = b3                 ;
```

```
Get_Bit4(VCID(b1, b2, b3, b4,
b5, b6))                        = b4                  ;
Get_Bit5(VCID(b1, b2, b3, b4,
b5, b6))                        = b5                  ;
Get_Bit6(VCID(b1, b2, b3, b4,
b5, b6))                        = b6                  ;
ofsort Bool
NullVCID eq NullVCID            = True                ;
NullVCID eq VCID(b1, b2, b3, b4,
b5, b6)                         = False               ;
VCID(b1, b2, b3, b4, b5, b6) eq
NullVCID                        = False               ;
(VCID1 ne NullVCID) and (VCID2 ne NullVCID) =>
VCID1 eq VCID2                  = (Get_Bit1(VCID1) eq
                                  Get_Bit1(VCID2)) and
                                  (Get_Bit2(VCID1) eq
                                  Get_Bit2(VCID2)) and
                                  (Get_Bit3(VCID1) eq
                                  Get_Bit3(VCID2)) and
                                  (Get_Bit4(VCID1) eq
                                  Get_Bit4(VCID2)) and
                                  (Get_Bit5(VCID1) eq
                                  Get_Bit5(VCID2)) and
```

```
                                        (Get_Bit6(VCID1) eq
                                        Get_Bit6(VCID2))       ;
       VCID1 ne VCID2                 = Not(VCID1 eq VCID2)    ;
endtype
```

```
(*
     The following definition defines the Virtual Channel Data Unit
Identifier (VCDU_ID) which contains the SCID and the VCID.
*)
type     VCDU_ID is SCID, VCID, Boolean
sorts    VCDU_ID
opns     create_VCDU_ID                  : -> VCDU_ID
         null_VCDU_ID                    : -> VCDU_ID
         VCDU_ID                         : SCID, VCID
                                           -> VCDU_ID
         Get_SCID                        : VCDU_ID -> SCID
         Get_VCID                        : VCDU_ID -> VCID
         _eq_, _ne_                      : VCDU_ID, VCDU_ID
                                           -> Bool
eqns     forall SCID1: SCID,
                VCID1: VCID,
                VCDU_ID1, VCDU_ID2: VCDU_ID
         ofsort SCID
         Get_SCID(VCDU_ID(SCID1, VCID1)) = SCID1               ;
         ofsort VCID
         Get_VCID(VCDU_ID(SCID1, VCID1)) = VCID1               ;
         ofsort Bool
         null_VCDU_ID eq null_VCDU_ID    = True                ;
```

```
        null_VCDU_ID eq
        VCDU_ID(SCID1, VCID1)          = False               ;
        VCDU_ID(SCID1, VCID1) eq
        null_VCDU_ID                   = False               ;
        (Get_SCID(VCDU_ID1) eq Get_SCID(VCDU_ID2)) and
        (Get_VCID(VCDU_ID1) eq Get_VCID(VCDU_ID2)) =>
        VCDU_ID1 eq VCDU_ID2           = True                ;
        (Get_SCID(VCDU_ID1) eq Get_SCID(VCDU_ID2)) and
        (Get_VCID(VCDU_ID1) ne Get_VCID(VCDU_ID2)) =>
        VCDU_ID1 eq VCDU_ID2           = False               ;
        (Get_SCID(VCDU_ID1) ne Get_SCID(VCDU_ID2)) and
        (Get_VCID(VCDU_ID1) eq Get_VCID(VCDU_ID2)) =>
        VCDU_ID1 eq VCDU_ID2           = False               ;
        (VCDU_ID1 ne null_VCDU_ID) and
        (VCDU_ID2 ne null_VCDU_ID) =>
        VCDU_ID1 eq VCDU_ID2           = (Get_SCID(VCDU_ID1) eq
                                          Get_SCID(VCDU_ID2)) and
                                         (Get_VCID(VCDU_ID1) eq
                                          Get_VCID(VCDU_ID2))   ;
        VCDU_ID1 ne VCDU_ID2           = not(VCDU_ID1 eq VCDU_ID2);
endtype
```

```
(*
     The following definition defines the Service_Table parameter used
in this specification.  The Service_Table is supplied by the CPN
Management and contains Grade of Service (GOS), B_PDU_Size, Type of
Service, and VCDU_ID.  The GOS, B_PDU_Size, and Type of Service are
defined as natural numbers.
*)
type     Service_Table is NaturalNumber, VCDU_ID
sorts    Service_Table
opns     Service_Table                    : Nat, Nat, Nat, VCDU_ID
                                            -> Service_Table
         Get_GOS                          :  Service_Table -> Nat
         Get_B_PDU_Size                   :  Service_Table -> Nat
         Get_Type_of_Service              :  Service_Table -> Nat
         Get_VCDU_ID                      :  Service_Table
                                            ->VCDU_ID
eqns     forall GOS1, B_PDU_Size1, Type_of_Service1: Nat,
                VCDU_ID1: VCDU_ID
         ofsort Nat
         Get_GOS(Service_Table(GOS1, B_PDU_Size1, Type_of_Service1,
         VCDU_ID1))                      = GOS1                     ;
         Get_B_PDU_Size(Service_Table(GOS1, B_PDU_Size1,
         Type_of_Service1, VCDU_ID1))    = B_PDU_Size1;
```

```
        Get_Type_of_Service(Service_Table(GOS1, B_PDU_Size1,

        Type_of_Service1, VCDU_ID1))    = Type_of_Service1       ;

        ofsort VCDU_ID

        Get_VCDU_ID(Service_Table(GOS1, B_PDU_Size1,

        Type_of_Service1, VCDU_ID1))    = VCDU_ID1               ;

endtype
```

```
(*
     The following definition defines the First Header Pointer (FHP)
which is part of the B-PDU header field.  The FHP consists of 11 bits.
*)
type    FHP is Bit, Boolean
sorts   FHP
opns    FHP                              : Bit, Bit, Bit, Bit, Bit,
                                           Bit, Bit, Bit, Bit, Bit, Bit
                                           -> FHP
        Get_Bit1, Get_Bit2, Get_Bit3,
        Get_Bit4, Get_Bit5, Get_Bit6,
        Get_Bit7, Get_Bit8, Get_Bit9,
        Get_Bit10, Get_Bit11             : FHP ->Bit
        NullFHP                          : ->FHP
        _eq_, _ne_                       : FHP, FHP -> Bool
eqns    forall b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11: bit,
               FHP1, FHP2: FHP
        ofsort Bit
        Get_Bit1(FHP(b1, b2, b3, b4, b5,
             b6, b7, b8, b9, b10, b11)) = b1                      ;
        Get_Bit2(FHP(b1, b2, b3, b4, b5,
             b6, b7, b8, b9, b10, b11)) = b2                      ;
        Get_Bit3(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b3                    ;
Get_Bit4(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b4                    ;
Get_Bit5(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b5                    ;
Get_Bit6(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b6                    ;
Get_Bit7(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b7                    ;
Get_Bit8(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b8                    ;
Get_Bit9(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b9                    ;
Get_Bit10(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b10                   ;
Get_Bit11(FHP(b1, b2, b3, b4, b5,
      b6, b7, b8, b9, b10, b11)) = b11                   ;
ofsort Bool
NullFHP eq NullFHP           = True                      ;
NullFHP eq FHP(b1, b2, b3, b4,
b5, b6, b7, b8, b9, b1, b11)  = False                    ;
FHP(b1, b2, b3, b4, b5, b6, b7,
b8, b9, b10, b11) eq NullFHP  = False                    ;
```

```
(FHP1 ne NullFHP) and (FHP2 ne NullFHP) =>

FHP1 eq FHP2                        = (Get_Bit1(FHP1) eq
                                       Get_Bit1(FHP2)) and
                                      (Get_Bit2(FHP1) eq
                                       Get_Bit2(FHP2)) and
                                      (Get_Bit3(FHP1) eq
                                       Get_Bit3(FHP2)) and
                                      (Get_Bit4(FHP1) eq
                                       Get_Bit4(FHP2)) and
                                      (Get_Bit5(FHP1) eq
                                       Get_Bit5(FHP2)) and
                                      (Get_Bit6(FHP1) eq
                                       Get_Bit6(FHP2)) and
                                      (Get_Bit7(FHP1) eq
                                       Get_Bit7(FHP2)) and
                                      (Get_Bit8(FHP1) eq
                                       Get_Bit8(FHP2)) and
                                      (Get_Bit9(FHP1) eq
                                       Get_Bit9(FHP2)) and
                                      (Get_Bit10(FHP1) eq
                                       Get_Bit10(FHP2)) and
                                      (Get_Bit11(FHP1) eq
                                       Get_Bit11(FHP2))       ;
```

```
        FHP1 ne FHP2                     = Not(FHP1 eq FHP2)      ;
endtype
```

```
(*
     The following definition defines the Spares which is part of the B-
PDU header field.  Currently the Spares field contains 5 bits.
*)
type    Spares is Bit, Boolean
sorts   Spares
opns    Spares                          : Bit, Bit, Bit, Bit,
                                          Bit -> Spares
        Get_Bit1, Get_Bit2, Get_Bit3,
        Get_Bit4, Get_Bit5              : Spares -> Bit
        NullSpares                      : -> Spares
        _eq_, _ne_                      : Spares, Spares -> Bool
eqns    forall b1, b2, b3, b4, b5: bit,
                Spares1, Spares2: Spares
        ofsort Bit
        Get_Bit1(Spares(b1, b2, b3,
        b4, b5))                        = b1                  ;
        Get_Bit2(Spares(b1, b2, b3,
        b4, b5))                        = b2                  ;
        Get_Bit3(Spares(b1, b2, b3,
        b4, b5))                        = b3                  ;
        Get_Bit4(Spares(b1, b2, b3,
        b4, b5))                        = b4                  ;
        Get_Bit5(Spares(b1, b2, b3,
        b4, b5))                        = b5                    ;
        ofsort Bool
        NullSpares eq NullSpares        = True                  ;
        NullSpares eq Spares(b1, b2,
               b3, b4, b5)              = False                 ;
        Spares(b1, b2, b3, b4, b5) eq
        NullSpares                      = False                 ;
        (Spares1 ne NullSpares) and
        (Spares2 ne NullSpares) =>
        Spares1 eq Spares2              = (Get_Bit1(Spares1) eq
                                          Get_Bit1(Spares2)) and
                                          (Get_Bit2(Spares1) eq
                                          Get_Bit2(Spares2)) and
                                          (Get_Bit3(Spares1) eq
                                          Get_Bit3(Spares2)) and
                                          (Get_Bit4(Spares1) eq
                                          Get_Bit4(Spares2)) and
                                          (Get_Bit5(Spares1) eq
                                          Get_Bit5(Spares2))    ;
        Spares1 ne Spares2              = Not(Spares1 eq Spares2);
endtype
```

```
(*
     The following definition defines the Bitstream Protocol Data Unit
(B-PDU) header field.  The B-PDU header contains the Spares field and
the FHP field.
*)
type    B_PDU_Header is Spares, FHP
sorts   B_PDU_Header
opns    B_PDU_Header                   : Spares, FHP -> B_PDU_Header
        Get_FHP                        : B_PDU_Header -> FHP
        Get_Spares                     : B_PDU_Header -> Spares
eqns    forall FHP1: FHP, Spares1: Spares
        ofsort FHP
        Get_FHP(B_PDU_Header(Spares1, FHP1)) = FHP1               ;
        ofsort Spares
        Get_Spares(B_PDU_Header(Spares1, FHP1)) = Spares1         ;
endtype
```

```
(*
    The following definition defines the B-PDU format.  The B-PDU
contains a B-PDU header field and strings of bits (Bitstream data).
*)
type    B_PDU is B_PDU_Header, Bitstream_data
sorts   B_PDU
opns    Project_B_PDU                   : -> B_PDU
        B_PDU                           : B_PDU_Header, Bitstream_data
                                          -> B_PDU
        Get_B_PDU_Header                : B_PDU -> B_PDU_Header
        Get_Bitstream_data              : B_PDU -> Bitstream_data
eqns    forall B_PDU_Header1: B_PDU_Header,
               Bitstream_data1: Bitstream_data
        ofsort B_PDU_Header
        Get_B_PDU_Header(B_PDU(B_PDU_Header1, Bitstream_data1))
                                        = B_PDU_Header1         ;
        ofsort Bitstream_data
        Get_Bitstream_data(B_PDU(B_PDU_Header1, Bitstream_data1))
                                        = Bitstream_data1       ;
endtype
```

```
(*
     The following definition defines the Insert Service Data Unit (IN-
SDU) received from the upper Insert layer.  The format and size of the
IN-SDU is unknown.
*)
type    IN_SDU_Type is Boolean
sorts   IN_SDU_Type
opns    create_IN_SDU                   : -> IN_SDU_Type
        null_IN_SDU                     : -> IN_SDU_Type
        zeros_IN_SDU                    : -> IN_SDU_Type
        _eq_                            : IN_SDU_Type, IN_SDU_Type
                                          -> Bool
        _ne_                            : IN_SDU_Type, IN_SDU_Type
                                          -> Bool
eqns    forall IN_SDU1, IN_SDU2: IN_SDU_Type
        ofsort Bool
        IN_SDU1 eq IN_SDU1              = true          ;
        IN_SDU1 ne IN_SDU1              = false         ;
        IN_SDU1 eq IN_SDU2 =>
        IN_SDU1 eq IN_SDU2              = true          ;
        IN_SDU1 eq IN_SDU2 =>
        IN_SDU1 ne IN_SDU2              = false         ;
endtype
```

```
(*
    The following definition defines the IN-PDU which consists of a
spare field, a IN-SDU, and a B-PDU.  The spare field size is currently
undetermined and therefore is set to a single value of Natural Number.
*)
type    IN_PDU_Type is NaturalNumber, IN_SDU_Type, B_PDU
sorts   IN_PDU_Type
opns    IN_PDU                          : Nat, IN_SDU_Type, B_PDU
                                          -> IN_PDU_Type
        Get_IN_PDU_Header               : IN_PDU_Type -> Nat
        Get_IN_SDU                      : IN_PDU_Type -> IN_SDU_Type
        Get_B_PDU                       : IN_PDU_Type -> B_PDU
eqns    forall IN_PDU_Header: Nat,
               IN_SDU: IN_SDU_Type,
                B_PDU: B_PDU
        ofsort Nat
        Get_IN_PDU_Header(IN_PDU(IN_PDU_Header,
        IN_SDU, B_PDU))                 = IN_PDU_Header          ;
        ofsort IN_SDU_Type
        Get_IN_SDU(IN_PDU(IN_PDU_Header,
        IN_SDU, B_PDU))                 = IN_SDU                 ;
        ofsort B_PDU
        Get_B_PDU(IN_PDU(IN_PDU_Header,
        IN_SDU, B_PDU))                 = B_PDU                  ;
endtype
```

```
(*
    Starting the Bitstream Procedure specification
*)
behaviour BITSTREAM_PROC[B_Req, man, slap, vca, insert, int]
                        (VCDU_ID,
                         B_PDU,
                         B_Timer,
                         B_Counter,
                         Service_Table)
where
(*
    The following LOTOS specification details the Bitstream Procedures
for the Space Link Subnetwork (SLS) of the return link of the CCSDS
Principal Network (CPN).  As a result, only the procedures involved in
the return link will be specified (e.g., the Bitstream Indication
service primitive will not be discussed).
*)
process  BITSTREAM_PROC [B_Req, man, slap, vca, insert, int]
                        (VCDU_ID:  VCDU_ID,
                         B_PDU:  B_PDU,
                         B_Timer: Nat,
                         B_Counter: Nat,
                         Service_Table:  Service_Table): noexit:=
(
    B_Req      ?VCDU_ID:  VCDU_ID;
    man        ?Service_Table1:  Service_Table
         (
              Let B_Counter: Nat = Get_B_PDU_Size(Service_Table) in
                   (
                        STARTPRO  [B_Req, man, slap, vca, insert, int]
                                  (VCDU_ID,
                                   B_PDU,
                                   B_Timer,
                                   B_Counter,
                                   Service_Table1)
                   )
         )
)
```

```
where

process  STARTPRO  [B_Req, man, slap, vca, insert, int]
                   (VCDU_ID:  VCDU_ID,
                    B_PDU:  B_PDU,
                    B_Timer:  Nat,
                    B_Counter:  Nat,
                    Service_Table: Service_Table): noexit:=
(*
```

Assuming that the bitstream received from the source contain all valid bits. If the bits are not really valid bits, it is assumed that the receiver will be able to detect them. The timer is set by the managment to specify a certain waiting interval before sending a filled B-PDU. If the timer equals timeout (i.e., B_Timer equal 0) and the B-PDU is not filled, the Bitstream internal procedure will generate "Project-specified" filled bits to fill the B-PDU and send it. A value "0" is used to specified that the timer is equal timeout.

Once a B-PDU is filled, it will be sent immediately. There is no queueing of B-PDUs involved in the procedure. Also, it is assumed that bits will be inserted into the B-PDU bit by bit.

```
*)
(
    int  ?B_Timer: Nat;
    (
       [B_Timer ne 0] ->

       ADD  [B_Req, man, slap, vca, insert, int]

            (VCDU_ID,

             B_PDU,

             B_timer,

             B_Counter,

             Service_Table)

    [] [B_Timer eq 0] ->

       ADDFILL   [B_Req, man, slap, vca, insert, int]

                 (VCDU_ID,

                  B_PDU,

                  B_Timer,

                  B_Counter,

                  Service_Table)

  )

)
```

```
where

process ADD   [B_Req, man, slap, vca, insert, int]

              (VCDU_ID: VCDU_ID,

               B_PDU: B_PDU,

               B_Timer: Nat,

               B_Counter: Nat,

               Service_Table: Service_Table): noexit:=

(

    [B_Counter eq 0] ->

    (

         [Get_Type_of_Service(Service_Table) eq 1]

(* "1" stands for Insert service *)

         ->insert  !VCDU_ID

                   !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),

                                 FHP(1,1,1,1,1,1,1,1,1,0,1)),

                        Bitstream_data);

         BITSTREAM_PROC [B_req, man, slap, vca, insert, int]

                        (VCDU_ID,

                         B_PDU,

                         B_Timer,

                         B_Counter,

                         Service_Table)

      [] [Get_Type_of_Service(Service_Table) ne 1] ->
```

```
(
    [Get_GOS(Service_Table) eq 1]
    ->slap    !VCDU_ID
              !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                        FHP(1,1,1,1,1,1,1,1,1,0,1)),
                        Bitstream_data);
    BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                  (VCDU_ID,
                   B_PDU,
                   B_Timer,
                   B_Counter,
                   Service_Table)
 [] [Get_GOS(Service_Table) eq 2]
    ->vca     !VCDU_ID
              !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                  FHP(1,1,1,1,1,1,1,1,1,0,1)),
                  Bitstream_data);
    BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                  (VCDU_ID,
                   B_PDU,
                   B_Timer,
                   B_Counter,
                   Service_Table)
```

```
          [] [Get_GOS(Service_Table) eq 3]
             ->vca      !VCDU_ID
                        !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                                 FHP(1,1,1,1,1,1,1,1,1,0,1)),
                                 Bitstream_data);
             BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                           (VCDU_ID,
                            B_PDU,
                            B_Timer,
                            B_Counter,
                            Service_Table)
        )
    )
[]  [B_Counter ne 0] ->
    B_Req     ?Bitstream_bit: Bit;
    int       !B_PDU(B_PDU_Header(Spares(0, 0, 0, 0, 0),
                        FHP(0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0)),
                   Add_Bit(Bitstream_data, Bitstream_bit));
    STARTPRO  [B_Req, man, slap, vca, insert, int]
              (VCDU_ID,
               B_PDU,
               B_Timer,
               Pred(B_Counter), (* Substract counter by 1 *)
```

```
                   Service_Table)
)
endproc (* end process ADD *)
```

```
process ADDFILL    [B_Req, man, slap, vca, insert, int]
             (VCDU_ID: VCDU_ID,
              B_PDU: B_PDU,
              B_Timer: Nat,
              B_Counter: Nat,
              Service_Table: Service_Table): noexit:=
(
    [B_Counter eq Get_B_PDU_Size(Service_Table)] ->
    FILLALL  [B_Req, man, slap, vca, insert, int]
             (VCDU_ID,
              B_PDU,
              B_Timer,
              B_Counter,
              Service_Table)
 [] [B_Counter ne Get_B_PDU_Size(Service_Table)] ->
    FILLHALF [B_Req, man, slap, vca, insert, int]
             (VCDU_ID,
              B_PDU,
              B_Timer,
              B_Counter,
              Service_Table)
)
```

```
where

process  FILLALL   [B_Req, man, slap, vca, insert, int]

                   (VCDU_ID: VCDU_ID,

                    B_PDU: B_PDU,

                    B_Timer: Nat,

                    B_Counter: Nat,

                    Service_Table: Service_Table): noexit:=

(

    [B_Counter eq 0] ->

    (

         [Get_Type_of_Service(Service_Table) eq 1]

(* "1" stands for Insert service *)

         ->insert  !VCDU_ID

                   !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),

                            FHP(1,1,1,1,1,1,1,1,1,1,0)),

                        Bitstream_data);

         BITSTREAM_PROC [B_req, man, slap, vca, insert, int]

                        (VCDU_ID,

                         B_PDU,

                         B_Timer,

                         B_Counter,

                         Service_Table)

      [] [Get_Type_of_Service(Service_Table) ne 1] ->
```

```
(
     [Get_GOS(Service_Table) eq 1]
     ->slap    !VCDU_ID
               !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                         FHP(1,1,1,1,1,1,1,1,1,1,0)),
                         Bitstream_data);
     BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                    (VCDU_ID,
                     B_PDU,
                     B_Timer,
                     B_Counter,
                     Service_Table)
  [] [Get_GOS(Service_Table) eq 2]
     ->vca     !VCDU_ID
               !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                    FHP(1,1,1,1,1,1,1,1,1,1,0)),
                    Bitstream_data);
     BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                    (VCDU_ID,
                     B_PDU,
                     B_Timer,
                     B_Counter,
                     Service_Table)
```

```
                [] [Get_GOS(Service_Table) eq 3]
                   ->vca     !VCDU_ID
                             !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                                      FHP(1,1,1,1,1,1,1,1,1,1,0)),
                                      Bitstream_data);
                   BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                                  (VCDU_ID,
                                   B_PDU,
                                   B_Timer,
                                   B_Counter,
                                   Service_Table)
            )
        )
    [] [B_Counter ne 0] ->
        int  ?proj_bit: Bit;
        int  !B_PDU(B_PDU_Header(Spares(0, 0, 0, 0, 0),
                             FHP(0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0)),
                         Add_Bit(Bitstream_data, proj_bit));
        FILLALL   [B_Req, man, slap, vca, insert, int]
                  (VCDU_ID,
                   B_PDU,
                   B_Timer,
                   Pred(B_Counter), (* Substract counter by 1 *)
```

```
                    Service_Table)
)
endproc (* end process FILLALL   *)
```

```
process  FILLHALF  [B_Req, man, slap, vca, insert, int]
                (VCDU_ID: VCDU_ID,
                 B_PDU: B_PDU,
                 B_Timer: Nat,
                 B_Counter: Nat,
                 Service_Table: Service_Table): noexit:=
(
    [B_Counter eq 0] ->
    (
         [Get_Type_of_Service(Service_Table) eq 1]
(* "1" stands for Insert service *)
         ->insert  !VCDU_ID
                   !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                             FHP(1,1,1,1,1,1,1,1,1,0,1)),
                        Bitstream_data);
         BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                        (VCDU_ID,
                         B_PDU,
                         B_Timer,
                         B_Counter,
                         Service_Table)
       [] [Get_Type_of_Service(Service_Table) ne 1] ->
          (
    [Get_GOS(Service_Table) eq 1]
    ->slap     !VCDU_ID
               !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                         FHP(1,1,1,1,1,1,1,1,1,0,1)),
                         Bitstream_data);
    BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                   (VCDU_ID,
                    B_PDU,
                    B_Timer,
                    B_Counter,
                    Service_Table)
[] [Get_GOS(Service_Table) eq 2]
    ->vca      !VCDU_ID
               !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                   FHP(1,1,1,1,1,1,1,1,1,0,1)),
                   Bitstream_data);
    BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                   (VCDU_ID,
                    B_PDU,
                    B_Timer,
                    B_Counter,
                    Service_Table)
[] [Get_GOS(Service_Table) eq 3]
```

```
            ->vca       !VCDU_ID
                        !B_PDU(B_PDU_Header (Spares(0,0,0,0,0),
                                  FHP(1,1,1,1,1,1,1,1,1,0,1)),
                                  Bitstream_data);
            BITSTREAM_PROC [B_req, man, slap, vca, insert, int]
                           (VCDU_ID,
                            B_PDU,
                            B_Timer,
                            B_Counter,
                            Service_Table)
        )
    )
[]  [B_Counter ne 0] ->
    int  ?proj_bit: Bit;
    int  !B_PDU(B_PDU_Header(Spares(0, 0, 0, 0, 0),
                   FHP(0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0)),
            Add_Bit(Bitstream_data, proj_bit));
    FILLHALF  [B_Req, man, slap, vca, insert, int]
            (VCDU_ID,
             B_PDU,
             B_Timer,
             Pred(B_Counter), (* Substract counter by 1 *)
             Service_Table)
```

```
)
endproc (* end process FILLHALF         *)
endproc (* end process ADDFILL          *)
endproc (* end process STARTPRO         *)
endproc (* end process BITSTREAM_PROC   *)
endspec (* end specification BITSTREAM *)
```

```
specification  INSERT   [Bitstream, man, mux, upperInsert, vca, int]
                        (VCDU_ID:  VCDU_ID,
                         B_PDU:  B_PDU,
                         IN_SDU: IN_SDU_Type
                        ): noexit
 (*
```

This specification represents the functional requirements of the Insert Procedures protocol, for use in the Return Link of the Space Link Subnetwork (SLS), as described in "Advanced Orbiting Systems, Networks and Data Links: Architectural Specification," Consultative Committee for Space Data Systems, CCSDS 701.0-R-2A, January 1989, RED BOOK Issue-2, Volume A.

The Insert Procedure is represented in this specification with six basic interface points: Bitstream, mux, man, upperInsert, vca, and int. The int interface point is hidden within the Insert Procedure. As a result, only five interface points (i.e., Bitstream, mux, man, upperInsert, and vca) are available to external sublayers.

```
                                   UPPER LAYER
+----[mux]---------[Bitstream]--[upperInsert]--+
| INSERT PROCEDURE                             |
|                                              |
|                                            [man] CPN
|                                              |    MANAGEMENT
|                                              |
+------------[vca]-----------------------------+
                                   LOWER LAYER
```

[man] represents the interface with CPN management, which is charged with the responsibility and authority to exercise control over certain Insert Procedure operations.

[vca] represents the virtual channel access point -- the lower layer through which Insert Procedure communicates to send IN-PDU and to receive VCA-PDU.

[int] represents internal interface between operations within Insert Procedure. Since [int] is a hidden interface within Insert Procedure, it does not represent a functional requirements; rather, it is a LOTOS-based mechanism which helps separate and define the non-deterministic operations within the Bitstream Procedure.

[Bitstream] represents the upper layer access point, through which the Insert Procedure communicates to receive high, low, or medium rate B-PDU.

[upperInsert] represents the upper layer access point, through which the Insert Procedure communicates to receive and send IN-SDUs.

[mux] represents the upper layer access point, through which the Insert Procedure communicates to receive and send M-SDUs. This [mux] gate is similar to the [Bitstream] gate. Since the scope of this thesis only addresses the Bitstream data, the [mux] gate is noted but will not be specified in details in the procedure.

*)

(* The data type definitions for this specification is exactly the same as of the Bitstream specification. As a result, it will not be repeated here. However, in actual simulation, it must be repeated in order for the simulation to run.

*)

```
 (*  Starting the Insert Procedure specification  *)
behaviour  INSERT_PROC  [Bitstream, mux, man, upperInsert,
                         vca, int]
                        (VCDU_ID, B_PDU, IN_SDU)
 where
(*
     The following LOTOS specification details the Insert Procedures for
the Space Link Subnetwork (SLS) of the return link of the CCSDS
Principal Network (CPN).  As a result, only the procedures involved in
the return link will be specified (e.g., the Insert Indication service
primitive will not be discussed).
*)
process    INSERT_PROC  [Bitstream, mux, man, upperInsert, vca, int]
                        (VCDU_ID: VCDU_ID,
                         B_PDU:  B_PDU,
                         IN_SDU:  IN_SDU_Type
                        ): noexit:=
        int    ? B_PDU_Avail:  Nat
               ? IN_SDU_Avail: Nat;
(*
     Enter 1 to indicate that a B-PDU or an IN-SDU is available.  Enter
0 to indicate otherwise.
*)
```

```
      (
         [B_PDU_Avail eq succ(0)] ->
         (
            [IN_SDU_Avail eq succ(0)] ->
            upperInsert  ?VCDU_ID1: VCDU_ID;
            Bitstream    ?VCDU_ID2: VCDU_ID;
            (
                [VCDU_ID1 eq VCDU_ID2] ->
                upperInsert       ? IN_SDU: IN_SDU_Type;
                Bitstream         ? B_PDU: B_PDU;
                vca               ! VCDU_ID1
                                  ! IN_PDU(0, IN_SDU, B_PDU);
(*
      The IN-PDU-Header is set to "0" since its value is currently to be
determined.

*)
                INSERT_PROC[Bitstream, mux, man, upperInsert, vca, int]
                           (VCDU_ID,
                            B_PDU,
                            IN_SDU
                            )
             [] [VCDU_ID1 ne VCDU_ID2] ->
            upperInsert      ? IN_SDU: IN_SDU_Type;

            int              ! Project_B_PDU;

            vca              ! VCDU_ID1

                             ! IN_PDU(0, IN_SDU, Project_B_PDU);

            INSERT_PROC[Bitstream, mux, man, upperInsert, vca, int]

                       (VCDU_ID,

                        B_PDU,

                        IN_SDU

                       )

        )

     [] [IN_SDU_Avail ne succ(0)] ->

        INSERT_PROC[Bitstream, mux, man, upperInsert, vca, int]

                   (VCDU_ID,

                    B_PDU,

                    IN_SDU

                   )

     )

  [] [B_PDU_Avail ne succ(0)] ->

     (

        [IN_SDU_Avail eq succ(0)] ->

        upperInsert          ? VCDU_ID: VCDU_ID

                             ? IN_SDU: IN_SDU_Type;

        int                  ! Project_B_PDU;
```

```
          vca                    ! VCDU_ID

                                 ! IN_PDU(0, IN_SDU, Project_B_PDU);

          INSERT_PROC[Bitstream, mux, man, upperInsert, vca, int]

                      (VCDU_ID,

                    B_PDU,

                    IN_SDU

                      )

       [] [IN_SDU_Avail ne succ(0)] ->

          INSERT_PROC[Bitstream, mux, man, upperInsert, vca, int]

                     (VCDU_ID,

                      B_PDU,

                      IN_SDU

                     )

       )

    )

endproc (* end process insert *)

endspec
```

APPENDIX B

HIPPO SIMULATION RESULTS

(*

The following section gives the simulation results of the Bitstream Service in communication tree format. The communication tree is the tree of events that were performed during the simulation. The output of tree gives in the first column the state number, in the second column the depth in the tree, and in the last column the last event that was performed to reach that state. The current state is marked with >>>.

*)

(*

Starting Case 1: where timer does not goes off, all valid data received from b_req gate, GOS =1, B_PDU_Size = 3, and Type_of_Service is not insert

*)

```
0    0    START  (* Starting the simulation *)
1    1    b_req  !null_vcdu_id (* getting the vcdu_id from the b_req
          gate *)
2    2    man  !service_table(1, 3, 2, null_vcdu_id) (* getting the
          service table from the CPN management *)
3    3    int  !1 (* internally sensing the timer; a value one
          indicates that the timer does not equal timeout yet. *)
```

```
4    4     b_req  !1 (* receiving a bit "1" from the b_req gate *)
5    5     int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                      fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                  add_bit(bitstream_data, 1)) (* Add the bit to the
           B-PDU Data Unit Zone (i.e., Bitstream_data) *)
6    6     int  !1  (* internally sensing the timer again *)
7    7     b_req  !1 (* receiving another bit "1" from b-req gate *)
8    8     int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                      fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                  add_bit(bitstream_data, 1)) (* internally add the bit
               to the data unit zone *)
9    9     int  !1 ( * internally sensing the timer again *)
10   10    b_req  !0 (* receiving bit "0" from the b_req gate *)
11   11    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                      fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                  add_bit(bitstream_data, 0)) (* internally add the bit
               to the data unit zone *)
12   12    int  !1 (* internally sensing the timer again *)
13   13    slap  !null_vcdu_id
           !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                 fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)), bitstream_data)
           (* output at gate "slap" the vcdu_id, and the b_pdu.  Note
           that the timer is still on but the data unit zone has been
```

```
          filled, therefore the procedure will output the pdu
          regardless of the timer *)
(*
     Starting Case 2: where the timer will go off during receiving
bitstream, GOS = 2, B_PDU_Size = 3, and the Type_of_Service is insert.
*)
14   14   b_req  !null_vcdu_id  (* getting vcdu_id from b_req *)
15   15   man  !service_table(2, 3, 1, null_vcdu_id) (* lookup
          management service table provided by the management *)
16   16   int  !1 (* internally sense the timer *)
17   17   b_req  !0 (* receiving a valid bit "0" from the b_req gate *)
18   18   int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                    fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                    add_bit(bitstream_data, 0)) (* internally add the
                   bit to the data unit zone *)
19   19   int  !0 (* internally sense the timer.  Timer is Timeout now!
          *)
20   20   int  !1 (* receiving a project-specified bit internally *)
21   21   int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                    fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                    add_bit(bitstream_data, 1)) (* Add the bit to to
                   the data unit zone *)
22   22   int  !1 (* keep receiving another project-specified bit
          internally *)
23   23   int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                      fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
             add_bit(bitstream_data, 1)) (* Add the bit to the data
             unit zone *)
24   24   insert  !null_vcdu_id
                       !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                                fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)),
                                bitstream_data)  (* output the vcdu_id and
                   the  b_pdu to the insert gate (data unit zone has
                   been filled with 3 bits *)
(*
     Start Case 3: where B-Timer goes off and the data unit zone is
empty.  Therefore the entire data unit zone is filled with project-
specified bits.
*)
25   25   b_req  !null_vcdu_id (* getting the vcdu_id from b_req *)
26   26   man  !service_table(1, 3, 2, null_vcdu_id) (* lookup service
          table provided by the management for the GOS, B_PDU_Size, and
          the Type_of_Service based upon the vcdu_id received *)
27   27   int  !0 (* internally sense that the timer is equal timeout
          *)
28   28   int  !1 (* receiving a project-specified bit internally *)
```

```
29    29    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                     add_bit(bitstream_data, 1)) (* Add the bit to the
                   data unit zone *)
30    30    int  !0 (* receiving a project-specified bit internally *)
31    31    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                     add_bit(bitstream_data, 0)) (* Add the bit to the
                   data unit zone *)
32    32    int  !1 (* receiving a project-specified bit internally (last
            bit) *)
33    33    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                     add_bit(bitstream_data, 1)) (* Add the bit to the
                   data unit zone *)
34    34    slap  !null_vcdu_id
                  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                              fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)),
                              bitstream_data)
              (* output the vcdu_id and the b_pdu to the slap gate (
              GOS = 1) *)
(*
    Start Case 4: where, the GOS = 2 and the Type_of_Service is not an
```

```
insert service and the timer will be off in the middle of the procedure.
This is similar to Case 2 except that the output will be sent to vca
gate.
*)
35   35    b_req  !null_vcdu_id
36   36    man  !service_table(2, 3, 2, null_vcdu_id)
37   37    int  !1
38   38    b_req  !1
39   39    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                     add_bit(bitstream_data, 1))
40   40    int  !1
41   41    b_req  !1
42   42    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                add_bit(bitstream_data, 1))
43   43    int  !0
44   44    int  !1
45   45    int  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                     fhp(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)),
                     add_bit(bitstream_data, 1))
46   46    >>>vca  !null_vcdu_id
                       !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)),

bitstream_data)
```

(*

The following section gives the simulation results of the Insert Service in communication tree format. The communication tree is the tree of events that were performed during the simulation. The output of tree gives in the first column the state number, in the second column the depth in the tree, and in the last column the last event that was performed to reach that state. The current state is marked with >>>.

*)

(*

Start Case 1: Both B-PDU and IN-SDU are available from upper layer. Also, the vcdu_ids do not match for the data received from the bitstream gate and the upperinsert gate.

*)

```
0    0      START (* Starting the procedure *)

1    1      int  !1  !1 (* internally sensing if a B_PDU and a IN_SDU are
            available from upper layer, in this case they both are
            available *)

2    2      upperinsert  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                                  vcid(1, 1, 1, 0, 0, 0))
            (* receiving vcdu_id from upperinsert gate from upper layer
            *)

3    3      bitstream  !vcdu_id(scid(1, 0, 0, 0, 1, 1, 1, 1),
                                vcid(1, 0, 0, 1, 1, 1))
```

```
          (* receiving vcdu_id from bitstream gate from upper layer *)
4    4    upperinsert  !create_in_sdu (* since the vcdu_ids do not
          match, the procedure only accepts a in_sdu (i.e.,
          create_in_sdu) from the upperinsert gate and ignore the
          bitstream gate *)
5    5    int  !project_b_pdu (* internally the procedure generates a
          project_b_pdu to fill in the b_pdu zone *)
6    6    vca  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                               vcid(1, 1, 1, 0, 0, 0))
                !in_pdu(0, create_in_sdu, project_b_pdu)
          (* output the vcdu_id and the in_pdu to the vca gate.  The
          in_pdu contains the in_sdu, the project_b_pdu (pre-defined),
          and a header set to value 0 *)
(*
     Start Case 2: where both the B-PDU and the IN-SDU are available and
the vcdu_ids match.
*)
7    7    int  !1  !1  (* sensing b_pdu and in_sdu availability *)
8    8    upperinsert  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                                      vcid(1, 1, 1, 0, 0, 0))
9    9    bitstream  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                                    vcid(1, 1, 1, 0, 0, 0))
10   10   upperinsert  !create_in_sdu (* since the vcdu_ids match, the
            procedure receives an in_sdu from the upperinsert gate *)
11    11    bitstream  !b_pdu(b_pdu_header(spares(0, 0, 0, 0, 0),
                                  fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)),
                                  bitstream_data) (* The procedure also
                       receives a b_pdu from the bitstream gate *)
12    12    vca  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                              vcid(1, 1, 1, 0, 0, 0))
            !in_pdu(0, create_in_sdu, b_pdu(b_pdu_header
                   (spares(0, 0, 0, 0, 0),
                   fhp(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)),
                   bitstream_data))
            (* output the vcdu_id and the in_pdu to the vca gate *)
(*
      Start Case 3: where B_PDU is available and IN_SDU is not available.
*)
13    13    int  !1  !0 (* internally sensing the b_pdu is available and
            in_sdu is not available *)
14    14    int  !0  !1 (* returning back to sense again until a in_sdu
            is available.  The procedure does not accept data from
            bitstream gate without an in_sdu available.  At present
            state, the procedure sense that a b_pdu is not available but
            a in_sdu is available, start a new case now. *))
15    15    upperinsert  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                                      vcid(1, 1, 1, 0, 0, 0))
                               !create_in_sdu (* receving in_sdu from
                          upperinsert gate from upper layer *)
16    16     int  !project_b_pdu (* internally create a project-filled
             b_pdu *)
17    17     vca  !vcdu_id(scid(1, 1, 1, 1, 0, 0, 0, 0),
                                 vcid(1, 1, 1, 0, 0, 0))
                      !in_pdu(0, create_in_sdu, project_b_pdu)
             (* output the vcdu_id received from the upperinsert and the
             in_pdu . *)
(*
      Start Case 4: where none of the data are available.
*)
18    18     int  !0  !0 (* data are not available, the procedure goes
             back to waiting stage and keep sensing for data. *)
19    19     >>>int  !0  !0
```

REFERENCES

[1] Consultative Committee for Space Data Systems, "Advanced Orbiting Systems, Network and Data Links: Architectuaral Specification", CCSDS 701.0-R-2A, RED BOOK Issues-2, Volume A, January 1989, Section 5.

[2] Consultative Committee for Space Data Systems, "Procedures Manual for the Consultative committee for Space Data Systems", CCSDS Document, Issue-1, August 1985.

[3] Gamble, Mark, Survey of Formal Definition Techniques, January 1989.

[4] ISO 7498, "Information Processing Systems - Open Systems Interconnection", Basic Reference Model.

[5] ISO 8348, "Information Processing Systems - Data Communications", Network Service Definition.

[6] ISO/DIS 8473, "Information Processing Systems - Data Communications", Protocol for Providing the Connectionless-mode Network Service.

[7] ISO/DIS 8807, "Information Processing System - Open Systems Interconnection", LOTOS (Formal description technique based on the temporal ordering of observational behaviour.

[8] Logrippo, L., Obaid, A., Briand, J. P., and Fehri, M. C., An Interpreter for LOTOS, A Specification Language for Distributed Systems, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada KIN 9B4, November 1987.

[9] Noles, J., McCoy, E., Pietras, J., "A Network Service Reference Configuration for the Space Station Information System", MITRE Working Paper, WP 88W00139, October 1988.

GLOSSARY

ARQ	Automatic Repeat Queueing
BNSC	British national Space Centre
B-PDU	Bitstream Protocol Data Unit
B-SDU	Bitstream Service Data Unit
CADU	Channel Access Data Unit
CAST	Chinese Academy of Space Technology
CCSDS	Consultative committee for Space Data Systems
CNES	Centre national D'Etudes Spatiales
CPN	CCSDS Principal Network
CVCDU	Coded Virtual Channel Data Unit
C&T	Communication and Tracking
DFVLR	Deutsche Forschungs-u Versuchsanstalt fuer Luft und Raumfahrt
DIF	Data Interface Facility
DOC-CRC	Department of Communications, Communications Research Centre
DMS	Data Management System
ESA	European Space Agency
FDT	Formal Description Technique
FHP	First Header Pointer
GOS	Grade of Service
ID	Identifier
IN-PDU	Insert Protocol Data Unit
INPE	Insitituto de Pesquisas Espacialis
IN-SDU	Insert Service Data Unit
ISO	International Organization for Standardization
ISRO	Indian Space Research Organization
LOTOS	Language of Temporal Ordering System
NASA	National Aeronautics and Space Administration
NASDA	national Space Development Agency of Japan
OSI	Open Systems Connection
PCA	Physical Channel Access
PCI	Protocol Control Information
PDU	Protocol Data Unit

SAP	Service Access Point
SCID	Spacecraft Identifier
SDU	Service Data Unit
SLAP	Space Link ARQ Procedure
SLAP-PDU	SLAP Protocol Data Unit
SLAP-SDU	SLAP Service Data Unit
SLS	Space Link Subnetwork
SSIS	Space Station Information System
TBD	To Be Determined
VC	Virtual Channel
VCA	Virtual Channel Access
VCA-PDU	Virtual Channel Access Protocol Data Unit
VCA-SDU	Virtual Channel Access Service Data Unit
VCDU	Virtual Channel Data Unit
VCID	Virtual Channel Identifier
VCDU-ID	Virtual Channel Data Unit Identifier
VCLC	Virtual Channel Link Control
VCGW	Virtual Channel Gateway