FORMAL DESCRIPTION IN ESTELLE OF AX.25

Michel Barbeau, VE2BPM
7429 Casgrain
Montreal, Canada
H2R 1Y4
E-mail: barbeau@iro.umontreal.CA

Abstract
A protocol description must be as precise as possible in order to avoid multiple subjective and incompatible interpretations by the different implementers. Natural language (e.g. english) descriptions are a priori easy to understand but they are in general ambiguous. It has been now recognized that formal techniques give rise to more precise and complete descriptions of software. Estelle is such a formal technique specially designed for communication protocols. This paper investigates the use of Estelle for describing AX.25 a link-level protocol for packet-radio.

1. Introduction

The experience has demonstrated that the clearest thing in most natural language protocol descriptions is their unclearness. This fact has motivated protocol designers to develop various description techniques with theoretical foundations. Among those, methods based on Finite State Machines (FSMs) and elements of programming language are very popular because while being forma? they remain relatively easy to understand by any software practitioner. Estelle is a description technique based on FSMs and a superset of Pascal. One of the interests in Estelle is its acceptance by many protocol designers and implementers. Estelle is published by the international organization for standardization ISO[1].

Good introductions to Estelle can he found in references 2and 3. To save space, we will mention here only the main features of Estelle. A protocol is described in Estelle as a collection of modules. Each module is a state machine capable of having memory, this is termed *Extended* FSM (EFSM). Modules can communicate with each other as well as with the environment of the specified protocol over channels (FIFO queues). Interactions between protocol entities (e.g. frames) are communicated in the channels. Decomposition of a protocol entity into modules is usually functional: a module that defines the transition function of the protocol, a module for mapping frames into interactions with the environment,, etc.

The language of Estelle is based on Pascal with extensions adapted to protocol specification. A transition of a module EFSM is coded using the following constructs. The **from/to-clauses** specify the major state change realized by the transition. The dependence of the transition on an input interaction is expressed using a **when-clause.** A transition without a when clause is said spontaneous. A spontaneous transition can be used to describe internal decisions of the protocol. The condition for firing a transition is expressed in a **provided-clause.** A condition is a boolean expression on input. interaction parameters and or on module context variables. The actions of a transit ion are put in a Pascal like **begin-end** block Act ions can be for example assignments to cont ext variables, calls to procedures or Est elle **output** st atements. A complete specificat ion of a simple link -level protocol (the alternating-bit) can be found in references land 2.

The original description of AX.25 packet -radio link-layer protocol is written in english[4]. We intend to demonstrate how a higher degree of formalism can be obtained with Estelle. A short survey of related experiences is given in section 2. Our approach is discussed in section 3. We conclude with section 4.

2. Related Work

AX.25 is an adapted version of ISO's HDLC (or other similar link level protocols such as CCITT's LAPB). Because of its context of use (over HF, VHF or UHF radio channels), AX.25 differs on two main points. First, whereas most link-layer protocols assume a master/slave (DCE/DTE) operation, AX.25 is a symmetrical protocol. The two entities at. the ends of the link are alike, the term DXE is used to refer to both devices. Second, each frame (*U*, *S* as well as *I* frames) always identify both the source and the destination DXEs. Moreover, to allow repeater operation the destination address field may contain a route specification that can be made of up to eight, repeater-addresses.

An at tempt to formalize the procedures of HDLC has been realized by Harangozo[5]. He uses a model incorporating regular grammars and a technique of indexing. Another met hod. using a model closer to Estelle, was used by Bochmann and Chung[6,7]. As in Estelle, they use state machines extended with context variables and Pascal like statements. Their description provided a basis for an implementation[8]. Taylor' presented a structured approach based on the notions of module, FSM and Pascal pseudo-code to describe an implementation of AX.25. His concept, of module interface is close to the concept of module header/body in Estelle. Here we will show the use of the formal description technique Estelle to define the link-level protocol AX.25. A similar experience with Estelle for an application-layer protocol can be found in reference 10.

3. AX.25 Description

A general approach to protocol description is presented in reference 11. The main points that. must be covered are context of the specification, protocol service, internal structure of the protocol, types of exchanged messages and the behavior of the protocol. Those aspects must be defined to a degree necessary to ensure compatibility between AX.25 entities but. not further. To save space, a simplified version of AX.25 will be described. It will support. only the I (Information), RR (Receive Ready), SABM (Set Asynchronous Balanced Mode), DISC (Disconnect), DM (Disconnected Mode) and UA (Unnumbered Acknowledgment) frames.

3.1 General Context

AX.25 is to be used as a level 2 protocol of ISO's seven layers reference model,. Its purpose is to provide error free point -to-point communication channels between geographically distributed packet-radio computer stations. The protocol is designed to operate over low speed high error rate shared radio channels (physical-layer). More details about the context can be found in reference 4.

3.2 Protocol Service

A protocol provides a given service to its users (e.g. communication channels). A protocol service is specified in terms of the command types (service primitives) available to the user. The service primitives are abstractly defined. Their physical realization in a particular environment (e.g. interrupts, procedure calls, etc.) is left. to the programmers.

AX.25 leaves completely open the interface definition between the protocol and its users. There is no explicit. constraint, on the structure of the messages and their relative ordering at this interface. The protocol defines only the rules for exchanging blocks of data (frames) between AX.25 entities over the physical layer. This is also the case with specifications of HDLC and LAPH. We will therefore reflect this designer's decision by leaving complete freedom on the service specification to the implementer.

3.3 Structure of the Description

We structure the functions of the protocol into two modules (see figure 1), namely module **Abstract Protocol** (AP) and **Transmitter/Receiver** (TR). The AP module contains the procedures of the link-level protocol but. it makes abstraction of frame encoding in terms of bits and bytes. The functions realized in the TR module are i) encoding/decoding of frames, ii) adding/removing of leading and trailing flags, iii) computing and adding FCS (Frame-Check Sequence) and iv) discarding frames received with incorrect FCS, invalid frames and improperly addressed frames. We will only consider the definition of the AP module.

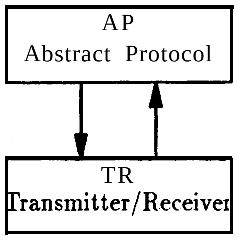

Fig. 1 Structure

3.4 Data Structures

Blocks of data exchanged between AX.25 protocol entities are called frames. There are three basic types of frames: *Unnumbered*, *Supervisory* and *Information*. Each frame consists of a field sequence.

At the level of the AP module we make abstraction of frame encoding in terms of bits and bytes. This task is left to the TR module. The interface between the AP and TR modules is defined by the following channel.

```
channel Abstract_Frames_Channel( User,Provider);
  by User out_frame( frame:frame_type);
  by Provider in_frame( frame:frame_type);
```

The role of *user* will be assigned to the AP module, whereas the TR. module will have the role of *provider*. A Pascal variant record is used to the define the frame data type.

```
frame-kind = (I,S,U);
frame-type = record
  case tag:frame_kind of
    I:( if:I_frame_type);
    S:( sf:S_frame_type);
    U:(uf:U_frame_type);
  end
```

The list of fields making up each frame kind is defined with a record data structure. For example the *I* and *U* frame kinds and their fields are defined as:

```
Address-type = array [1..7] of char;
Addressfield = record
{ repeater operation is not supported }
  destination, source:Address_type;
  end;
 I_Control_field = record
   NS,NR:0..7;
   P:boolean;
   end;
 U_Control_field = record
   MM:(SABM,DISC,DM,UA);
   P:boolean;
   end;
 PID_field = ( AX25,IP,Addr_res,no_lay_3,esc);
 Info-field = record
   data: array [0..255] of char;
   length :integer;
   end;
 I_frame_type = record
   Address:Address field;
   Control:I Control_field;
   PID:PID_field;
   Info :Info_field;
   end;
 U_frame_type = record
   Address:Address_field;
   Control:U_Control_field;
   end;
```

S frame structure can be similarly defined. The encoding/decoding procedures in the TR modules would be defined based on the information provided in reference 4. While the encoding rules must be strictly respected by the implementer, we generally try to leave as much freedom as possible on the sequence of operations to achieve the desired encoding. It is therefore expected that a large part of the TR module would remain more **or less** informally specified.

3.5 Abstract Protocol

In the AP module we will define actions in response to frames from the peer entity and actions internally initiated. The interaction point with the TR module is named *fc* and the channel used at this point is of type *Abstract_Frames_Channel*. The module header would therefore look like:

```
module Abstract-Protocol
 (T1:integer; MyAddress:Address_type);
 ip
 fc:Abstract-Frames-Channel; end;
```

The Acknowledgement timer T1 value and the local DXE address are specification parameters. The internal behavior of AP is described in the module ***body***. In the body declaration part are listed major state names under the ***state*** statement.

```
state s1, s2, . . . . s16;
```

This FSM is extended with the send and receive variables V(S) and V(R). V(S) contains the number of the next I frame to send. The number of the next expected I frame is kept in V(R). The variables *RemoteAddress* and ***Buffer*** are also defined.

```
var VS, VR:0..7;
 RemoteAddress:Address-type;
 buffer:frame_type;
```

Pascal procedures are defined for building up frames from their parameter values. For example, the procedure *Format_SABM* is defined as:

```
procedure Format-SABM( d,s:Address_type;
    var b:frame_type);
begin
with b do begin
  tag:=U;
  uf.Address.destination:=d; uf.Address.source:=s;
  uf.Control.MM:=SABM;
end; end;
```

The module has the following initialization part It starts in the major state *s1*.

```
initialize
  to s1
begin end;
```

The possible state changes of AP are defined by the transitions. The transition list is divided into five sections (phases): *disconnected*, *link-setup*, *disconnect-request*, *information-transfer* and *waiting-acknowledgement*. The disconnected and link-setup phases are discussed.

A **disconnected** DXE initiates a link setup by transmitting a SABM command to the remote DXE, starting timer T1 and going to the link-setup phase. The Estelle *delay-clause* is used to model the timer Tl. After the module has been in state s2 for T1 time units, the transition with the delay clause (bellow) is enabled. The implementer determines how *RemoteAddress* is obtained.

```
trans
from s1 to s2
begin
  RemoteAddress:=...;
  Format-SABM( RemoteAddress,MyAddress,b);
  output fc.out_frame(b);
end;
```

Upon receipt of a SABM command, a disconnected DXE returns a UA response. resets its send and receive variables VR and VS to zero and considers that the link is setup, i.e.. it enters *information-transfer* phase.

```
trans
from s1 to s5
when fc.in frame
provided frame.tag=U and
  frame.uf.Control.MM=SABM
begin
  RemoteAdrress:=frame.uf.Address.destination;
  Format_UA(RemoteAddress,MyAddress,b);
  output fc.out_frame(b);
  VR:=0; VS:=O;
end;
```

On receiving a DISC command in the disconnected state, the DXE sends a DM response and remains in the disconnected state.

```
trans
from s1 to s1
```

```
when fc.in_frame
provided frame.tag=U and
  frame.uf.Control.MM=DISC
begin
  RemoteAdrress:=frame.uf.Address.destination;
  Format-DM( RemoteAddress,MyAddress,b);
  output fc.out_frame( b);
end;
```

In the disconnected state, the DXE ignores and discards any frame from the remote DXE, except SABMs and DISCs.

```
trans
from s1 to s1
when fc.in_frame
provided not(frame.tag=U and
  (frame.uf.Control.MM=SABM or
  frame.uf.Control.MM=DISC))
begin end;
```

Upon reception of a UA response. in the **link-setup** phase, the local DXE will reset. its send and receive variables VS and VR to zero, stop timer T1 (implicitly realized by the transition) and consider that the link is setup.

```
trans
from s2 to s5
when fc.in_frame
provided frame.tag=U and
  frame.uf.Control.MM=UA
begin
  VR:=0; VS:=0;
end;
```

After the DXE has sent the SABM command, if a **UA** or a DM is not correctly received then timer T1 will run out. The DXE will then resend the SABM and will restart timer T1 (implicit).

```
trans
from s2 to s2
delay (T1)
begin
  Format_SABM( R.emoteAddress,MyAddress,b);
  output fc.out _frame( b);
end;
```

The DXE having sent the SABM command will ignore and discard any frame from the remote DXE, except SABMs, DISCs, or UAs.

```
trans
from s2 to s2
when fc.in_frame
provided not(frame.tag=U and
  (frame.uf.Control.MM=SABM or
  frame.uf.Control.MM=DISC or
  frame.uf.Control.MM=UA))
begin end;
```

In the link-setup phase, the receipt. of a SABM command from the remote DXE will result. in a collision. The local DXE sends a UA response and considers that the link is setup.

```
trans
from s2 to s5
when fc.in_frame
provided frame.tag=U and
  frame.uf.Control.MM=SABM
begin
  VR:=0; VS:=0;
end;
```

On the receipt of a DISC command, the DXE sends a DM response and enters disconnected phase.

```
trans
from s2 to s1
when fc.in_frame
provided frame.tag=U and
  frame.uf.Control.MM=DISC
begin
  Format-DM( RemoteAddress,MyAddress,b);
  output fc.out_frame( b);
end;
```

Would remain to be specified the disconnect-request, information-transfer and waiting- acknowledgement phases. Those phases are described easily in Estelle using the clauses that have been used up to now.

4. Conclusion

We presented the use of Estelle for describing AX.25 a link-level packet-radio protocol. The translation in Estelle was made easier because the informal description contains transition tables. The tables provided a skeleton for the formal specification. The

AP module gives a high level abstract view of AX.25 because many details not relevant to the understanding of the protocol behavior are delegated to the TR module.

Directions for further work would be: i) completion of the description in Estelle of AX.25 and ii) evaluation of other formal description techniques, such as Lotos[12] or SDL[13], for description of AX.25.

References

[1] ISO/TC 97/SC 21, *Estelle - A Formal Description Technique Based on an Extended State Transition Model*, Draft International Standard 9074, 1987.

[2] M. Barbeau, *Estelle: A Formal Description Technique for Communication Protocols*, Proceedings of the 6th ARRL Computer Networking Conference, Redondo Beach, California, August. 1987.

[3] S. Budkowski and P. Dembinski, *An Introduction to Estelle: A Specification Language for Distributed Systems*, Computer Net works and ISDN Systems. Vol. 14, No. I, 1987.

[4] T. L. Fox, *AX.25 Amateur Packet-Radio Link-Layer Protocol*, available from ARRL, Newington CT USA 06111, 1984.

[5] J. Harangozo. *An Approach to Describing a Link Level Protocol with a Formal Language*, Proceedings of the Fifth Data Communication Symposium, Snowbird, Utah. 1977.

[6] G. V. Bochmann and R. J. Chung, *A Formalized Specification of HDLC Classes of Procedures*, Proceedings of' the National Telecommunications Conference, Los Angeles, 1977.

[7] G. 'I-. Bochmann, *A General Transition Model for Protocols and Communication Services*, IEEE Transact ion on Communications, Vol. COM-28, No. 4, April 1980.

[8] G. V. Bochmann and T. Joachim, *Development and Structure of an X.25 Implementation*, IEEE Transaction on Software Engineering, Vol. SE-5, pp. 429-439, Spet. 1979.

[9] P. Taylor, *Design Abstraction for Protocol Software*, Proceedings of the 6th ARRL Computer Net working Conference, Redondo Beach, California, August) 1987.

[10] P. D. Amer, F. Ceceli and G. Juanolle, *Formal Specification of ISO Virtual Terminal in Estelle*, Proceedings of IEEE INFOCOM'88, New Orleans, Lousiana, March 1988.

[11] G. V. Bochmann a:nd C. Sunshine, *Formal Methods in Communication Protocol Design*, IEEE Transaction on Communications, Vol. COM-28, No. 4, April 1980.

[12] T. Bolognesi and E. Brinksma, *Introduction to the ISO Specification Language LOTOS*, Computer Networks and ISDN Systems. Vol. 14, No. 1, 1987.

[13] CCITT/SGXI, *Functional Specification and Description Language*, Recommendations 2.101 to 2.104, 1985.