Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering
Shaw Conference Center, Edmonton, Alberta, Canada May 9-12 1999

Description and Validation of the In ternet Stream Protocol (ST2+) Using SDL/MSC *

T. H. Bui, T. O. Elshabrawy, F. Khendek, S. Tahar and T. Le-Ngoc
Departmen t of Electrical and Computer Engineering
Concordia Univ ersity
1455 De Maisonneuve W.
Mon treal, QC, Canada H3G 1M8
email: {hy, tallal, khendek, tahar, tho}@ece.concordia.ca

Abstract

In this paper, we propose a formal description of a large subset of the Internet Stream Protocol version 2 (ST2+) protocol using the Specification and Description Language (SDL) and Message Sequencœ Charts (MSC). ST2+ is an experimental resourœ reservation protocol developed by the Internet task force. The checking of general properties of the model were performed using the ObjectGEODE tool. We have found that all the ST2+ protocol properties that we tested succeeded the verification. In addition, we have found that the use of SDL/MSC simplifies the description and validation of the protocol, and facilitates the possibility for future extensions of our model.

1 Introduction

Internet Stream Protocol version 2 (ST2+) [1] is an experimental resource reservation protocol intended to provide end-to-end real-time applications over the Internet. It allows applications to build multi-destination simplex data streams with a desired quality-of-service (QoS). ST2+ is a connection-oriented internetworking protocol that operates at the same level as connectionless Internet protocol (IP). ST2+ relies on reservation of bandwidth for real-time streams across network routes. This reservation, in conjunction with the appropriate accessing and scheduling mechanisms in all nodes running the protocol, guarantees a well-defined QoS to ST2+ applications. Its main application areas are real-time transport of multimedia data and distributed simulation/gaming, across internets. Though ST2+ assumes that data transmission is unreliable, for such applications, partially correct delivery of data is acceptable while retransmission of erroneous data is not.

Streams form the core concept of ST2+. They are unidirectionnally established from a sending origin to one or more receiving targets in the form of a routing tree. Each node in the tree represents an ST agent, i.e., an entity executing the ST2+ protocol. Any node in the middle of the tree is called an intermediate agent or router. An agent may have any combination of origin, target, or routing capabilities. Each ST agent maintains state information describing the streams flowing through it.

An ST application at the origin may create, expand, reduce, change, send data to, and delete a stream. An ST application at the target may join, receive data from, and leave a stream. Figure 1 shows the ST2+ service primitives implemented in this project.

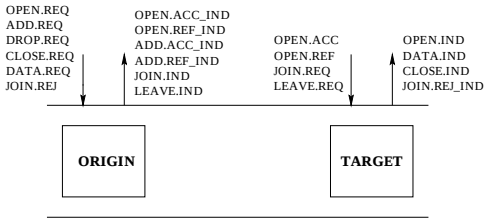

Figure 1: ST2+ service primitives

When a new stream is created, it is necessary to define the join authorization level associated with the stream. This level determines the protocol behavior in case a new target wants to join an existing stream. The available levels are:

- Level 0: targets are not allowed to join this stream.
- Level 1: targets are allowed to join the stream and the origin is notified.

*This work was supported by a grant from the Canadian Institute for Telecommunications Research, under the NCE program of the government of Canada, and by a Scholarship from NSERC.

- Level 2: targets are allowed to join the stream and the origin does not need to be notified.

ST2+ is actually composed of two protocols: ST for the data transport and the stream control message protocol (SCMP) for all control functions. Communication in ST2+ requires two phases: stream setup phase, where the real-time channels are built; transmission phase, where data is transmitted over the established streams. Both ST and SCMP packets use a single PDU format in order to achieve low communication delays. An ST header contains a stream identifier (SID). This SID is selected at the origin so that it is globally unique over the Internet. The SID must be known by the setup protocol as well. At stream establishment time, the setup protocol builds an entry in the local database containing stream information of each ST agent belonging to the stream. The SID can be used as a reference to this database, to obtain quickly the necessary replication and forwarding information. The resulting effect is a more efficient packet forwarding process.

SCMP follows a request-response model. The PDU used in the interaction are as specified in Table 1.

Requesting PDU	Response PDU
CONNECT	ACCEPT or REFUSE
JOIN	CONNECT or JOIN-REJECT
DISCONNECT	no response
NOTIFY	no response
REFUSE	no response

Table 1: SCMP PDU interaction

The reception of all the PDU in Table 1 is acknowledged with either an ACK or an ERROR PDU. SCMP messages are made reliable through the use of retransmission after timeout. The protocol specifies that each PDU has a timer (for acknowledgment and response as needed) that will timeout for a predetermined number of times after which predefined actions are taken.

In this paper, we will concentrate on the modeling of the ST and SCMP protocols, as our goal is to verify and validate the ST2+ protocol behavior. The routing function will be implemented, but it will only carry the strict minimum to emulate a network of networks. In addition, we have modeled the ensemble of routers by one router.

In the next section, we will discuss our design model. We will start with a brief discussion on message sequence charts (MSC) and finite state machines (FSM). We will then present our specification description language (SDL) model based on these FSM. In Section 3, we introduce the scenario case study and the obtained results. Finally, we will discuss our findings and give our conclusions in Section 4.

2 SDL Model

We chose SDL [3] as the formal technique to model the FSM. SDL is an ITU standard formal description language that is used to describe systems using graphical representations as well as textual representations. SDL describes a system as a set of processes. Each process is an extended FSM. Transitions in SDL from one state to another is triggered by the reception of a message. For each process, SDL describes the actions it is allowed to take and which events are expected to happen. In SDL, a system is divided into building blocks that communicate using channels. Blocks are composed of processes. Processes (within a block) are connected using routes. Each process has its own infinite queue and is assumed to operate independently from all other processes. Detailed descriptions of SDL and how to model a system using SDL can be found in [6] and [7].

For the modeling of ST2+ protocol, we begin with MSC [4] to describe the protocol behavior as a sequence of information exchanges. Figure 2 shows an example of a MSC used for ST2+.

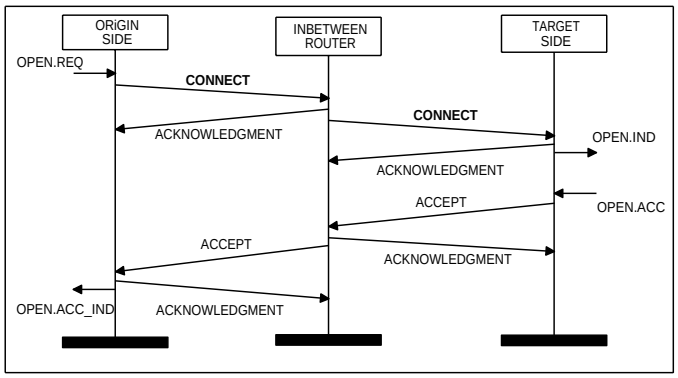

Figure 2: Successful stream establishment MSC

From the MSC, we derived by hand the equivalent FSM for the origin, target and router processes (Figures 3, 4 and 5).

Our SDL model is based on the FSM shown in Figures 3, 4 and 5 and is comprised of three stations, each connected to the router in a star topology. The model is fully expendable to accommodate a larger population of stations (requiring minor modifications, we chose three stations for simplicity). Each station is comprised of a monitor module and an ST module (for communication across the network). Given the proper

stimuli, an origin or a target module is instantiated by the monitor module. The router is also composed of a monitor module and an ST module. Again, given the proper stimuli, a connection handling process will be instantiated.

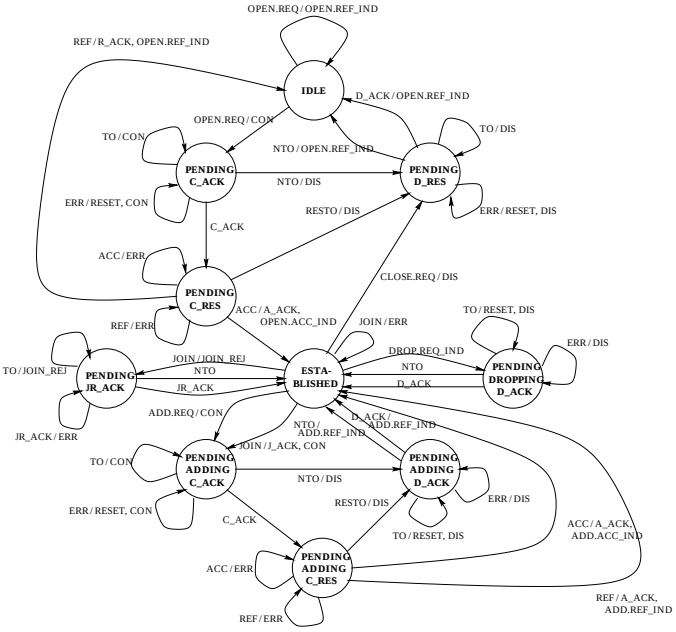

Figure 3: Origin process FSM of ST2+

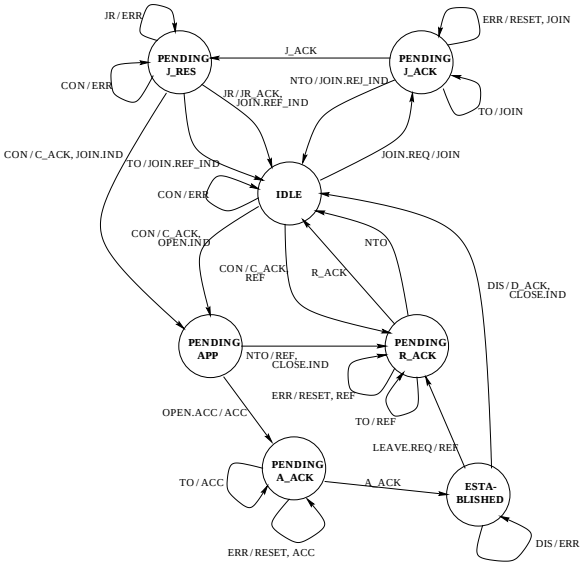

Figure 4: Target process FSM of ST2+

At the beginning of the simulation, the router assigns a unique network address to each station (we

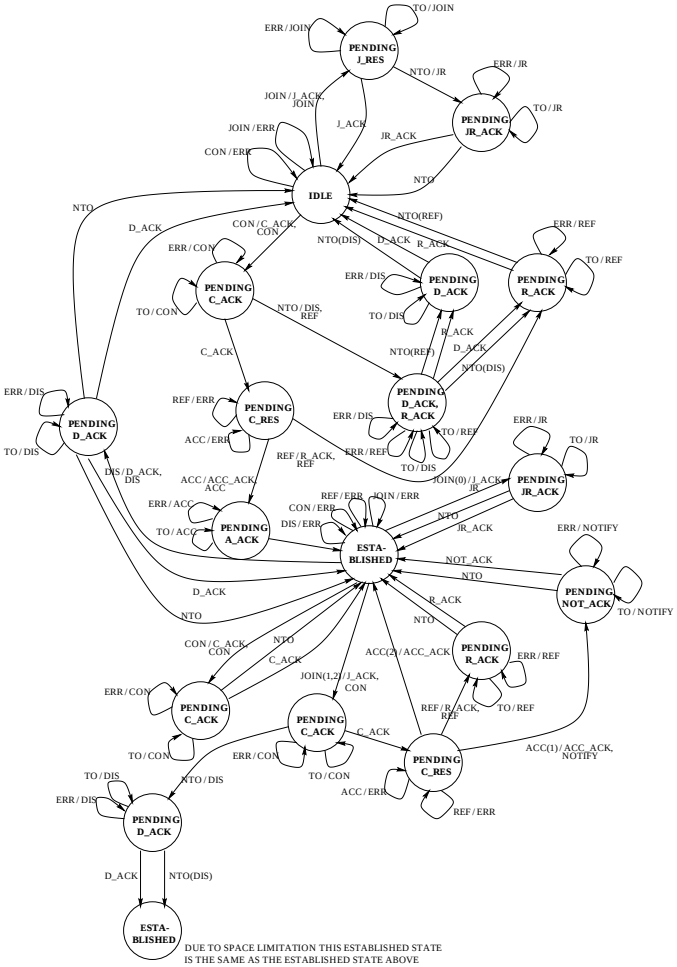

Figure 5: Router process FSM of ST2+

assumed the address of 1000, 2000, and 3000 for the three stations). The system then waits for an upper layer signal. When a station receives an OPEN.REQ, the monitor module of the station immediately detects it and instantiates an origin module. It will also forward an OPEN.REQ to the newly created origin. From this point, the origin will handle autonomously its stream. When a station receives a CONNECT signal, the monitor module of the station consumes it and takes appropriate action depending on if it is part of a new stream setup or a join process. If it is a new stream setup, the monitor creates an instance of a target module and forwards a CONNECT signal to it. If it is part of a join process, the monitor simply forwards the CONNECT signal to the proper target. From this point, the target will be an independent process from the monitor.

2.1 ST system

Figure 6 shows the ST system model. W e assume the ST system to be composed of three stations and a router. Each station is an instance of block type ST and the router of the block type ROUTER. From the figure, we see that an ST block has a medium gate through which it communicates with the ROUTER (thus the ROUTER has three gates, one to each station).

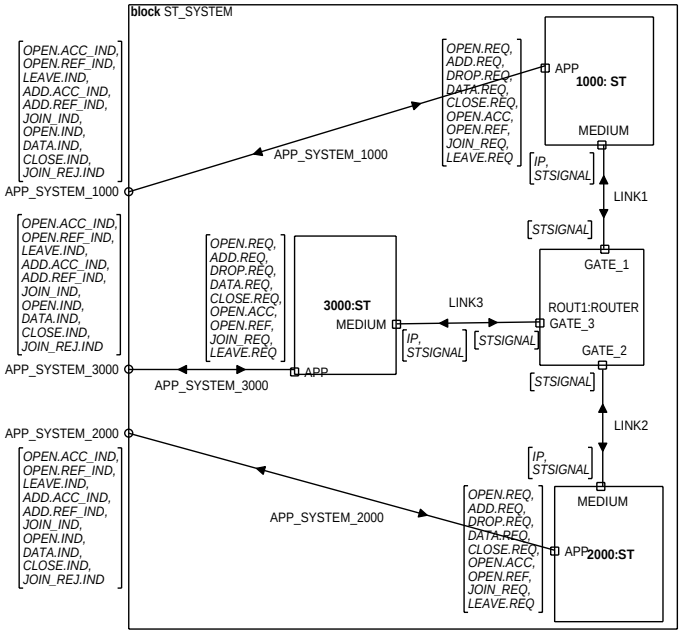

Figure 6: ST system description

2.2 ST block

Figure 7 depicts the ST block, which represents a station. The block type has two gates: the application gate for communication with the application and the medium gate for comm unication with the network. The ST block has four processes:

1. Monitor: it is used for connection managemen t. It keeps track of all the SID in use at its station and it is responsible for resource management.
2. Origin: it is involved in the connection establishment of a stream (associated to a SID) and it carries information about its state (whether it can send data or not).
3. Target: it is involved in the connection establishment of a stream (associated to a SID) and it carries information about its state (whether it can receive data or not).

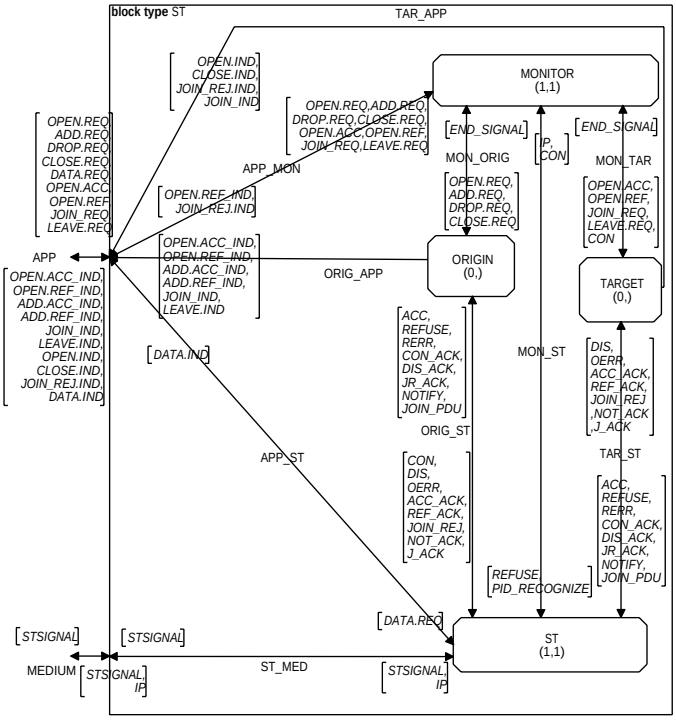

Figure 7: ST block type description

4. ST: it communicates directly with the network. It encapsulates SCMP messages into ST packets to be sent. It is responsible for the actual data transmission.

Figure 8 shows part of the SDL code for the origin process.

Figure 9 shows the ST packet format, which is used by the stations to communicate.

- SID field: it contains the stream identification. Since it is a unique number in the system and since we assume that there cannot be a stream between an origin and a target within the same station, we have also used this value to identify our connection end-point.
- D-bit field: this bit is used to indicate if the ST packet is a control (D-bit set to 0) or a data packet (D-bit set to 1).
- SCMP field: this field carries the SCMP message.
- Data field: this is our representation of a data field.
- Opt field: this is the option field. The only option implemen ted is the join process.
- TL field: it contains the list of all the targets' address.

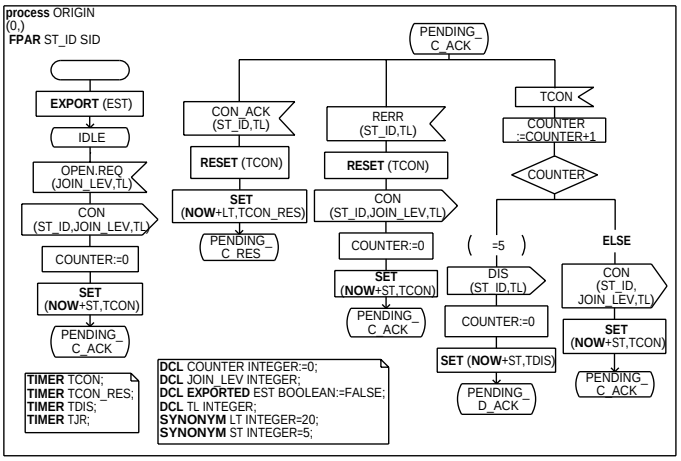

Figure 8: Part of SDL code for origin process

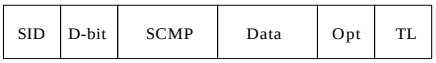

Figure 9: ST packet format

The application communicates with the station through the monitor. Thus the monitor is responsible for the distribution of all the application primitives to the correct module. This is made possible since the monitor instantiates all the required processes.

However, when the monitor receives an OPEN.REQ or a JOIN.REQ, it has to check with the station's resources before taking any action. If there is no resource left, the monitor has the authority to refuse the OPEN.REQ or JOIN.REQ. In the opposite case, the monitor has to instantiate an origin or a target depending on if the signal is an OPEN.REQ or a JOIN.REQ, respectively. After which, the monitor has to forward the application request to the newly created module and to update its SID list and process pointer list.

When a monitor instantiates an origin/target process, it assigns a new SID and relegates all the responsibility of handling the communication process (stream setup, communication with application, etc.) to the newly created module. In turn, when the origin/target completes its communication session, it must send back an end signal to the monitor before it terminates so the monitor can update its list.

The origin/target module is responsible for the stream establishment, to be up-to-date with the state of the stream (can/cannot send/receive data) and update the ST module of any change in the state of the stream.

The ST exchanges primitives directly with the application. When the ST module receives a DATA.REQ, it checks the state of the stream of the corresponding SID and sends data only if the state allows it to carry out that action. When the ST module receives an ST signal that carries data (D-bit set to 1), it checks the target stream state in the same way it did with the origin when it is requested to send data. Thus it will only receive the data if the state of the stream allows it.

The ST module receives SCMP messages from the origin and the target. It then encapsulates it with the appropriate SID, a D-bit (D-bit set to 0), Opt (if it is a join process) and a TL before sending the ST packet to the medium.

When the ST module receives an ST signal carrying an SCMP message (D-bit set to 0), it has to parse it and to forward the message to the correct process.

2.3 ROUTER block

Figure 10 depicts the ROUTER block. The

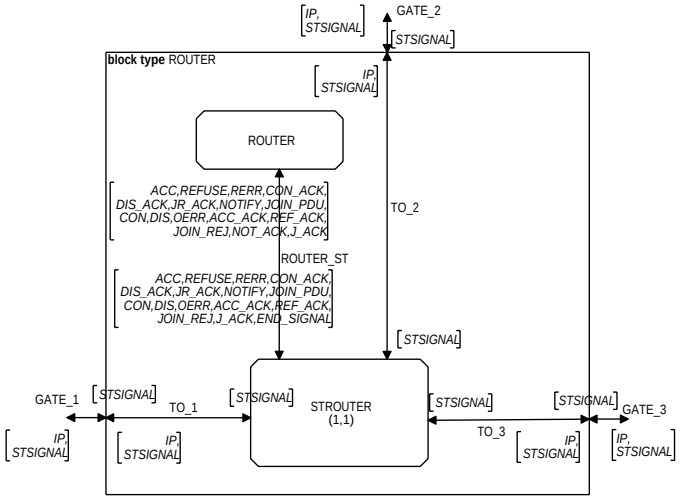

Figure 10: Router block type description

ROUTER has two processes:

1. ST router: it is the ROUTER's monitor which instantiates the router processes. It also forwards data as well as SCMP messages to the correct destination.
2. Router: it handles the connection establishment and termination of a stream.

The ST router exchanges with the medium through ST signals. It forwards the ST signals (data or SCMP) to correct destination.

When the ST router receives an ST signal from the medium, it has to decide whether it forwards it directly to the requested destination (data message

when the associated stream is established) or to deliver it to a router process (SCMP message).

The ST router exchanges SCMP messages with the Router processes. It will instantiate a Router module only if resources are available.

When the ST router receives an SCMP message from one of its Router modules, it will encapsulate it and decide to which gate it should forward the ST packet.

The Router process is responsible for stream establishment, expansion, reduction, etc. We assume that the Router does not implement the multicasting function in our model (for model complexity and protocol validation relevancy issues). We also assume that a stream has to be first established between the origin and one target. After which stream expansion is realized through either an addition of a target by the origin or the joining of a new target. Stream reduction is accomplished through the dropping of a target by the origin or the leaving of a target. Furthermore, since the Router can communicate only to one target, we assume that the Router can correctly close a stream with one target. This assumption will never affect the origin or any given target in the CLOSE or OPEN operations since they still perform according to the specification. However, the Router still has the ability to count the number of targets participating in a given stream. It will terminate whenever the number of targets is equal to zero.

3 Protocol Verification and Validation

As ST2+ is a large protocol, its correctness is hard to verify manually using an informal description. During the last decade, formal techniques have been used successfully for the description and validation of communications protocols [2]. A system is correct if it satisfies general as well as specific properties. General properties of a protocol include the non-existence of deadlocks, unspecified receptions, livelocks and unreachable states. Specific properties, on the other hand, are specified by the user. MSC are among the most popular methods to describe the specific properties.

We have verified the SDL model of the ST2+ protocol using ObjectGEODE [5] validation tools against the general properties, mainly deadlocks. In this section we will simulate a scenario case study to validate our design against some of the specific properties of the modeled ST2+ protocol. We will also discuss the results obtained.

We have chosen the following scenario to validate the model:

1. Establish a stream between station 1000 and station 2000.
2. The origin at station 1000 adds a target at station 3000 to the established stream.
3. The origin at station 1000 drops the target at station 2000 from the stream.
4. A target at station 2000 asks permission to join the established stream.
5. The target at station 3000 leaves the stream.
6. The origin closes the stream specifying station 2000.

With the above scenario, we have covered all the protocol primitives specified as well as all important scenarios, but not all possible scenarios. Furthermore, we have decided to send data along the stream every time a protocol primitive is tested to verify that the data is being properly routed through the network to the targets on the list.

We generated a number of MSC for this scenario case study to check the system protocol functionality at each stage of the test. We also traced the exchange of signals between the processes to be certain that the proper messages are being transmitted. For the MSC, we were concerned with checking if:

- The proper signal is being transmitted from the module involved.
- The signal propagates to the correct destination module.
- The signals within the bounds of a station are exchanged only between the concerned processes.

We also check that the monitor can handle a number of streams (origin or target processes) within the station resource limits. We did a similar check for the router's resources. With the same token, we verified the dynamic allocation of the processes' resources, e.g., that resource is freed when a process terminates.

We have verified all of the above mentioned conditions and found that none was violated.

4 Discussions and Conclusions

In this paper, we have modeled and verified an ST2+ system using SDL and MSC. From our work, the following items can be used as benchmarks in the evaluation of SDL as a formal description language for communications protocol modeling:

Simplicity and scalability: in our model, we used a station population of three to facilitate the model design. However, it is fairly easy to expand the model to fully accommodate any number of stations (e.g., the router

has to assign more network addresses).

Ease of model enhancement: in designing our system, we have added at different moment in time the different functions specified in ST2+. This reflected the evolution of our model into one that most closely resembles to the ST2+ specifications. We have found that the structure of SDL eased the process of model enhancement.

Timing issues: we have found that the value chosen for the timers affect the overall behavior of the protocol. This can result in lower performance of the system.

Multicasting: the multicasting function of the router was left out due to the complex description involved in SDL. The main problem was the need for a router to handle several communications with multiple targets which can independently respond differently. This would require the router to make more than one transition from a given state as required by the different targets. However, this is not possible because the router will not be able to keep track of all participating targets. We propose a solution for this case by instantiating a router process with the same SID for each target involved and creating a pointer at the ST router to each of these processes using a two-dimension array. We strongly believe that multicasting represents a strong case study for SDL.

We conclude that SDL is a simple and very efficient method to verify and validate protocols. Furthermore, SDL allows an easy extension of the model to accomodate any features that might be added to the protocol specification. Finally, we identified that timing in SDL is a hot research topic which is currently investigated.

References

[1] *Internet Stream Protocol Version 2*, RFC1819, 1995.

[2] E. Clarke and J. Wing, *Formal Methods: State of the Art and Future Directions*, CMU Computer Science Technical Report CMU-CS-96-178, August 1996.

[3] *Recommendation Z. 100 - Specification and Description Language (SDL)*, 1993.

[4] *Recommendation Z. 120 - Message Sequence Charts (MSC)*, 1996.

[5] *ObjectGEODE*, Verilog, Toulouse, France, 1996.

[6] A. Olsen et al., *System Engineering Using SDL-92*, Elsevier Sciences B.V., 1994.

[7] Kenneth J. Turner, *Using Formal Description Techniques, An Introduction to ESTELLE, LOTEOS and SDL*, John Wiley and Sons, 1993.