SDL Specification and Verification of a Distributed Access Generic Optical Network Interface for SMDS Networks

Sharif M. Shahrier[†] Roy M. Jenevein[‡]

[†]Department of Electrical and Computer Engineering
[‡]Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1014

Abstract

This paper presents the design and specification of a BISDN user-to-network interface (UNI) named DRAGON (Distributed Access Generic Optical Network) for SMDS networks. The UNI allows clusters of nodes to be connected to an SMDS network via fiber-optic lines. The capacity of each line is shared by all the nodes in the cluster to make more efficient use of bandwidth.

Within each cluster, transmissions are scheduled on first-come-first-served (FCFS) order of message arrivals, by considering a globally distributed queue. A novel scheme is proposed for controlling access to the fiber-optic transmission network. By using two logically separate subnetworks called the *reservation channel* and the *reservation ring* Slot reservations and message transmissions can proceed independently and concurrently. The reservation channel is a broadcast channel for notifying nodes within the cluster when to reserve a slot. Access to the reservation channel is controlled by the reservation ring: a token ring network. All accesses for the queue slots are completely fair and the bandwidth attainable is independent of the position of the node within the cluster. Unlike previous distributed queue protocols, the DRAGON facilitates both fixed-sized and variable sized transmissions.

We constructed an extended finite state model (EFSM) of the DRAGON using ITU standard Specification and Description Language (SDL). The model was simulated and validated using the SDT 3.02 toolset from Telelogic. An extensive set of simulations were conducted to acertain correct logical behavior. The model was then independently verified using two different algorithms: bit-state and random walk. The results showed that the design was verified to a high degree of coverage.

Index Terms – computer networks, BISDN, SMDS, UNI, SDL.

1 Introduction

Broadband Integrated Services Digital Networks (BISDN) have been an active area of research over the last decade. Both local and metropolitan area networks (LANs & MANs) have appeared in great proliferation in the marketplace, and a large number have been deployed in commercial and academic sites. To support the diverse applications required of BISDN networks, a large number of protocols have been developed, for example ATM [1], SONET [2], SMDS [3], frame relay [4], X.25 [5], and many others. Optical fibers provide the transmission infrastructure for BISDN networks to allow transmission at hundreds of megabytes per second. Recently, international standards for BISDN has been produced by the International Telecommunications Union (ITU) so that equipment produced by different manufacturers were compatible with one another.

An important part of broadband networks is the the *user-to-network interface (UNI)* [6]: the interface between the host node and the broadband network. In this paper, we designed a Switched Multimegabit Data Services (SMDS) UNI called *DRAGON (Distributed Access Generic Optical Network)*. The DRAGON allows a cluster of nodes to share the same transmission medium at the UNI, by scheduling transmissions in a globally distributed First-Come-First-Served (FCFS) order of message arrivals. This is an optimal transmission policy in the sense that it has the smallest delay and delay variance of all other transmission policies [7]. Thus, it maximizes the throughput and minimizes the delay and delay jitter. It is also possible to determine the upper delay bound of cells. One method is to estimate the mean and variance of the cell delay and then use Chebyshev's inequality to estimate the upper delay bound. This had been illustrated in [8].

The distributed queue in the DRAGON is called a *reservation queue (RESVQ)*. Other variants of this scheme have been implemented in other networks, for example DQDB [9], Hangman [10] and S++ [11]. One of our main objective was to improve on the previous distributed queue designs. Recently, there had not been a great deal of interest in the DQDB type of protocols. We hope that the improvements we've proposed to the distributed queue architecture will generate more interest.

Most of the previous distributed queue protocols were developed for dual slotted-bus networks and cannot be applied directly to multiple node clusters connected to a *single* high-speed fiber-optic line, as is the case in our design. Our design is also different in that slot reservations and message transmissions can proceed independently and concurrently on separate logical channels, by sharing the same optical medium, by means of Wavelength Division Multiplexing (WDM). This means all the transmission bandwidth can be utilized for transmitting messages, and none of it is utilized for transporting slot reservation information. DQDB and S++ uses the same channel for transmissions and slot reservations. Access to the slots in the DRAGON are completely fair and is independent of the nodes position within the cluster, whereas in DQDB the nodes closest to the slot generators have a greater chance of reserving a slot. Hence, downstream nodes have fewer chances of acquiring a slot, and consequently the bandwidth attainable to a node is dependent on its position in the network. This problem does not arise with the DRAGON.

DQDB and S++ transmits only fixed-sized synchronous packets. Thus, they cannot be used directly for transmitting variable-sized packets such as X.25 and frame relay. The DRAGON, however, is suitable for both fixed-sized packet and variable-sized packet transfers. We have developed a SMDS and frame relay versions of the DRAGON to illustrate this point, and performance results

are presented in [12]. And finally, we included a reservation FIFO within the DRAGON so that slot reservations for multiple number of messages can be made. This feature was not supported by the DQDB and S++ protocols, thus they can only reserve one message slot at a time.

An SMDS network architecture is shown in Fig. 1. Data is transmitted in the network in the form of fixed sized cells. The SMDS interface protocol (SIP) converts data into cells and transmits them to the SONET/SMDS interface (SSI) at a rate of 49.54 Mbits/sec, i.e. the capacity of the SONET/STS-1 payload. The cells are then transported via SONET to an SMDS switching network. The SIP is organized into three layers. The highest layer, layer 3, accepts data from the higher layer protocols and converts it into variable sized frames referred to as AAL3/4 CS-PDU (ATM Adaptation Layer 3/4 Convergence Sublayer-Protocol Data Unit). Layer 2 takes these frames and converts it into fixed sized cells of 48 byte payload and 5 byte header referred to as AAL3/4 SAR-PDU (ATM Adaptation Layer 3/4 Segmentation And Reassembly-Protocol Data Unit). The AAL3/4 cells are encapsulated onto a STS-1 payload by the SSI and transmitted over the fiber-optic line at a rate of 51.84 Mbits/sec to the SMDS network. At the switching network, the cells are extracted from the STS-1 payload and routed via the switches. Before leaving the network, the cells are again remapped onto the STS-1 payload and transported over the fiber-optic line to the destination cluster.

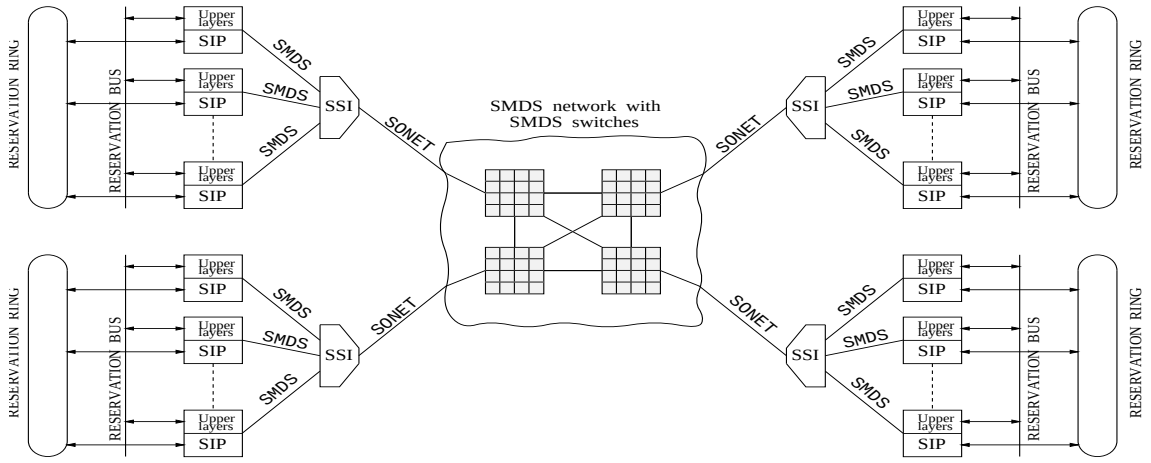


Figure 1: SMDS network with 4 node clusters

A diagram depicting the development steps of the DRAGON is shown in Fig. 2. The development cycle consists of a dual track approach. In the first phase, a formal specification of DRAGON was presented in SDL (Specification and Description Language) [13, 14, 15, 16]. This is a non-proprietary international standard notation based on Extended Finite State Machines. There are several tools for SDL, but we entered a complete set of SDL/GR (Graphical Representation) diagrams into the SDT (SDL Design Tool) from Telelogic. Finally, conformance and validation testing were performed to ensure that the design functioned correctly. Conformance testing was to ensure that the SDL model of the DRAGON conforms to the original specification. This was done by simulating the SDL system using a known set of inputs and observing the outputs. Validation was performed using two well established algorithms called *bit-state* and random-walk [17, 18]. These methods were provided by the SDL validator. The second phase of development consists of an RTL implementation model of DRAGON in VHDL. Extensive timing simulations were performed

using concurrent video and data traffic. Trace driven performance simulation was performed using an integrated mix of video and data traffic using actual traces provided by Bellcore. This paper is concerned with the SDL modeling and validation of DRAGON. The RTL development aspects were treated in a separate publication [8, 19]. The remainder of the paper is organized as follows. Section 2 provides an informal description of the DRAGON architecture. Section 3 presents a complete set of SDL specification diagrams of the DRAGON. Section 4 provides a description of the two validation methods used in this work. The conformance test and validation coverage results are presented in section 5. The paper is concluded in section 6.

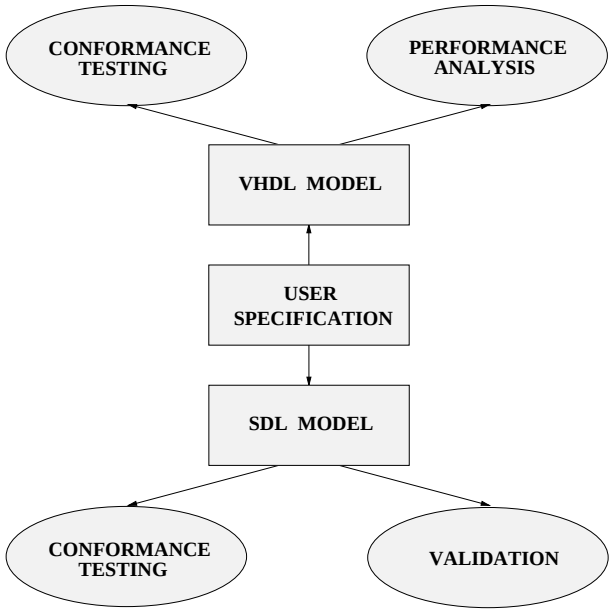


Figure 2: Development steps of the DRAGON in SDL

2 The DRAGON Overview

2.1 The Cluster Organization

This section provides an informal presentation of the DRAGON architecture for the clarity of understanding. The interconnection of the nodes within a cluster is shown in Fig. 3. All accesses to the network must be controlled so that there are no collisions. This is done in part by two different subnetworks: the *reservation bus* and the *reservation ring*. The reservation bus is a single-bit broadcast channel used for notifying all nodes when to reserve one slot in the reservation queue (RESVQ). Only one node at a time can send on the reservation bus. Thus, to prevent more than 1 node transmitting simultaneously, a reservation ring is used. The reservation ring is a high-speed, single-bit token ring network for controlling access to the reservation channel. The channels operate as follows.

The reservation ring has basically a single-bit token ring architecture. The difference is that after the token is received, the node is only allowed to reserve a slot in the RESVQ; it cannot

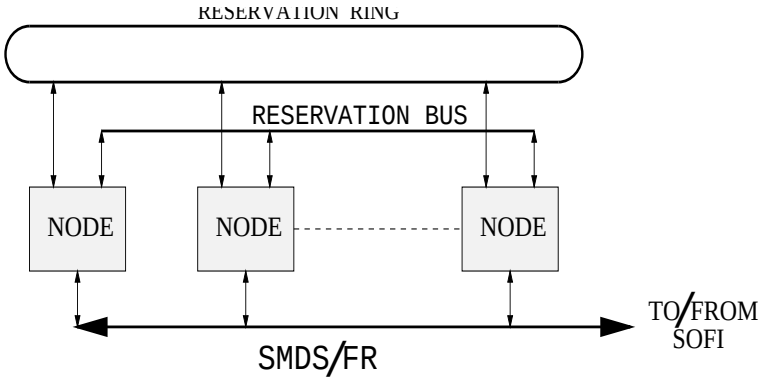


Figure 3: LAN configuration

transmit packets immediately after receiving the token as in normal token ring networks. A signal called **TOKen** circulates around the reservation ring, passing from one node to the next. If no new messages have arrived at a node, the node doesn't reserve any slots in the RESVQ. It simply passes the **TOKen** to its successor node. If however, a message has arrived at a node, the node waits for the **TOKen**, reserves a slot in its RESVQ and notifies all other nodes to also reserve a slot by transmitting a **RESV** signal on the reservation channel. The **TOKen** is then passed to its successor. Hence, by requiring a node to own the **TOKen** before broadcasting **RESV** will guarantee that there will be no multiple simultaneous transmissions of **RESV** on the reservation channel, and thus no collisions. The reservation channel and the reservation ring are independent subnetworks and may share the same fiber-optic medium by WDM technique.

2.2 The DRAGON Prototype and Datapath

We shall now provide the layout of the individual functional units within the DRAGON. A block diagram of the DRAGON prototype and its datapath signals are shown in Fig. 4. Signals are indicated by the signal names placed inside squared brackets. The direction of the signal is indicated by arrow heads. Some of the signal routes have arrow heads at both ends, in which case these signal routes are bidirectional. The SONET transmit and receive links to and from the switching network are called **SONETxmt** and **SONETrcv** respectively. The transmit and receive lines connecting the DRAGON to the SOFI are labeled **PKTxmt** and **PKTrcv** respectively. Next, we shall discuss each of the functional units within the DRAGON prototype. A complete description of the signals and transition tables for the finite state machines are provided in [8].

2.3 The Reservation Queue (RESVQ) Operation

Every node in the network contains a reservation queue. As explained previously, after a message arrives, a slot must be reserved in the reservation queue. Consequently, the RESVQs of all the nodes must be updated. The basic blocks of the RESVQ consist of the reservation queue, the reservation queue controller, the reservation ring controller and the transmitter/receiver. There is also an external interface named SOFI which is shared by all the nodes in the cluster.

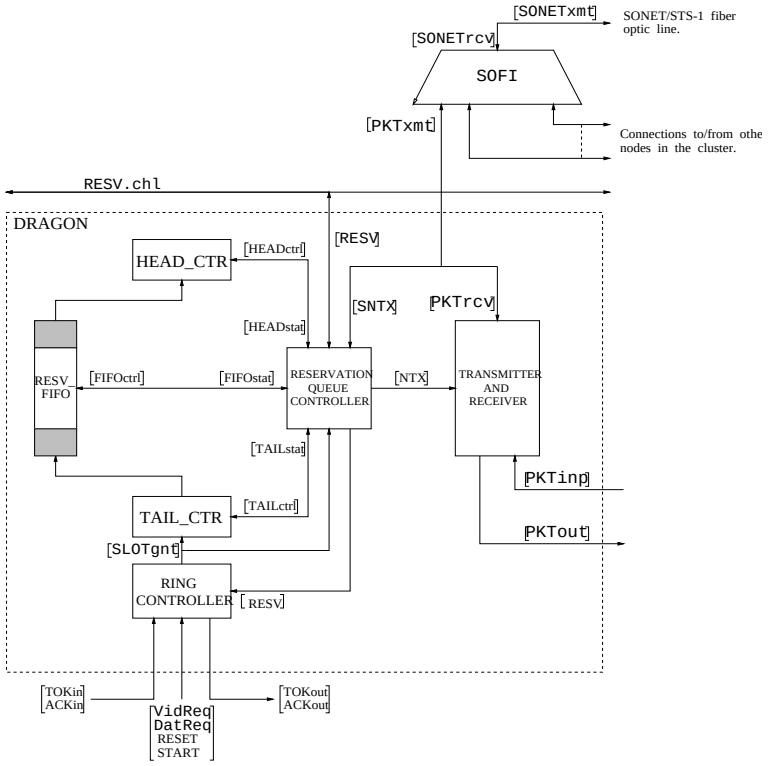


Figure 4: The DRAGON prototype and its datapath

- **Reservation Queue:** Reserves FCFS slots pertaining to the order of message transmissions. Contains an up/down counter named TAIL_CTR, a down counter named HEAD_CTR and a FIFO named RESV_FIFO.
- **Reservation Queue Controller:** FSM for controlling the operation of the RESVQ, including assigning transmission slots and updating the queue for scheduling the next transmission.
- **Reservation Ring Controller:** FSM to perform all functions of the token access control. Contains up/down slot COUNTER.
- **Transmitter/Receiver:** Transmits and receives messages to and from the SONET/STS-N network. Messages may consist of a sequence of SMDS cells or FR frame.
- **SONET Fiber-Optic Interface (SOFI):** External interface shared by all the cluster nodes. It performs mapping and demapping of message packets onto SONET frames [20]. It insets stuff (idle) bytes into the SONET payload when necessary. It also broadcasts signal **SNTX** to all the nodes at the end of every message transmission.

In the following sections, we shall provide an overview of each functional unit. But first, we provide an overview of the entire system.

2.4 Systems Overview

A flow diagram depicting the entire operation of the slot reservation procedure is shown in Fig 5. Initially, every node in the cluster waits to receive a **TOKen**. After it is received, the node checks whether it has any messages waiting to be scheduled for transmission. If it doesn't, the **TOKen** is simply passed on to its successor. If, however, a message is waiting, the interface broadcasts the signal **RESV** to all the other nodes in the cluster over the reservation bus. Every node monitors the reservation bus, and when it detects the **RESV**, the RESVQ is updated and a slot is reserved for the message. The **TOKen** is then sent to the successor node, and the process repeats.

If a **TOKen** isn't received, the interface checks whether the signal **SNTX** had been sent by the SOFI. **SNTX** is broadcast to the cluster at the end of every message transmission. If **SNTX** is received, the nodes update the RESVQ to determine which node is to transmit next. Whichever node is the next to transmit, its RESVQ will send the signal **NTX** to its Transmitter. The Transmitter will then begin sending cells/frames to the SOFI. In the following sections, we shall describe each of the 4 functional units of DRAGON and the SOFI using flow-charts. The RESVQ and the RESVQ controller has been grouped together into a single flowchart, and likewise the Transmitter/Receiver and SOFI have also been grouped together.

2.5 The Reservation Ring Controller

The flowchart for the reservation ring (RESV_RING) controller is shown in 6. Each node waits to receive a **TOKen** from its predecessor. It then acknowledges its predecessor by sending the acknowledgement **ACK** signal. The node then checks whether its COUNTER is larger than zero. The COUNTER registers the number of outstanding messages waiting to be allocated a RESVQ slot. When a new message arrives at the DRAGON, the COUNTER is incremented by 1, and anytime it is zero it means there are no pending messages so the **TOKen** simply passed to the successor node. Otherwise, the controller sends signal **SLOTgnt** to the RESVQ and the COUNTER is decremented. The reservation bus is monitored, and after signal **RESV** had been broadcast by the RESVQ, **TOKen** signal is sent to the successor node.

2.6 The Reservation Queue

2.6.1 Reserving a Slot

The flow chart for reserving a slot is shown in Fig. 7. Every node in the cluster checks whether it had received **SLOTgnt** from the RESV_RING controller, and if so, it broadcasts **RESV** over the reservation bus. Subsequently, it increments its TAIL_CTR, writes the TAIL_CTR contents into the RESV_FIFO and clears the TAIL_CTR. The down counter named HEAD_CTR value plus the composite values of all the RESV_FIFO elements is the number of messages which must be serviced before the currently reserved message can transmit.

For example, suppose that all the messages have equal priority. Further, suppose that at this

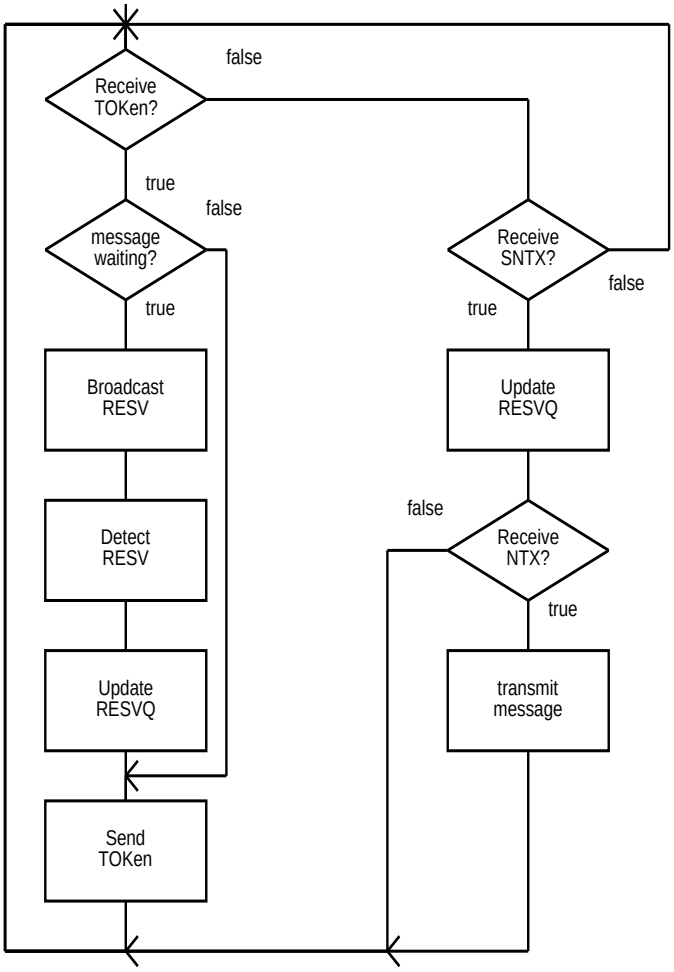


Figure 5: The DRAGON systems overview

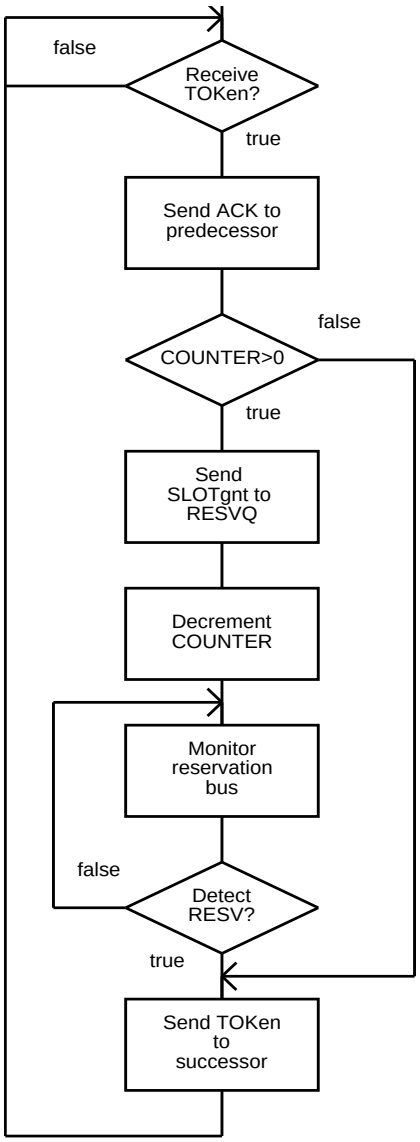


Figure 6: Reservation ring controller

instant HEAD_CTR=5 and RESV_FIFO={3,7,2}, where '2' is the head element and '3' is the tail element. Thus, the newly arrived message will have to wait until 17 $(3+7+2+5)$ messages which had arrived before it to be serviced before it can transmit. In the example, the node had scheduled 4 messages for transmission. This is given by the number of RESV_FIFO elements plus one, if the HEAD_CTR¿0.

Now, if there were two different priorities of messages, as in the case of video and data, slot reservations are still made in the usual way: but the higher priority message will always be transmitted ahead of all the lower priority messages. For instance, if the tail element '3' in the RESV_FIFO is a slot allocated to a high priority message and the remaining 3 are low priority messages, the high priority message will be transmitted when the HEAD_CTR counts down to zero, irrespective of the fact that the slot was allocated to a low priority message. Further, suppose that the node doesn't receive **SLOTgnt**, but detects **RESV** on the reservation bus: it will increment its TAIL_CTR, thereby reserving a slot for some other node within the cluster.

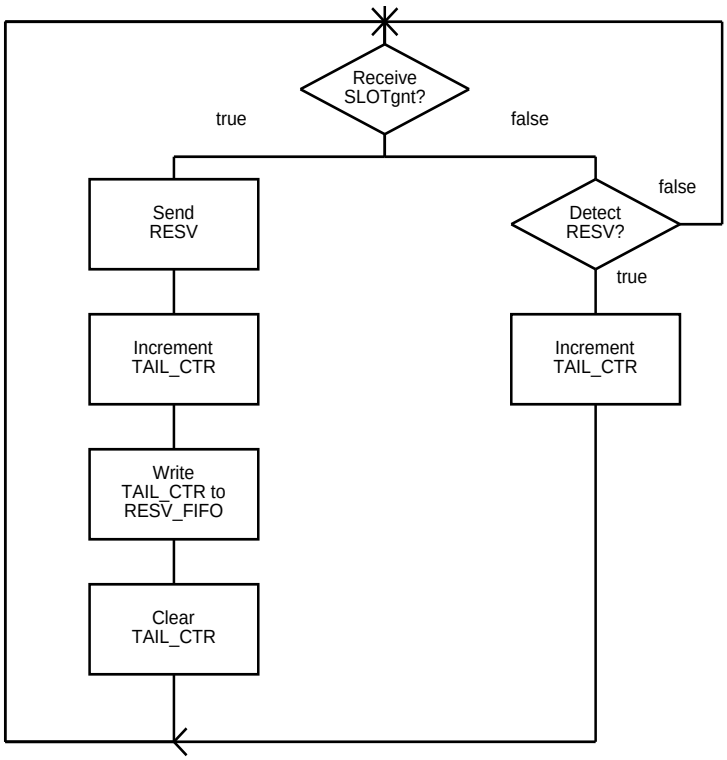


Figure 7: Reserving a slot

2.6.2 Scheduling a Transmission

The flow chart for scheduling a message for transmission is shown in 8. The protocol initiates by checking the value of the HEAD_CTR. First, consider the previous example where HEAD_CTR=5 and RESV_FIFO={3,7,2}. The RESVQ checks whether it has received **SNTX** from the SOFI, and if so, it decrements the HEAD_CTR. The new HEAD_CTR value is now 4 and it signifies the number of other nodes that must be serviced before this node can transmit. Thus, after the

HEAD_CTR reaches zero, signal **NTX** is sent to the Transmitter/Receiver block informing it to begin transmission.

Next, consider the case where HEAD_CTR is equal to zero. The RESV_FIFO is checked, and since its not empty, the top element '2' is loaded into the HEAD_CTR. As before, this value is the number of transmissions that must be made by the other nodes within the cluster before this node can transmit. If RESV_FIFO is empty and HEAD_CTR is zero and **SNTX** is **true**, the TAIL_CTR is decremented if its value is non-zero. When HEAD_CTR and RESV_FIFO are both empty, a non-zero TAIL_CTR signifies the number of messages scheduled for other cluster nodes: but not this node.

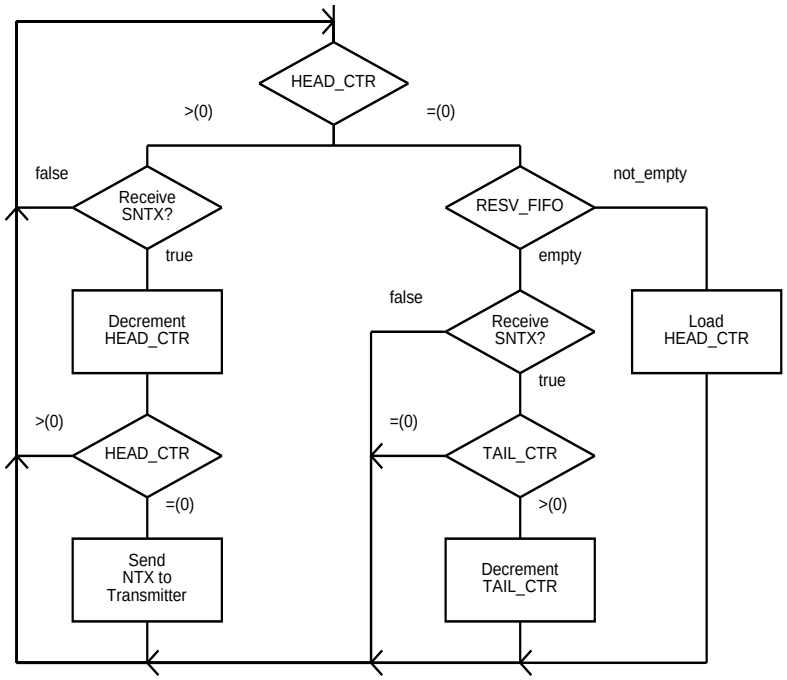


Figure 8: Scheduling a transmission

2.7 The Transmitter/Receiver

The flow chart defining the Transmitter/Receiver and SOFI combination is shown in Fig. 9 and 10. The flow chart in Fig. 9 shows how the SOFI interacts with the nodes for scheduling the transmission of packets. Fig. 10 describes how packets are extracted from incoming SONET frames and transported to the destination cluster node.

2.7.1 Transmitting Packets

Fig. 9 shows the procedure for transmitting packets. After the SOFI receives a message packet from a cluster node, it checks whether the message consists of SMDS cells or FR frame. If the message

is SMDS, then each cell of the message is transmitted to the SOFI, where it is mapped onto the Synchronous Payload Envelope (SPE) of a SONET frame and transmitted over the SONET network. For every cell, its type as indicated by the ST field in the payload is checked. If the cell type is BOM or COM, it means that the message transfer is not complete. Thus, the next cell is transmitted.

If the cell type is SSM or EOM it means that the message transmission is complete, and so the SOFI broadcasts the signal **SNTX** to the nodes to notify them of this condition. Each node will then update its RESVQ by decrementing their respective HEAD_CTRs or the TAIL_CTRs as explained earlier. The node whose HEAD_CTR decrements from $1 \rightarrow 0$ will issue an **NTX** to its Transmitter, and the next message transfer will begin. A similar sequence of events occurs when the message type is FR.

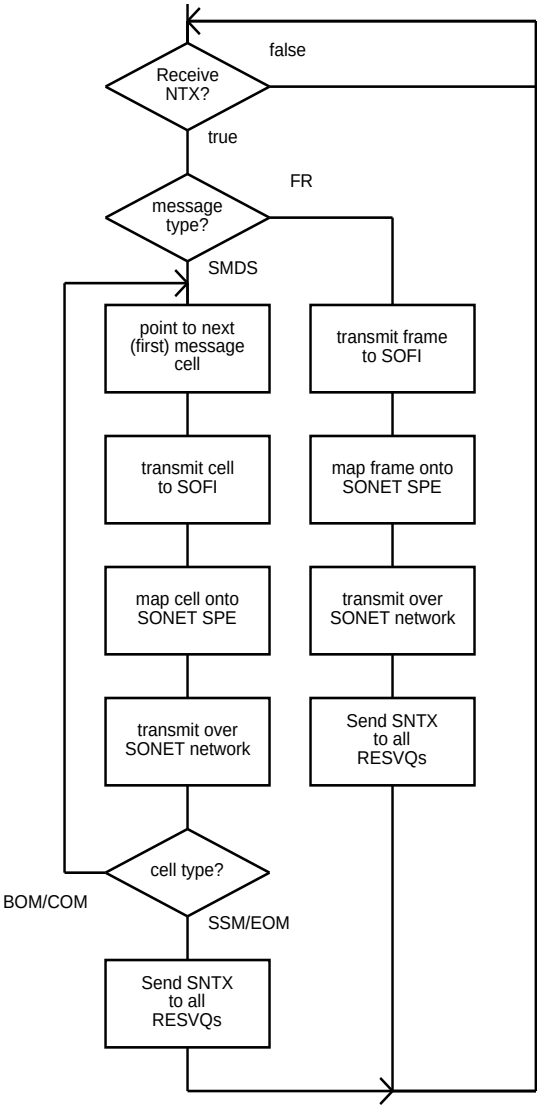


Figure 9: Transmitting packets

2.7.2 Receiving Packets

The flow chart describing the method for receiving packets is shown in Fig. 10. After a SONET frame is received from the switching network, the SOFI demaps the packets from the SPE of the incoming frame. All the stuff bytes are ignored and valid packets are broadcast to all the nodes. The nodes can determine whether it is the recepient of the packet by: (a) SMDS: comparing its ID with the *Multiplexing Identifier* (MID) value of the cells, (b) FR: comparing its ID with the *Data Link Connection Identifier* (DLCI) of the frame. If a match occurs the packet is accepted, otherwise it is rejected.

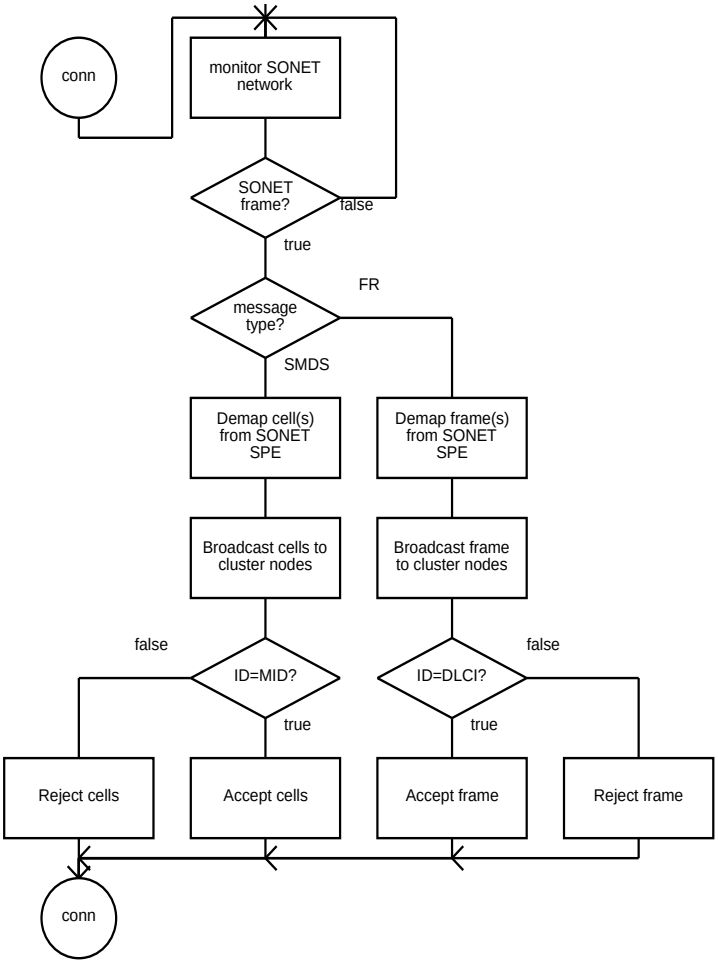


Figure 10: Receiving packets

3 DRAGON Prototype SDL Specification

A complete set of SDL diagrams of the DRAGON prototype is provided in Figs. 3 to 16. These diagrams specify the behavior of the system in a top down manner; starting with the system

definition, then to the level of the blocks, and finally down to the process definitions using Extended Finite State Machine (EFSMs) notation. Each node has its own DRAGON block type. The system CLUSTER consists of the block set DRAGONs containing a number of blocks of type DRAGON. The number of instances of DRAGON is specified by the parameter called NoOfNodes. The CLUSTER also contains the SMDS/SONET interface (SSI) and the BROADCAST block responsible for transmitting signals **RESV** and **TOKen** to all the blocks within the DRAGONs block set. The set of channels S_1 and S_2 are called the reservation bus and the reservation ring respectively. There are *NoOfNodes* of channels within each channel set. Whenever a **RESV** signal is sent, it is broadcast over all the S_1 channels, and likewise the **TOKen** is broadcast over all the S_2 channels. The reservation ring is modeled as an *IEEE 8802-4 token bus* [21], because it was easier to do it this way in SDL.

The SDL protocol for broadcasting the **RESV** signal consists of two stages. In the first stage, one of the RESVQ_CTLR processes sends signal **SLOT** to the Broadcast process. After that, the Broadcast process broadcasts the **RESV** signal to each of the RESVQ_CTLR processes within the DRAGONs. In order to address each RESVQ_CTLR process individually, its PId must be known. The PIds were obtained after consuming the **Id1** signals and then applying the PId-expression **sender**. Each RESVQ_CTLR process instances sends **Id1** to the Broadcast process immediately after it has executed the *start* symbol. The Broadcast process stores the PId values in array IdArray1.

The SDL procedure for passing the **TOKen** signal around the reservation ring works in a similar way to above. As mentioned earlier, the **TOKen** passing scheme was implemented in SDL using the token bus protocol. This works by broadcasting the **TOKen** signals to all the RING_CTLR processes within the DRAGONs block set. The **TOKen** conveys the PId of the next RING_CTLR process which is designated the **TOKen**. After consuming the **TOKen** signal, each of the RING_CTLR processes checks whether this PId matches its own PId value. If a match occurs then the **TOKen** is accepted, otherwise the **TOKen** is rejected. As with the RESVQ_CTLR process, the PIds of the RING_CTLR processes must also be known by the Broadcast process, and likewise they are extracted after consuming the signal **Id2**. The RING_CTLR process PIs are stored in array IdArray2.

The DRAGON block type contains the main components of the SMDS user-to-network interface. These consists of the reservation queue (RESVQ) and the cell transmitter/receiver (TX_RCV) blocks. SMDS cells enter the TX_RCV block in sequential order over the **SMDSinp** channel. Received cells which are destined for the node are accepted and sent via **SMDSout** channel for reassembly. Although the DRAGON has been modeled for the SMDS protocol, it can be adapted for other protocols also such as ATM, X.25 and frame relay. This merely involves respecifying the TX_RCV block for the desired transmission protocol. The RESVQ block remains unchanged.

The RESVQ block is shown in Fig. (a). It contains three basic elements: an up/down counter named tail counter (TAIL_CTR), a down counter named head counter (HEAD_CTR) and a FIFO named reservation FIFO (RESV_FIFO). When a new message arrives at the Transmitter, a **SLOTreq** signal is generated to request the reservation of a slot in RESVQ for the message. This operation is controlled by two finite state machines named the reservation queue controller (RESVQ_CTLR) and the reservation ring controller (RING_CTLR).

A part of the slot reservation is the **TOKen** access control and this is performed by the

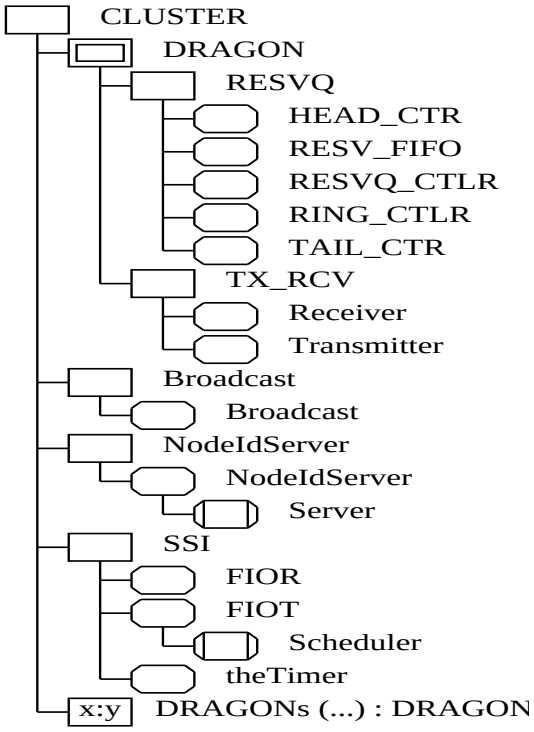
RING_CTLR process. After the RING_CTLR process receives the **TOKen**, it checks if the PId it conveys is equal to the processes own PId. If it is, it implies that the node is the next one to access the **TOKen** and so the **TOKen** is accepted, otherwise the **TOKen** is rejected. After accepting the **TOKen**, if the process has a **SLOTreq** signal pending, the **SLOTgnt** signal is issued to the RESVQ_CTLR process. The **NxtTokRnd** (Next Token Round) signal is then sent to the Broadcast process so that it can start the next token bus operation.

The RESVQ_CTLR process controls the operations of the HEAD_CTR and the TAIL_CTR. After it receives a **SLOTgnt** signal from the RING_CTLR, the **RESV** signal is broadcast to all the nodes. This is done by the Broadcast process as described earlier. After consuming the RESV signal, the nodes increment their TAIL_CTRs. In addition to this, the node that had issued the SLOT signal also pushes the contents of its TAIL_CTR into the RESV_FIFO. The TAIL_CTR is then cleared.

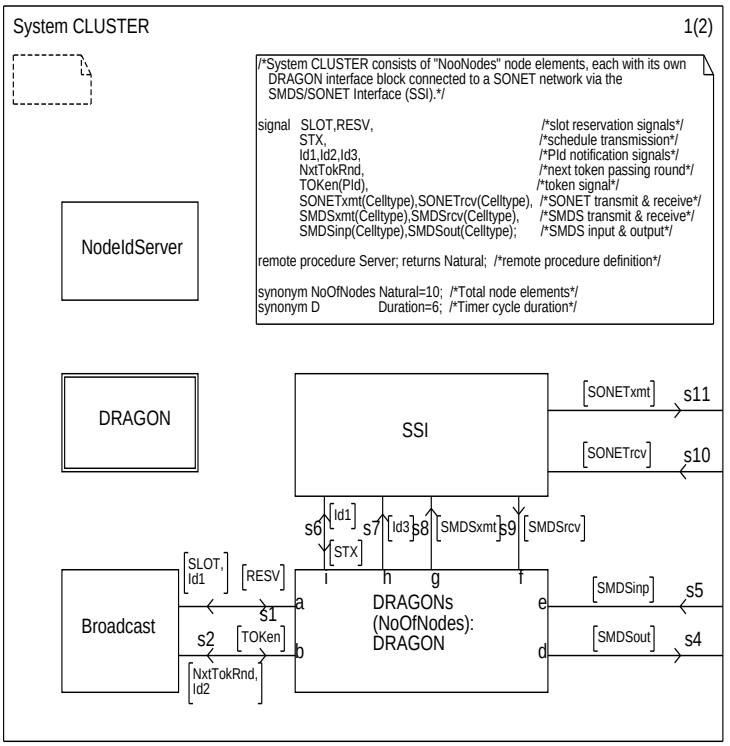
After the transmission of a message cell sequence is completed, the Fiber-Optic Transmitter Interface (FIOT) sends the signal **STX** to all the RESVQ_CTLR processes. Following this, the HEAD_CTRs are decremented if its value is greater than zero, otherwise the TAIL_CTR is decremented if its value is greater than zero. If at this stage, the HEAD_CTR counts down from "1" to "0" the signal **NTX** is sent to the Transmitter process, informing it to begin transmission.

The transmission and reception of cells to and from the SONET channels is performed by the FIOT and FIOR processes respectively. Both these processes reside within the SSI block. We first consider the actions of the FIOT processes. This process starts off by recording the PId values of all the RESVQ_CTLR processes in array IdArray1. As before, the PIds are extracted after consuming the Id1 signals. After this, the process enters the *Xmitcell* state. Then, one of two events may occur. Firstly, the **timeout** signal may be received from theTimer process. This will occur if the timer expires due to inactivity in the **SMDSxmt** channel over a time duration D. This results in a transition whereby the **STX** signal is broadcast to all the RESVQ_CTLR processes. Subsequently, the reservation queue is updated and the next message (if any) is scheduled for transmission. The other possible event is that a cell may be received via the **SMDSxmt** channel. This cell is consumed and retransmitted over the **SONETxmt** channel: the SONET transmit channel. The timer is then "freezed" until the final cell of the message has been transmitted, after which it is again restarted. Finally, the Scheduler broadcasts STX to all the nodes to schedule the next transmission.

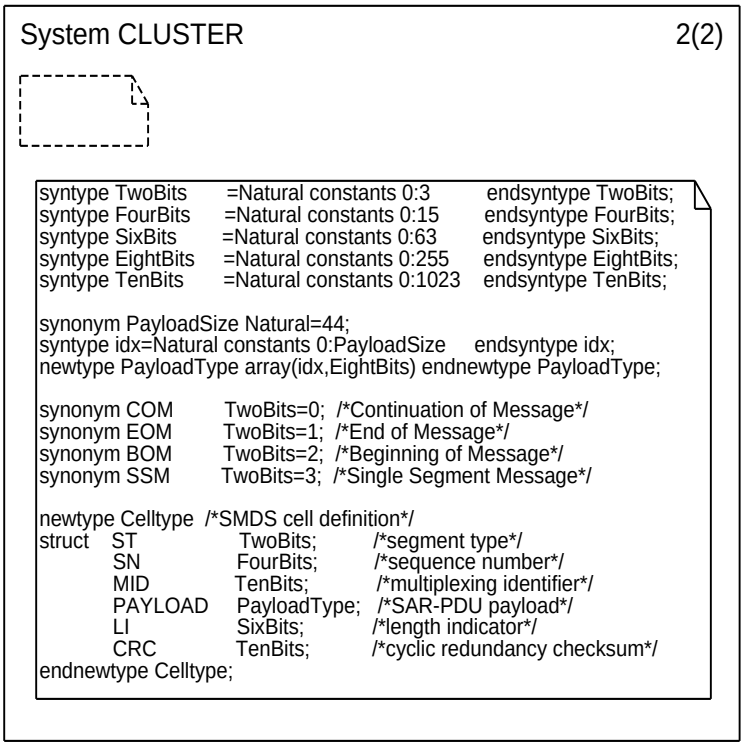
The FIOR process starts off by storing the PIds of the Receiver process in array IdArray3. It then goes to the *RcvCell* state and waits for cells to arrive via the SONET receive channel **SONETrcv**. After receiving each cell, it is broadcast to all the Receiver processes within the DRAGONs block set. To determine if a particular Receiver process is the destination of the cell, each one of them is assigned an identification number called **MyId**. The assignment is done by the remote procedure named Server which returns a distinct integer value to each calling process. The integers are distinct because the calls to the remote procedure are serialized in SDL, and thus it is implied that each calling process will returned a different integer. The **MID** field extracted from each incoming cell and checked whether it matches the node's **MyId** value. If it does, the cell is accepted and sent out via the **SMDSout** channel for reassembly. In case of a mismatch, the cell is rejected.



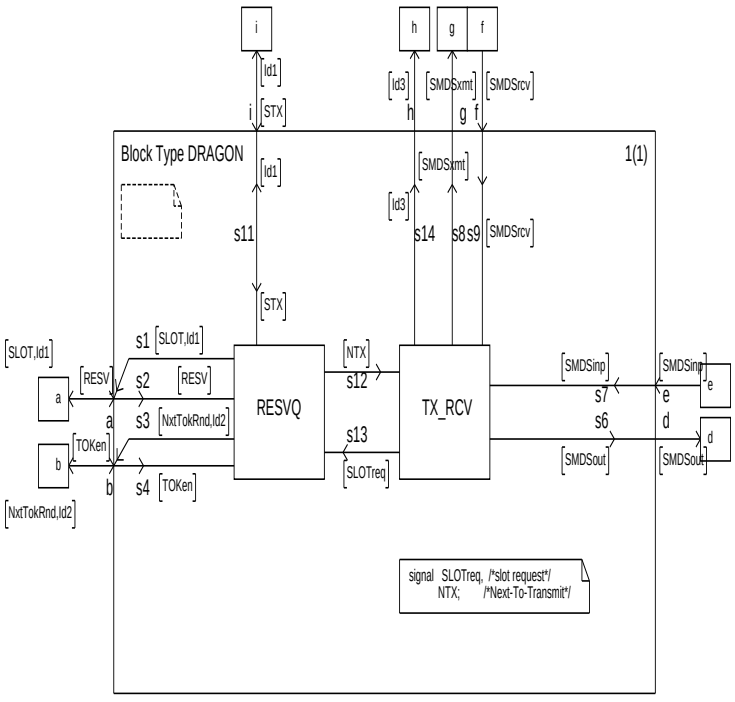
(a) System tree organization



(b) Cluster system

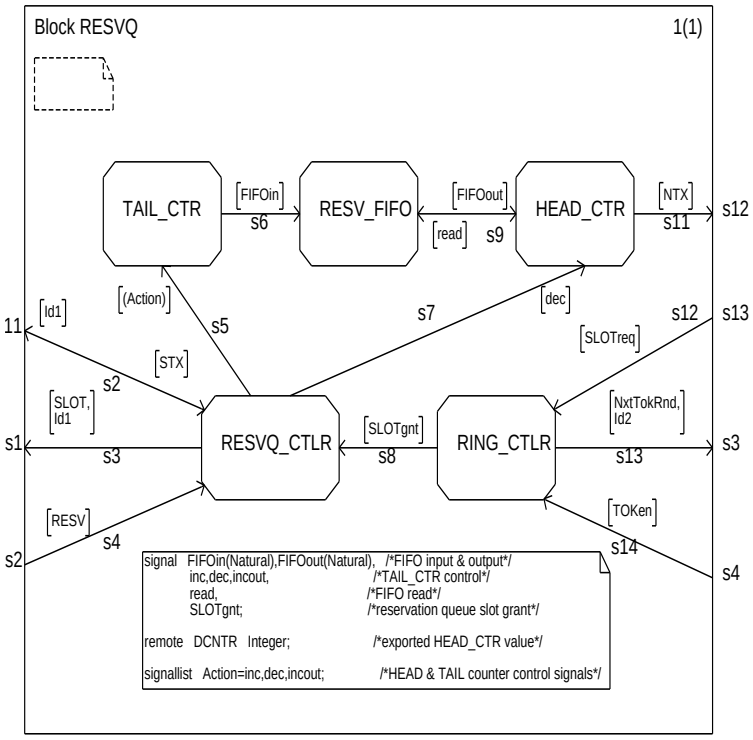


(c) Cluster system

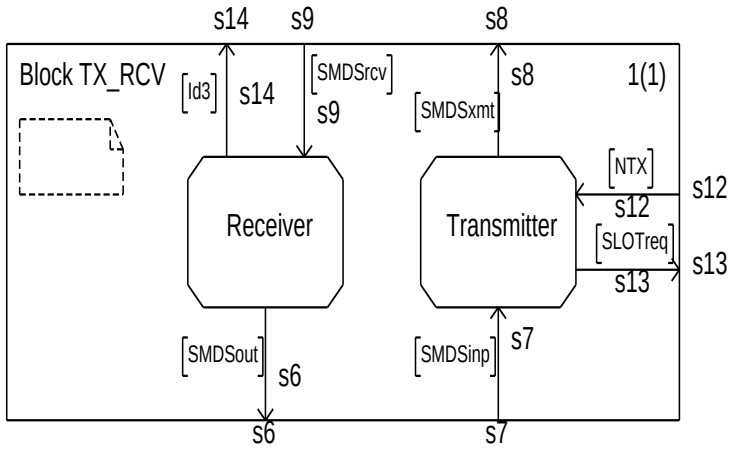


(d) DRAGON block

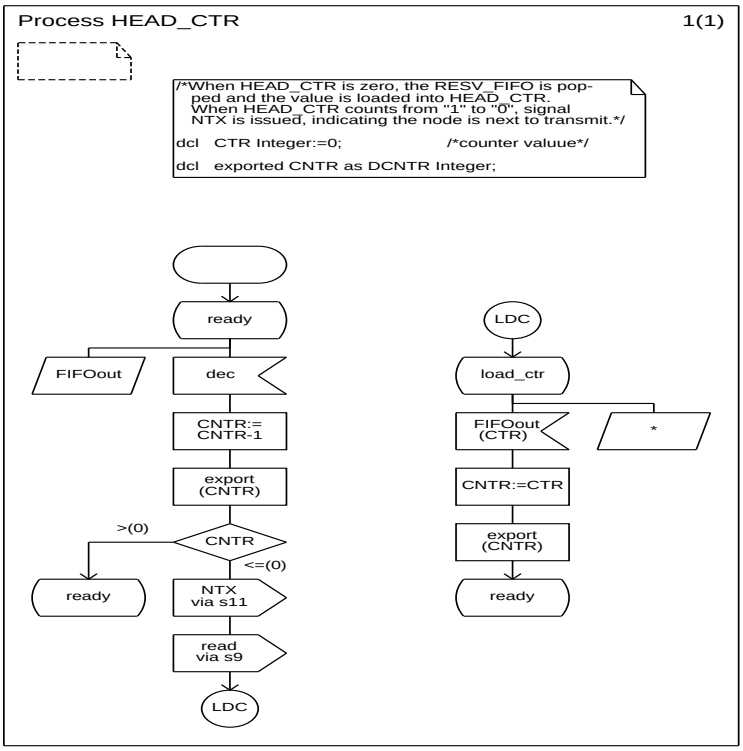
Figure 11: SDL specification diagrams



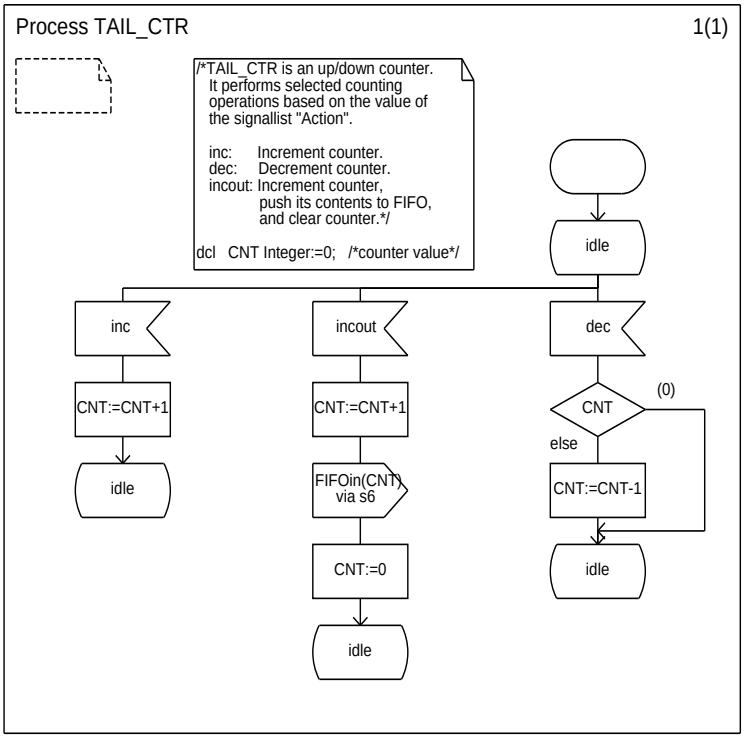
(a) Reservation queue block



(b) Transmitter/Receiver block

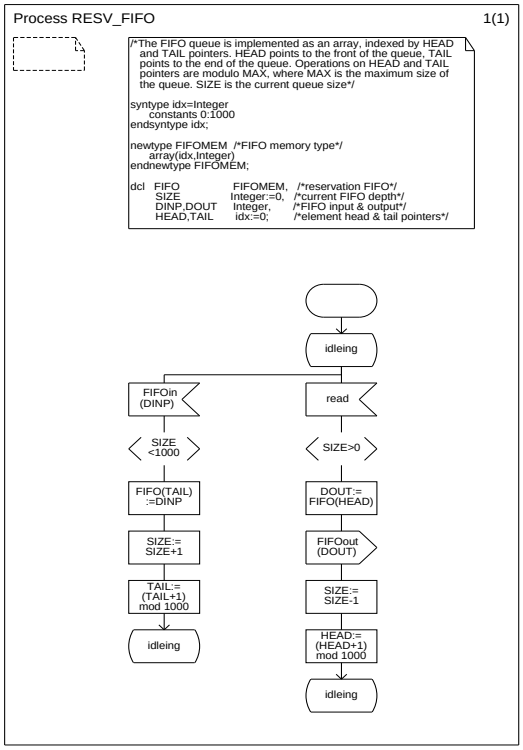


(c) Head counter

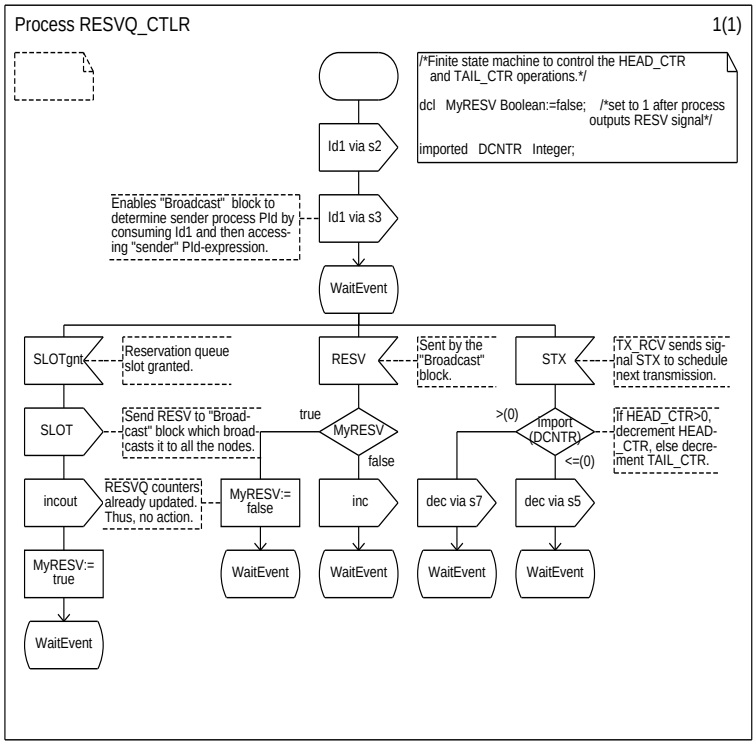


(d) Tail counter

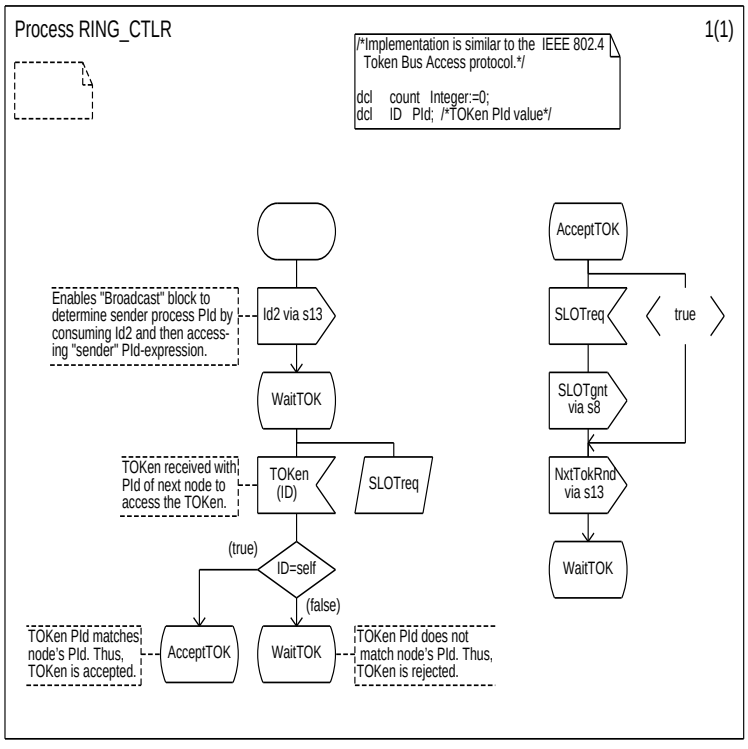
Figure 12: SDL specification diagrams (cont.'d)



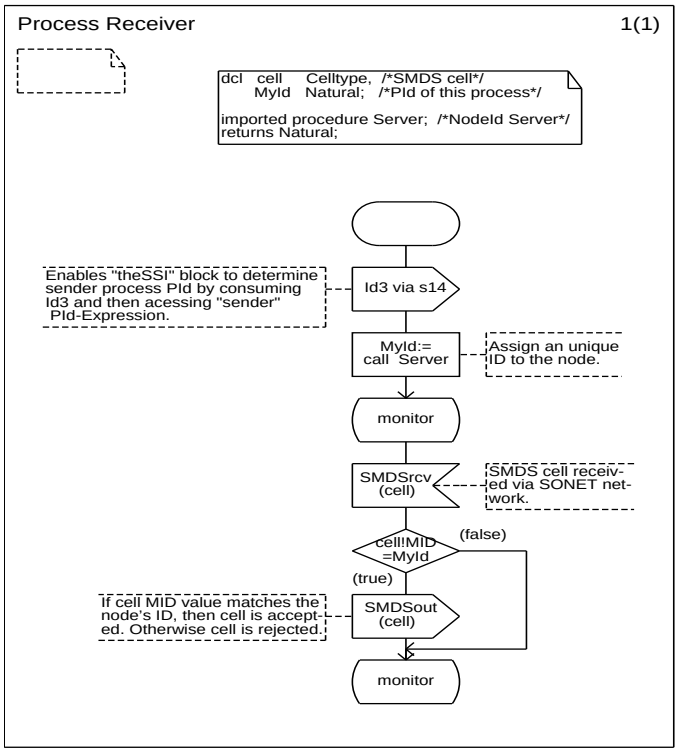
(a) Reservation FIFO



(b) Reservation queue controller

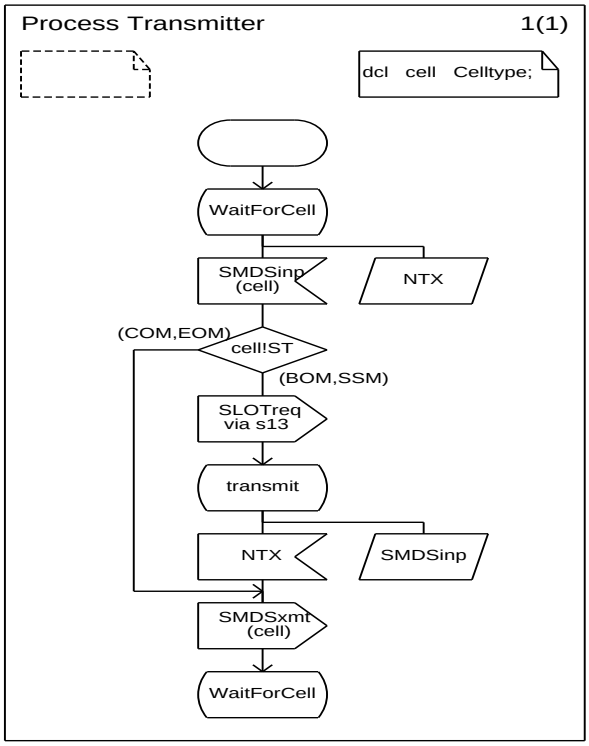


(c) Reservation ring controller

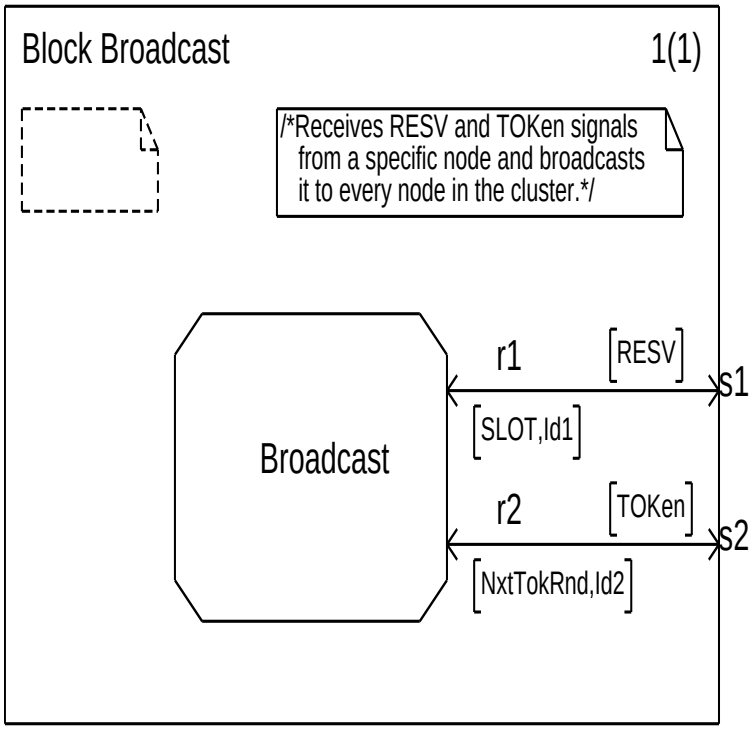


(d) Receiver process

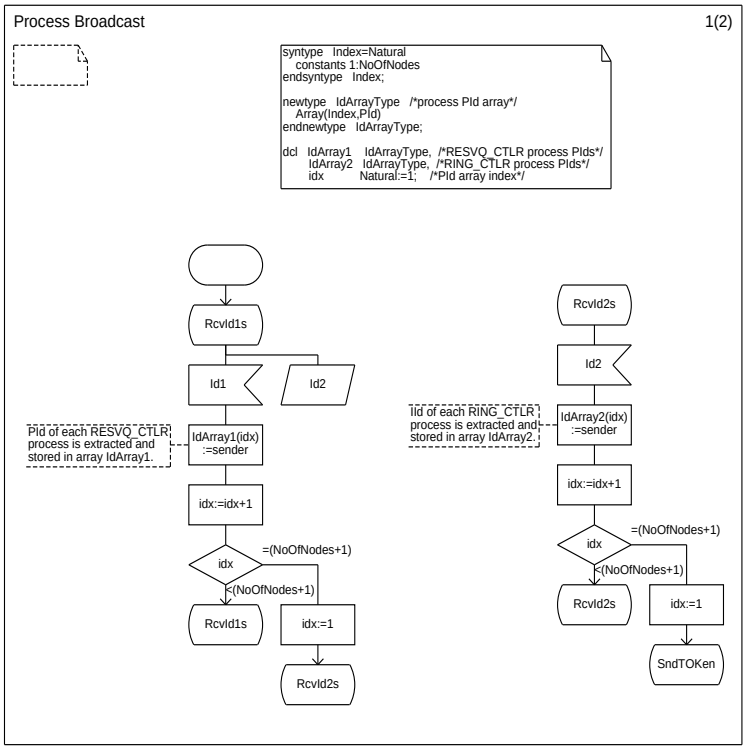
Figure 13: SDL specification diagrams (cont.'d)



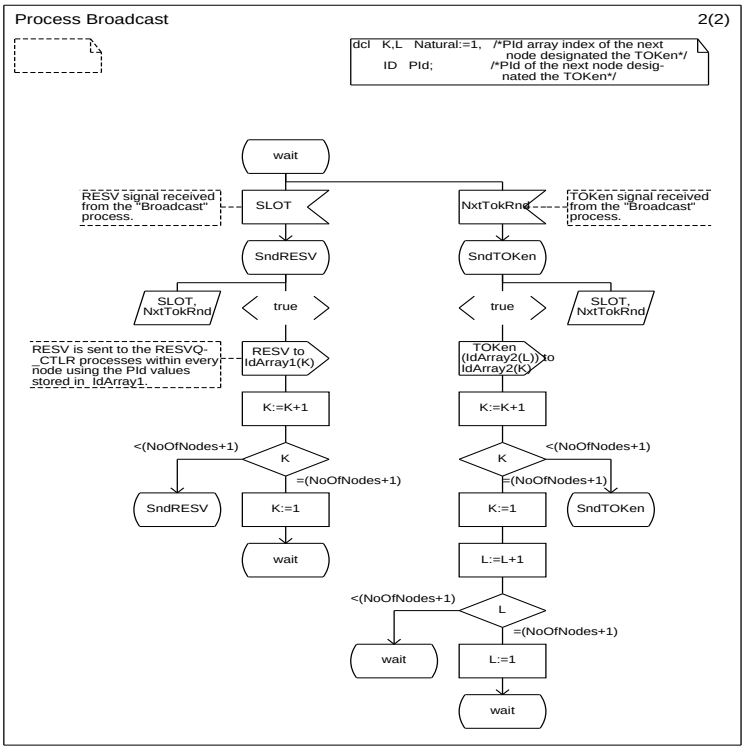
(a) Transmitter process



(b) Broadcast block

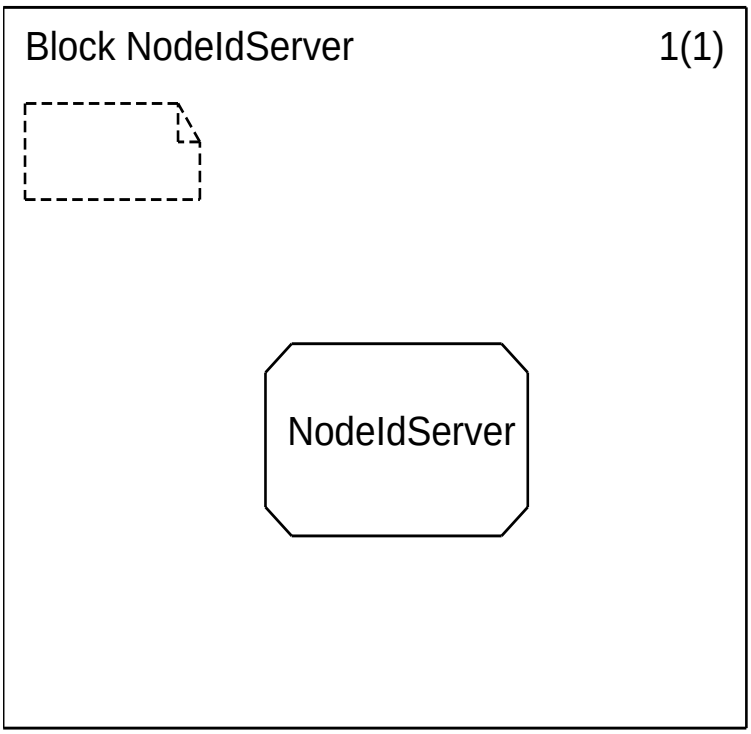


(c) Broadcast process (1)

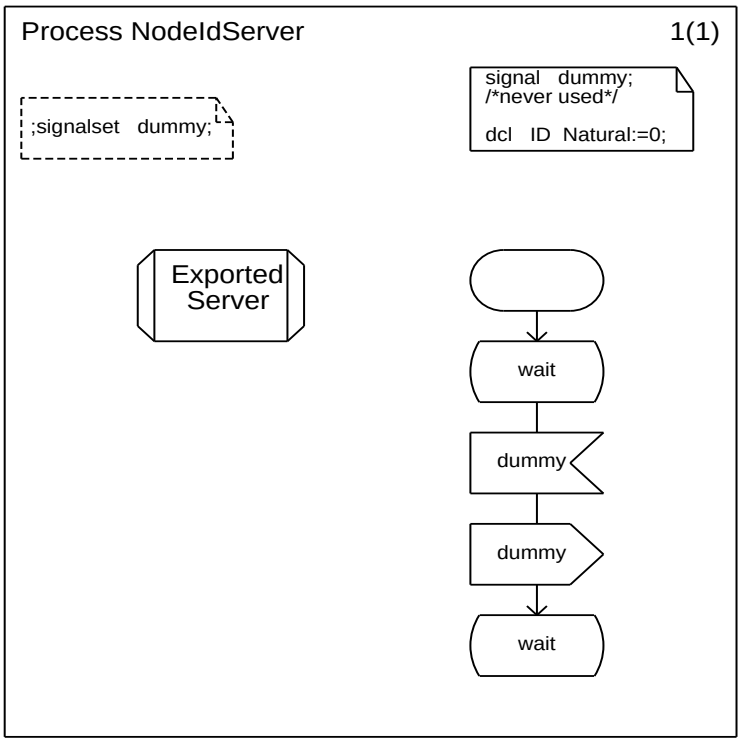


(d) Broadcast process (2)

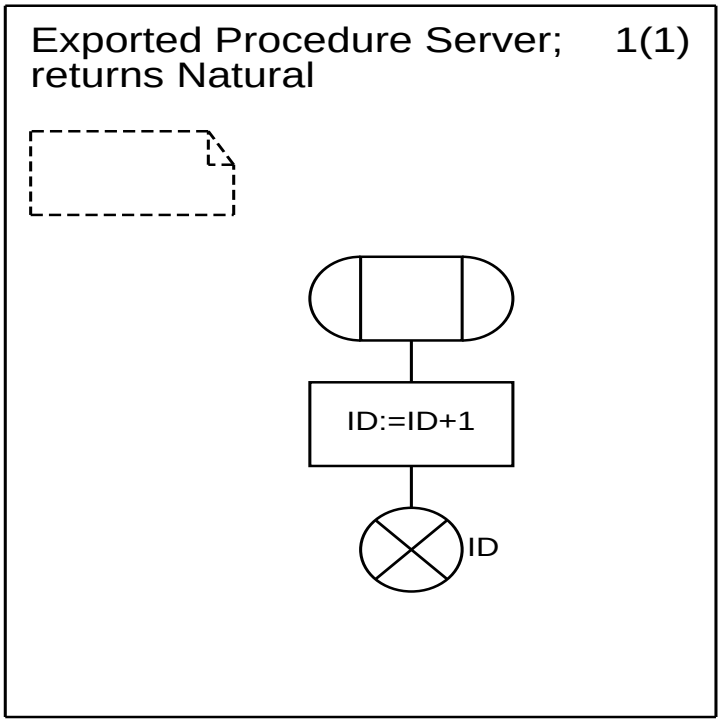
Figure 14: SDL specification diagrams (cont.'d)



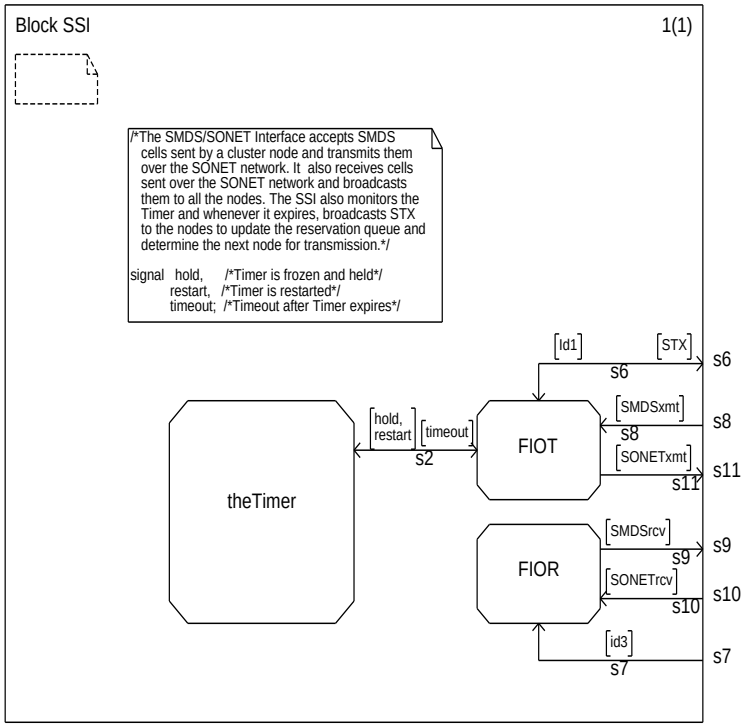
(a) Node ID server block



(b) Node ID server process

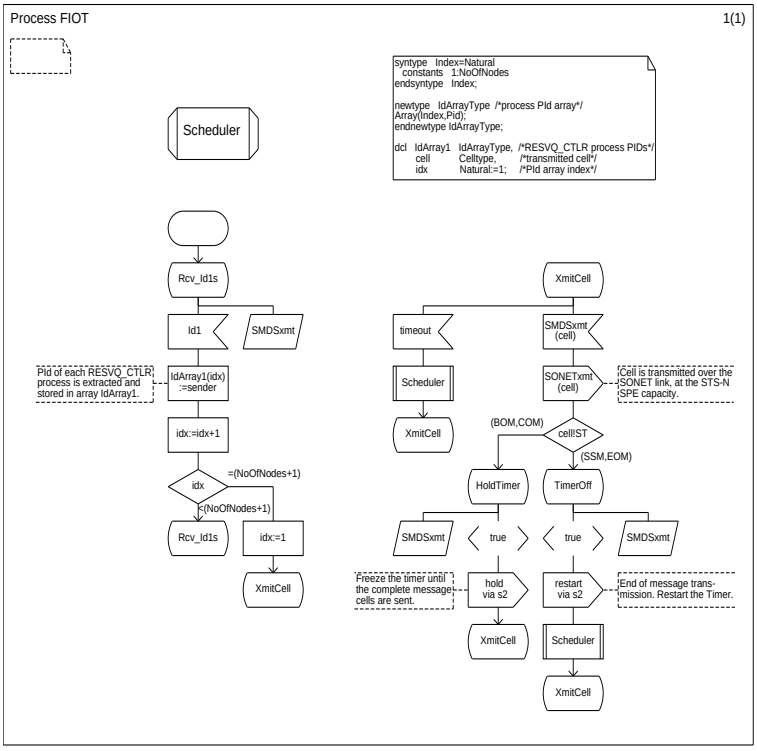


(c) Node ID server procedure

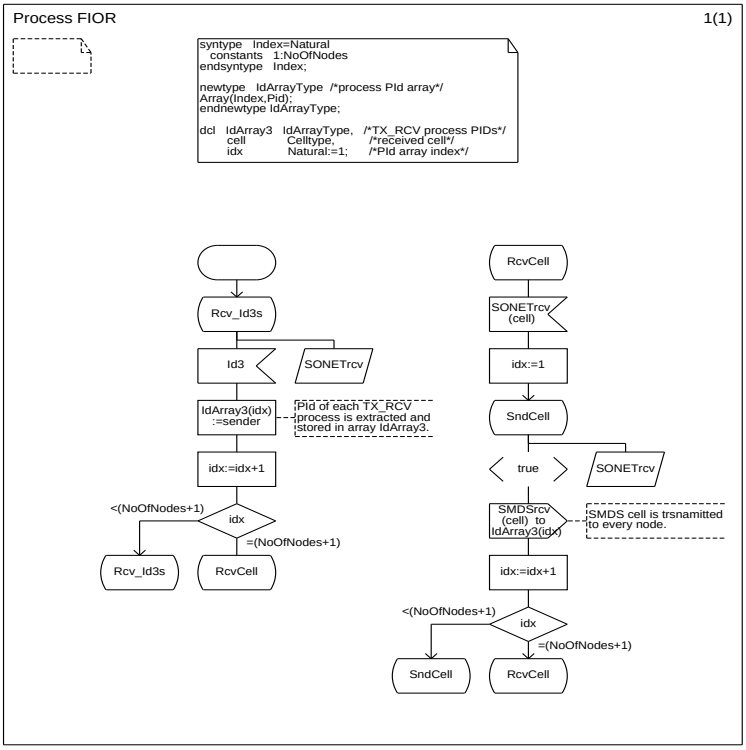


(d) SMDS/SONET Interface (SSI) block

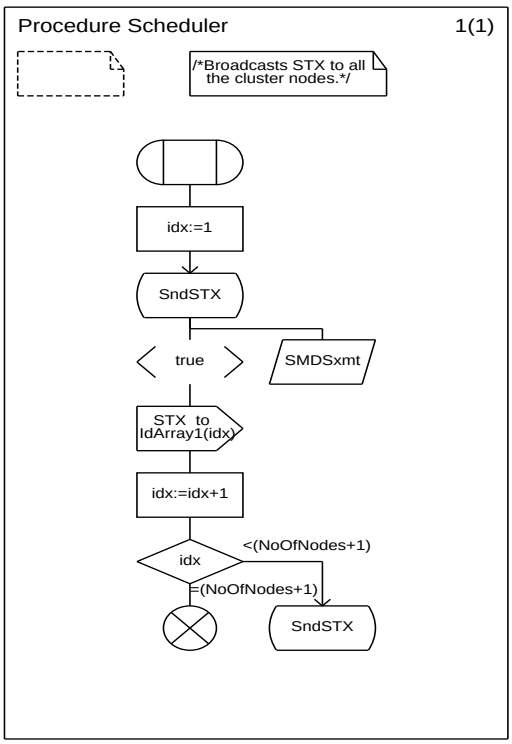
Figure 15: SDL specification diagrams (cont.'d)



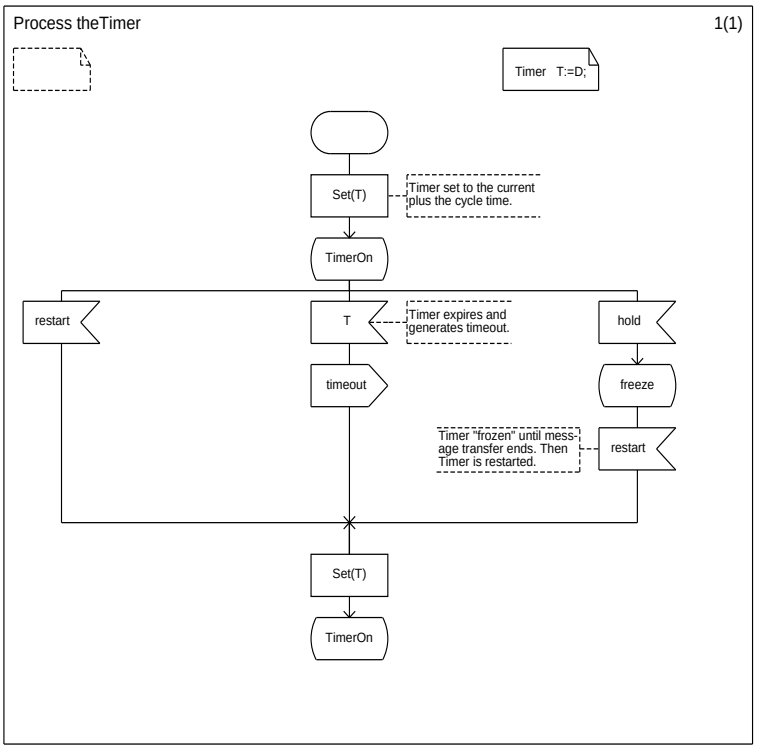
(a) Fiber-Optic Transmitter (FIOT) process



(b) Fiber-Optic Receiver (FIOR) process



(c) Reservation queue scheduler



(d) Timer process

Figure 16: SDL specification diagrams (cont.'d)

4 SDL Validation Methods

In this section we shall describe two algorithms that are most commonly used for validations. Both of them has been implemented in the SDT validator. Before proceeding we shall define some terminology. The *state space* of a system is the composite of all finite state machine states, variables and queue states together with all the combinations of local state transitions. The state space is divided into two disjoint classes: *reachable* states and *unreachable* states, Unreachable states are unexecutable states, and all errors should be limited to within these states. Reachable states are the executable states. All these states must meet a verification criteria consisting of a conformance test and validation criteria. Validation criteria consists of checking the states against a set of correctness rules: no deadlocks, no looping, no range and array index errors, no implicit signal consumption and so on.

4.1 Bit-State

Bit-state is a useful technique for validating large systems of up to 10^8 states. The algorithm uses a storage technique called *hashing*. Initially, all the bits in the hash table are set to '0'. Next, using a depth-first procedure, the state space of the system is generated. For each state, a hash value is computed. if the hash value position in the table is '0', the algorithm changes it to a '1' and continues by investigating the successors of the new state. If the value is already a '1', then it is assumed that the state has already been searched and thus the subtree eminating from that state is pruned. The algorithm backs up one state and continues with the search.

4.2 Random-Walk

The random-walk is used for validating very large systems. It performs the search of the state space by randomly selecting a node from the current level of the subtree and then performing validation up to a given maximum depth. The procedure is repeated at the next level subtree, and so on, up to a specified maximum number of repetitions. The algorithm then backs up to the top of the tree and starts all over again. The advantage of this technique is that is doesn't need to store the state space at all, and consequently leads to a considerable savings in memory.

5 SDL Validation Results

5.1 Conformance Test

This section discusses the conformance test results obtained from simulations of the DRAGON cluster using the SDT. The input and output signals to the system is shown in Fig. 3 (b). There are two input signals named **SMDSinp** and **SONETrcv**, and two output signals named **SMDSout** and **SONETxmt**. All of these signals take a parameter of the type CELLTYPE, i.e. an SMDS cell. The number of DRAGON block instances were limited to 5 in order to be consistent with

the number used in the validations. The test values for the inputs were split into two classes: single cell messages (SSM) and multiple cell messages (BOM,$\cdots$COM$\cdots$,EOM). Both of the system inputs were provided with at least one message from each class. To verify that the cells were received correctly at the outputs and in the proper order, we monitored the outputs **SMDSout** and **SONETxmt**.

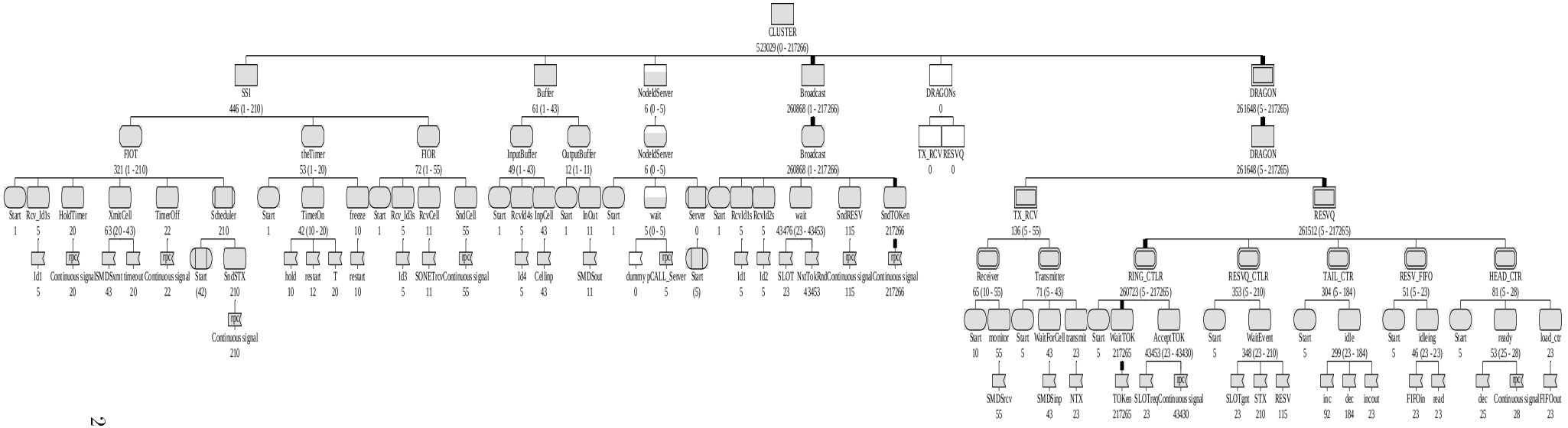
The simulator also provided results of *transition coverage* and *symbol coverage*. Transition and symbol coverages denotes the proportion of the transitions and symbols that have been executed so far. Each transition in the transition coverage tree (Fig. 17) has a number allocated to it to indicate the number of times the transition had been executed. Also indicated in parenthesis beside the state symbols are the maximum and minimum transition frequencies in the subtree. The coverage trees show that all the transitions and symbols have been executed at least once, except for those in process NodeIdServer. This is expected because the **dummy** signal is never used by this process.

5.2 Validation

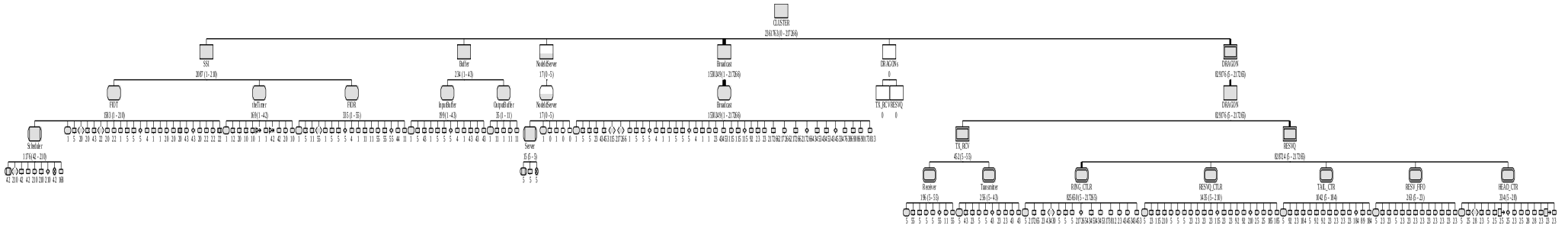
In this section, we shall discuss the results of our validation experiments. We performed two independent validations on a cluster of 5 nodes using the bit-state and random-walk methods. Each validation was executed for a CPU time of 24 hours. The results obtained were compared against the total number of states generated, the size of memory utilized by each algorithm, the average rates at which the states were analyzed and the coverage. Our results are summerized in Table 1.

The random-walk generated more than twice the number of states of the bit-state method. However, not all the states in the random-walk are unique. The bit-state algorithm saves the state space in a hash table, so that if a state had been visited once before, the subtree eminating from that state is not explored again. This saves search time, and guarantees that every bit that is set within the hash table corresponds to a unique state. However, the random-walk does not save the state space. It randomly traverses a specific number of times through the state space, disregarding if the same (unique) state is visited more than once. Hence, the random-walk is likely to traverse through far more "duplicated" states than the bit-state method over the same period of time.

The random-walk method was also twice as fast as the bit-state method as indicated by the efficiency values. The coverage value indicates the percentage of the total symbols executed by each algorithm. They are approximately equal at 96.67 and 97 percent respectively for the bit-state and random-walk. Most of the remaining 3 percent coverage is due to the fact that the **dummy** signal in Fig. (b) is never used, so the transition is never executed. So 3 symbols in each of the 10 instances of DRAGON are not executed at all, thus totaling 30 symbols. If these unexecutable symbols are ignored, the coverage should be closer to 100 percent. Also, the memory size used up by the processes of each algorithm were was larger for the bit-state. This is because the random-walk does not use storage for the state space, and thus utilized considerably less memory.



(a) Transition coverage tree



(b) Symbol coverage tree

Figure 17: Simulation profile for a 5 node cluster

ALGORITHM NAME	UNIQUE STATES ($\times 10^6$)	MEMORY SIZE (megabytes)	RUN TIME ($\times 10^3$secs.)	EFFICIENCY (states/sec.)	COVERAGE (percent)
BIT-STATE	6.93	99.46	86.44	80.17	96.67
RANDOM-WALK	14.50	38.44	86.41	167.61	97.0

Table 1: Validation results for a 5 node cluster

6 Conclusion

In this paper, we developed a novel user-to-network interface for BISDN networks. The interface uses an improved form of distributed queue to schedule messages from transmissions. Access to the network by the cluster nodes are completely fair and is independent of the position of the node in the cluster. Also, unlike DQDB and other similar protocols, the DRAGON is equally suitable for fixed-sized cell transmission (SMDS) as well as variable length packet transmission (frame relay).

We presented a complete set of SDL specifications for a cluster of DRAGON interfaces connected to a broadband network. We focused primarily on the interface segment. The design was extensively simulated using the SDT simulator and the results showed that the system was functionally correct. That is to say, the system generated the correct outputs for numerous combination of test inputs. The transition and symbol coverage showed that virtually every transition and symbol. element within the system had been executed at least once, thus further augmenting the fact that all parts of the system had functioned correctly.

The system was then independently validated using 2 different algorithms. Both methods produced a high coverage, meaning that the system was verifyable to a high degree of probability. In a separate publication [8], we presented an implementation model of the DRAGON cluster using VHDL. Performance studies were done using an integrated mixture of video and data traffic. Finally, it should be noted that creating a complete and correct SDL specification is a ardious task and several incorrect versions had to be modified, but the language and tool proved a very effective way to produce a formal and understandable specification in which we have a high confidence.

7 Acknowledgements

We would like to thank Rick Reed, TSE Ltd., Lutterworth, U.K. for his help, constructive suggestions and feedback on the SDL modeling part of this manuscript. We also thank Gerard Holzmann, Bell-Laborotories, Princeton, New Jersey, U.S.A. for his comments on the validation section of this paper.

References

[1] M. dePrycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN, 2nd. ed.* Ellis Horwood, 1993.

[2] R. Ballat and Y. Ching, "SONET: Now it's the Standard Optical Network," *IEEE Communications Magazine*, pp. 8–15, March 1989.

[3] R. Klessig, *SMDS: Wide-Area Data Networking with Switched Multi-Megabit Data Service*. Prentice Hall, 1995.

[4] U. Black, *Frame Relay Networks: Specifications and Implementations*. McGraw-Hill, 1994.

[5] R. Deasington, *X.25 Explained: Protocols for Packet Switching Networks*. Ellis Horwood, 1986.

[6] ITU-T Recommendation I.413, *BISDN User-Network Interface*. March 1993.

[7] D. K. Sharma and S. R. Ahuja, "A First-Come-First-Serve Bus Allocation Scheme Using Ticket Assignments," *The Bell System Technical Journal*, pp. 1257–1269, September 1981.

[8] S. M. Shahrier and R. M. Jenevein, "A Distributed Access Generic Optical Network Interface for Cell-Relay Networks," *accepted IEEE International Performance, Computing and Communications Conference*, February 1997.

[9] *IEEE Standard Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN), P802.6*. 1988.

[10] G. Watson and S. Ooi, "What Should a Gbit/s Network Interface Look Like," *Protocols for High-Speed Networks*, pp. 237–250, November 1990.

[11] G. C. Watson and S. Tohme, "S++ – a new mac protocol for gb/s local area networks," *IEEE Journal on Selected Areas in Communications*, pp. 531–539, May 1993.

[12] S. M. Shahrier and R. M. Jenevein, "A Performance Comparision of SMDS and Frame Relay Protocols at the DRAGON User-to-Network Interface," *submitted SUPERCOMM*, June 1997.

[13] ITU-T Recommendation Z.100, *CCITT Specification and Description Language*. March 1993.

[14] B. Sarikaya, *Principles of Protocol Engineering and Conformance Testing*. Ellis Horwood Limited, 1993.

[15] K. J. Turner, *Using Formal Description Techniques: An Introduction to Estelle, LOTOS and SDL*. Wiley, 1993.

[16] A. Olsen et. al., *Systems Engineering Using SDL–92*. North Holland, 1994.

[17] G. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[18] G. J. Holzmann, "Algorithms for Automated Protocol Verification," *AT&T Technical Journal*, pp. 32–44, January 1990.

[19] S. M. Shahrier and R. M. Jenevein, "A Distributed Access Generic Optical Network Interface for SMDS Networks," *submitted Computer Networks and ISDN Systems*, October 1996.

[20] B. Kumar, *Broadband Communications: A Professional's Guide to ATM, Frame Relay, SMDS, SONET and BISDN*. McGraw-Hill, 1995.

[21] *IEEE Standard 8802-4 Token Bus Access Method and Physical Layer Specifications*. 1993.