PROTOCOL SPECIFICATION AND VERIFICATION USING THE SIGNIFICANT EVENT TEMPORAL LOGIC [1]

By

George K. Tsiknis and Son T. Vuong

Technical Report 88-3

January 1988

ABSTRACT

In this report we discuss the Significant Event Temporal Logic specification technique (SIGETL), a method for protocol specification and verification using a temporal logic axiomatic system. This technique is based on the idea that the state and the behaviour of a module can be completely described by the sequence of the significant events with which the module was involved in communicating with its environment till the present time. The behaviour of a module at any time is specified by simple temporal logic formulas, called transition axioms or properties of the module. Both, the safety and liveness properties of a module, as well as the global properties of a system, can be proven from its axioms using the axiomatic temporal logic system. As an example, we apply SIGETL to specify and verify a simple data transfer protocol. The general correspondence between SIGETL and ESTELLE FDT is also discussed.

---

[1] This paper is a revised version of the one accepted for presentation at the Fifth IFIP International Workshop on Protocol Specification, Testing and Verification in Moissac-Toulouse, France, in June 1985.

1. INTRODUCTION

Computer network protocols are often structured as a layered system in which each protocol layer provides specific services to the next higher layer using the services provided by the adjacent lower layer in the hierarchy. A service specification defines the services provided by the protocol, regardless of the way these services are realized by the entites; it merely describes the behaviour of the protocol layer as it is visible to the user at the next layer. A protocol specification on the other hand, describes how the services are supported and realized by each protocol entity at the layer being specified, by means of interactions with the services of the next lower layer. In this paper we deal mainly with protocol specification and its verification rather than the service specification per se. We take the view that service specification defines the properties the protocol specification must satisfy and verification is the process of proving that the protocol specification actually meets the given service specification. Therefore, the specification and verification technique should be powerful enough to allow any desirable property of a protocol to be expressed and verified.

Among the existing specification techniques which cater to verification, the temporal logic approaches predominate. An excellent overview of the most significant temporal logic methods is presented in [12]. Temporal logic techniques are subdivided into two (not necessarily disjoint) categories : the state-based methods and the event-based ones.

Representative state-based techniques are the unbounded state method pursued by Hailpern [2] and the bounded state one pursued by Schwartz and Melliar-Smith [11]. Hailpern includes in the state an unbounded auxiliary variable for each input and output of each module to record the sequence of messages exchanged between the module and its environment. Modules are specified in terms of properties the sequence variables must satisfy. The bounded state method includes some (bounded) internal state variables which record a finite history of the past for each module. Modules are specified in terms of temporal logic formulas which reflect the cause-effect relationship between the module interfaces.

Typical event-based techniques are the methods pursued by Vogt [14] and Schwartz and Melliar-Smith [12]. In these methods event sequences are (implicitly or explicitly) used to establish the context (state) in which a future event can occur. A module specification consists of temporal logic formulas which express constraints on allowable sequences of events. Any sequence that satisfies the constraints constitutes a valid behaviour. Vogt's method explicitly uses event sequence variables to establish the necessary context, while Schwartz and Mellier-Smith avoid it by using significantly more complex formulas for this purpose.

As it is indicated in [12] none of the previous methods is satisfactory because they inherit a great complexity and they are difficult to understand. On the other hand, state-transition specifications are straightforward but they provide very limited means for verification. In this paper we pursue a specification-verification technique which amalgamates the state-transition and event-based approaches. It uses event sequences to establish the sufficient context (state of the module) for a module specification, and temporal logic formulas to define the transition function of the state-transition machine which describes the module behaviour.

The SIGETL method we developed exhibits some similarities to Lamport's specification technique [18], in the sense that both are based on the state-transition paradigm and both use temporal logic as their specification language. Nevertheless, the two methods differ significantly in the following aspects. Whereas in Lamport's the state of a module is specified by a group of state variables, in SIGETL the state is represented by a sequence of events. In Lamport's events are mainly "procedure calls" whereas in SIGETL they are primitive objects. As a result, the specification does not rely on any kind of procedure call mechanism. Moreover, a specification in our method can enjoy any desired level of abstraction, which might not be the case in the other technique. For instance, in SIGETL the channels as well as other modules that are not of prime concern, can be described just by their high-level properties necessary for the verification of the system. An explicit specification of such modules is neither necessary nor desirable. Finally, our method was intentionally designed to be closely related to the Estelle FDT and to facilitate the development of a semi-automatic verification tool based on SIGETL.

In the remainter of the paper we assume that the reader has some knowledge of Estelle FDT [6] and temporal logic [1].

2. THE SIGNIFICANT EVENT MODEL (SEM)

The basic building unit in a system and in a protocol specification in particular is the module. A module is capable of performing certain functions and, in turn, may be composed of several submodules which interact with each other. The interactions between themselves and with their environment are realized by means of events. In a module we can distinguish two kinds of events : input events that is, interactions initiated by the environment and output events which are initiated by the module itself. Our model makes no distinction between modules and submodules; both are called modules.

All interactions (input and output events) between two modules are realized through a communication channel shared by the modules. The events in a channel shared by two cooperating modules are directly coupled in the sense that an output event at

one end of the channel is simultaneously viewed as input event at the other end. Moreover, only one event can occur in a channel at an instance of time. This is what we call Directly Coupled Events Rule (DCER).

The state of a module at any instance of time is described by the significant event sequence (SES) denoted by σ. This is the sequence of all the significant events that have occurred in any channel of the module from its initiation till the present time. By the term significant event we denote an event that, when it occurs, changes the state of the system. All the possible state changes for a module are denoted by the transitions or transition axioms of the module. Thus, an input event is a significant one provided at the time it occurs, a transition is enabled (it triggers a state transition), while any output event is a significant one.

The following rules hold for the events of any module:

a. Many input events can occur simultaneously (at different channels) but only one of them is realized by the module at a time. The rest remain outstanding for the next time. The choice is arbitrary. By the term "realized" we mean that an event has occured and the module has decided to act on it.

b. If an input event has been realized at a specific time then, if it is a significant one, it is concatenated to the event sequence in the next time interval (meaning it becomes past). Otherwise, it is simply ignored and the system remains at the same state.

c. When an input event occurs it is always realized at some future time.

d. If a significant input event (when realized) causes some output events to occur, then these output events are concatenated to σ at the same time with the input event (meaning that the transitions are atomic), and they consitute the current events of the module at that time.

The above rules are called Event Rules or ER .

Formally, a module is a 4-tuple (Ch, σ, T, S) where: Ch is the set of channels, defining all the types of the module events, σ is the past-present significant event sequence of the module at the present time, T is the set of the transition axioms (or transitions), and S is the future state sequence (possibly infinite). The state sequence has the form $s=s_0, s_1, \ldots$, where s_0 is the current state described by the sequence $\sigma 0$ and s_i is the ith future state of the module.

The method resembles the Event-Based method developed by Vogt [14], in the sense that it also uses event sequences to encode information about a module. It differs from the

Event-Based technique in many respects. In particular, in SIGETL each module has its own sequence which includes only the significant events occurred in the module, and it describes the state of the module at any time. SIGETL specifies the module behaviour by a set of state transitons which define the next possible state of the module. Cosequently invariant (safety) proofs are feasible in SIGETL. Finaly, the axiomatic system used in SIGETL is quite different from the one used by Vogt.

3. SIGETL AS A SPECIFICATION LANGUAGE

A protocol, according to the ISO model, is viewed as a set of interacting modules . Consequently, a protocol specification consists of the specification of the protocol modules and the channels shared among them. In SIGETL, the channels are specified in exactly the same way as in ISO Estelle formal description technique(FDT) [6],[6'].

A module specification consists of:

1. The module header which indicates the name of the module and the channels used by it (this is again identical to Estelle module header);
2. The event sequence σ for the module; and
3. The transitions.

The language used to specify the transitions is the temporal logic augmented by some functions on σ, and some special predicates.

For a given state sequence s, we use the notation

$$s+n \equiv s_n , s_{n+1} , \ldots$$

and Æ is the existential quantifier. The interpretation of a temporal formula A on s, denoted as A_s is defined as :

1. If A is an atomic formula then $A_s = s_0(A)$; that is, A is evaluated at the current state.
2. $(A \; L \; B)_s \equiv A_s \; L \; B_s$ where L can be $\wedge$, V, $\supset$ and $(\sim A)_s \equiv \sim A_s$
3. $(\Box A)_s \equiv \forall n \geq 0 \; (A_{s+n})$
4. $(\Diamond A)_s \equiv Æ n \geq 0 \; (A_{s+n})$
5. $(OA)_s \equiv A_{s+1}$
6. $(A \text{ Until } B)_s \equiv \forall n \geq 0 \; [\forall 0 \leq i \leq n \; (\sim B_{s+i})] \supset A_{s+n}$

7. $(A \text{ Until-After } B)_s \equiv (A \text{ Until } (A \wedge B))_s$

It is clear that Until is the only operator that we need since $\Box A \equiv A$ Until false and $\Diamond A \equiv \sim\Box\sim A$

For reasoning about the event sequences, we use the following notation :

σ : $[A_1, \ldots, A_n]$ denotes that σ is a sequence which may contain events of kind $A_1, \ldots, A_n$ only.

$\sigma = \langle e_i \rangle_{i=0}^{n}$ denotes that σ contains the events (must be of the same kind) $e_0, \ldots, e_n$ in this order. In addition e^* denotes 0 or more consecutive instances of e, while e^+ denotes 1 or more.

The following functions on the σ sequence are used :
$|\sigma|$ returns the number of events in σ. Note that:

$$\text{if } \sigma = \langle e_i \rangle_{i=0}^{n} \text{ then } |\sigma| = n+1.$$

$P(\sigma, Y)$ is the projection function on σ. It returns a sequence which contains only the events of kind Y which are in σ and in the same order they occur.
Finally, $suff(\sigma, Y)$ is the suffix function (as it is defined by F. Vogt in his event model). It is defined as:

$$\sigma = \langle \sigma 1 \rangle \langle y \rangle \langle suff(\sigma, Y) \rangle$$
$$\text{and, } P(suff(\sigma, Y), Y) = \lambda$$

where λ is the empty sequence.

The only operation allowed to be done on σ by the transitions is the addition of new events. We use the notation

$$\sigma \langle e_1, \ldots, e_n \rangle$$

to define the sequence resulting from σ by concatenating to it the events $e_1, \ldots, e_n$. The order of the concatinated events is irrelevant, but an expression of the form

$$\sigma \langle e_1, \ldots, e_n \rangle \langle m_1, \ldots, m_k \rangle$$

denotes that events e_i occur before events m_j. Consequently, events e are past events, while, events m are the current ones.

Transitions, in general, express the fact that if at a certain instance of time a valid input event (not violating the

history σ) is realized, then the next time this becomes "past", and the output events triggered by this, become the last "current" events of σ. The last events are considered current and past at the same time. (This is similar to the notion of present-past events defined by F. Vogt in his event model).

Consequently, each transition has the form

$$C \wedge [\sigma=\sigma 0] \wedge [at(e)] \supset O([\sigma=\sigma 0<e><o_1,\ldots,o_n>] \wedge [o_1] \ldots \wedge [o_n])$$

where [at(e)] means that input event e has occurred and realized by the module; $[\sigma=\sigma 0]$ means that the sequence of events σ is $\sigma 0$, while $[\sigma=\sigma<e><o_1,\ldots,o_n>]$ means that the sequence σ contains all the events in $\sigma 0$ and the events $e, o_1,\ldots,o_n$; C is the enabling condition of the transition; $o_1,\ldots,o_n$ are the output events triggered by the transition; and $[o_i]$ means that the event o_i occurs.

Note that for an input event e, [e] means that the event e has occurred but not yet realized, while [at(e)] means that e has been realized by the module. Consequently, the following rule holds

- $[e] \supset \Diamond[at(e)]$ (E-Rule)

Also note the necessity of using "[ ]" to denote that every term is a proposition (either true or false at any state of the system).

SIGETL offers mainly two levels of abstraction. At the lower level a module is specified by its SES and the transition axioms as described earlier. If a higher level of abstraction is desired, the module can be specified as a black box; that is, its behaviour is described by its safety and liveness properties. In this description the module's SES does not need to be specified; instead its properties can be specified by using the event sequences of the modules with which this module interacts. Examples of black box specifications are the "additional modules" of a simple data transfer protocol in APPENDIX 3.

4. SIGETL SYSTEM FOR PROTOCOL VERIFICATION

The axiomatic temporal logic system used in SIGETL is shown in APPENDIX 2. This system is different from the DUX system presented in [1] mainly because it uses the Until-After modality and it has a different induction axiom (A6). Our induction axiom postulates that if it is always the case that when p is true and q is false then p is true at the next state, then if p is true at the current state it will remain true up to the first state at which q becomes true. This axiom is more suitable for the verification of the protocols than the old induction axiom which

is a theorem in our system (Th1) as an immediate consequence of A6 with q=false. Also, axiom A7, which defines the Until-After modality, does not exist in DUX.

Among the derived rules the first three are typical rules in any S4 system while the rest are special SIGETL rules very frequently used in the proofs of the liveness properties of the protocols. The latter rules can be derived from the axioms by the inference rules and propositional logic (PL) as it is illustrated by the following derivation of the rule D4:

1. p ⊃ p Until-After q	Hyp
2. <>q	Hyp
3. p ⊃ □(p ∧ ~q) ∨ <>(p ∧ q)	1,A9,A8,PL
4. <>q ⊃ ~□(p ∧ ~q)	PL,A4
5. p ⊃ <>(p ∧ q)	2,3,4,PL

Similar derivations can be given for the other derived rules.

We distinguish two kinds of properties of a module to be verified : safety properties and liveness properties. Safety properties (or invariants, analogous to partial correctness of programs) have the form □I(σ) where I is a modality-free logic formula. To prove □I(σ) it is sufficient to prove that I holds initially (when $\sigma=\lambda$) , and that it is preserved by each transition; that is, if I(σbefore) ⊃ I(σafter) where σbefore and σafter are the sequences before and after the "execution" of a transition. (I-Rule). Liveness properties (or commitments) have the form A ⊃ B where A and B are temporal logic formulas. These properties are proven from the transitions using the axiomatic system. Finally, the global properties of a system are proved from the properties of its components using the axiomatic system.

5. TRANSLATION FROM ESTELLE TO SIGETL

As mentioned previously, SIGETL is very similar to Estelle [6] in some respects but, while Estelle is implementation oriented , SIGETL is a verification oriented formalism. We now give some quidelines for the translation between the two techniques. The reader will notice the intentional similarities between them.

In Estelle the state of a module is defined by the values of the "major state" variable and some other variables called state components (e.g. sequence number, etc.). These state components need to be translated in SIGETL using the event sequence σ and the defined functions. In some cases special functions on σ must be defined in order to express some variables (such as credits available or finite sequence number) as functions of σ.

The Estelle-to-SIGETL translation of a specification is relatively straitforward. The type and channel definitions of both formalisms are identical. Furthermore, an Estelle transition of the form :

```
from <fromstate>
to <tostate>
   when <event>
      provided <enabling condition>
         begin
         <transition body>
         end
```

is translated to

translate(fromstate, enabling condition) $\Lambda[\sigma=\sigma 0]$ Λ [at(event)] $\supset$ $\bigcirc$([$\sigma=\sigma 0$<event><out events>] Λ [out events])

where translate(fromstate,enabling condition) is the condition that results from translating fromstate and enabling condition in SIGETL terms; and out events are the events initiated in the transition body. Similar rules can be deduced for the other transition types.

As an example, the SIGETL specification of a simple data transfer protocol is provided in Appendix 3, which was derived from the Estelle specification version in Appendix 4. The type and channel definitions in both formalisms are identical and there is one transition axiom in the SIGETL specification for a module for each Estelle transition of the same module. In addition there are some additional transition axioms which describe the behaviour of the module when an event, expected at a "certain state", does not occur (e.g. axioms t4, t5 and r3 in APPENDIX 3). Moreover in SIGETL additional axioms are needed to insure that only one input event can be realized at any time by the module (axiom t6 in Transmitter).

For verification purposes, in SIGETL we need in addition to the entity specifications, the specifications of all the other modules involved in the system (i.e. network module, user modules, op. system module, etc.). These specifications are called "Additional Modules". For each one of these modules only their properties (safety and liveness) need to be specified and they are given in the section called "Additional Modules".

In the rest of the paper, we give an example using a simple data transfer protocol.

6. SIGETL SPECIFICATION OF A SIMPLE DATA TRANSFER PROTOCOL

This protocol is similar to the one presented by Hailpern in [4]. It is essentially an "alternating bit" protocol with an unbounded sequence number. In this protocol the sender gets the next message to be sent from user1, appends to it the sequence number and sends it to the receiver via the network(medium). The sender retransmits the same message repeatedly until it receives an acknowledgment for that message. Whenever the receiver receives the next expected message, it delivers the message to user2 and sends an acknowledgment for the message to the sender.

The SIGETL specification of the protocol is given in APPENDIX 3 while APPENDIX 4 contains the Estelle specification of the same protocol.

The abbreviations (state1),(state2) in the SIGETL specification give a hint for the translation between the two specifications. In addition, we need to express the other state components namely send-seq and recv-seq with SIGETL formulas. This is achieved by :

in ESTAB ⊃ send-seq = $|us(\sigma)|$
in ACK-WAIT ⊃ send-seq = $|us(\sigma)|-1$
and recv-seq = $|ur(\sigma)|$

Having stated that, the translation should be relatively straitforward. However, some explanations for the properties of the additional modules are in order.

In the SIGETL specifications :
(U1), states that the user1 if it sends a message, it sends it after it received an indication that the previous one was indeed delivered to the other user.
(U2) states the willingness of user1 to send a message after the last acknowledgment.
(Timer) gives the liveness property of the timer. That is, if the timer is set after the last TIMEOUT or STOPTIMER then a TIMEOUT is expected.

The Network module describes the properties of a medium with minimum requirements. This medium may lose or destroy or corrupt a finite number of messages (N3, N4, N5, N6), but can not generate messages by itself (N1,N3). Moreover, this medium is not necessarily a FIFO one. It is assumed that message corruption is detected by a lower-level mechanism and corrupted messages are discarded.

In this specification we use the pseudo-functions data, id, seq, which if applied to any event will return the value of the coresponding field of the message of the event.

We are now ready to discuss the verification of this example to illustrate how the SIGETL system can be used for protocol verification in general.

7. VERIFICATION OF THE DATA TRANSFER PROTOCOL

For this protocol we want to prove:
First, that if any messages are delivered to User2, these are the messages sent by User1 and they are delivered in the order they were sent (safety).
And second, that if User1 keeps sending messages, at some future time messages will indeed be delivered to User2 (liveness).

In order to prove the above properties we need first to state and prove the invariants and commitments of the sender (consisting of the transmitter and the system module) and the receiver.

For simplicity, we shall omit enclosing every term in the properties as well as in the proofs given in this section in '[ ]'. Furthermore, without any confusion , we simply use σ for both Sender and Receiver.

a. Sender

The safety specifications of the sender are given by the invariants:

- $\square(us(\sigma)=<us_i>_{i=0}^{|us(\sigma)|-1} \wedge sd(\sigma)=<sd_i^+>_{i=0}^{|us(\sigma)|-1})$ (T1)
- $\square(ra(\sigma)=<ra_i>_{i=0}^{|us(\sigma)|-2})$ (T2)

T1 states that at any time $|us(\sigma)|$ messages have been received from the user and all these messages have been sent to the network (each one many times) in the order they have been received. T2 in conjunction with T1 denotes that if n messages have been sent, then the first n-1 must have been acknowledged.

Since the proofs of T1 and T2 are quite similar we only give the first one.

Proof of T1

In this proof we use σ to denote the sequence before the execution of a transition and σnext to denote the sequence at the next state (that is after executing a transition), and T1 refers to the formula T1 without $\square$.

1. in ESTAB $\wedge$ at(us) $\wedge$ T1(σ) $\supset$ T1(σnext) — t1

   since $us(\sigma next)=us(\sigma)<us_{|us(\sigma)|}>$

   and $|us(\sigma)| = |us(\sigma next)| - 1$,

   and $sd_{|us(\sigma next)|-1} \in \sigma next$

2. in ESTAB $\wedge$ at(us) $\wedge$ T1(σ) $\supset$ O(T1(σ)) — 1, O-interp.

3. in ACK-WAIT $\wedge$ at($ra_{|us(\sigma)|-1}$ $\wedge$ T1(σ) $\supset$ T1(σnext) $\supset$ O(T1(σ)) — t2, O-int.

4. in ACK-WAIT $\wedge$ at(TIMEOUT) $\wedge$ T1(σ) $\supset$ T1(σnext) $\supset$ O(T1(σ)) — t3, O-int.

5. $T1(\sigma) \supset OT1(\sigma)$ — 1,...,4, t4, t5, t6

6. $\Box(T1(\sigma) \supset OT1(\sigma))$ — 5, gen.

7. $T1(\sigma) \supset \Box T1(\sigma)$ — Th1.

8. $T1(\sigma)$ since $T1(\lambda)$ is true

9. $\Box T1(\sigma)$ — 7,8,mp.

The liveness properties of the transmitter are expressed by the commitments:

- We define: $UC \equiv \Box((\sigma=\lambda) \vee (ua \in suff(\sigma,us)) \supset <>us)$
- $UC \supset \Box<>sd$ (T3)
- $\forall k(ra_k \supset |us(\sigma)| \geq k+1) \supset \forall k(ra_k \supset <>(ra_k \in \sigma))$ (T3')
- $\forall k(ra_k \in \sigma \supset |us(\sigma)| \geq k+1)) \wedge UC \supset$ (T4)

  $\forall j(ra_j \in \sigma \supset \Box<>sd_{j+1} \vee <>(ra_{j+1} \in \sigma))$

T3: states that if the user keeps sending messages to the transmitter then, the transmitter keeps sending to the network. T3': if the acknowledgment of any message is received after the message has been sent, then any acknowledgement received is added to the sequence (is a significant one). T4: if the acknowledgment of any message is received after that message has been sent then, if the acknowledgment for the message j is received, the transmitter keeps sending the next message until its acknowledgment is received.

To prove these commitments, we need some properties for the sender deduced from the transition axioms and (Timer). These are:

- in ESTAB $\supset$ in ESTAB Until-After at(us) (tp1)
- in ACK-WAIT $\supset$ (tp2)

  in ACK-WAIT Until-After $at(ra_{|us(\sigma)|-1})$
- in ACK-WAIT $\wedge \sim at(ra_{|us(\sigma)|-1}) \supset <>TIMEOUT$ (tp3)
- $ua \in suff(\sigma,us) \equiv ar_{|us(\sigma)|} \in \sigma$ (tp4)

• □(in ESTAB V in ACK-WAIT) (tp5)

tp1, tp2 are deduced from the axioms t3, t4, t5, t6 and A6.

tp3 can be deduced from the fact that

in ACK-WAIT $\wedge$ $\tilde{}$at($ra_{|us(\sigma)|-1}$) $\supset$ O(SETTIMER $\in$ suff(σ,[STOPTIMER,TIMEOUT])).

tp4 comes from the fact that ua and $ra_{|us(\sigma)|-1}$ are always added to σ at the same state (t2).

We now give the proof of the commitment T3. The other proofs are similar to this.

Proof of T3

1. UC	Hyp
2. in ESTAB $\supset$ <>us	1,tp4, state1
3. in ESTAB $\supset$ <>(in ESTAB $\wedge$ at(us))	2,tp1, E-Rule,D4
4. in ESTAB $\supset$ <>sd	3,t1,Th1, $\supset$-Trans
5. in ACK-WAIT $\supset$ □(in ACK-WAIT $\wedge$ $\tilde{}$at($ra_{\|us(\sigma)\|-1}$)) V <>(in ACK-WAIT $\wedge$ at($ra_{\|us(\sigma)\|-1}$))	tp2, A8, $\supset$-Trans
6. in ACK-WAIT $\supset$ □<>(in ACK-WAIT $\wedge$ at(TIMEOUT)) V <>(in ACK-WAIT $\wedge$ $ra_{\|us(\sigma)\|-1}$)	5, tp3, E-Rule,D4
7. in ACK-WAIT $\supset$ □<>sd V <>(in ESTAB)	6,t2,t4
8.in ACK-WAIT $\supset$ <>sd	7,4
9. <>sd	4,8, Proof by Cases, tp5

10. $\Box\Diamond$sd — 9, gen

11. UC $\supset \Box\Diamond$sd — 1,10,D3.

b. Receiver

The safety properties of the receiver are :

- $\Box(ur(\sigma)=<ur_i>_{i=0}^{|ur(\sigma)|-1} \wedge \forall i \leq |ur(\sigma)|-1(rd_i \in \sigma))$ (R1)
- $\Box(sa(\sigma)= < sa_i^+ >_{i=0}^{|ur(\sigma)|-1})$ (R2)

and its liveness commitments are:

- $\Box\Diamond rd \supset \Box\Diamond sa$ (R3)
- $\forall k(rd_k \supset \Diamond(rd_k \in \sigma))$ (R3')
- $\Box(rd_k \supset |ur(\sigma)| \geq k) \wedge \Box\Diamond rd \supset$ (R4)

  $\forall j(rd_j \in \sigma \supset (\Diamond(|ur(\sigma)| \geq j+1) \wedge$

  $(\Box\Diamond sa_j \vee \Diamond(rd_{j+1} \in \sigma))))$

Informally :
R1: states that the data delivered to the user are the data of the in-sequence messages received by the receiver.
R2: insures that the receiver sends an acknowledgement (possibly many times) for every in-sequence message it receives.
R3: if an unbounded number of messages reach the receiver, then an unbounded number of acknowledgments have been sent.
R4: Assuming that message k does not arrive until the receiver has processed message k-1, and that messages do not stop coming to the receiver, the receiver will keep sending an acknowledgment for the last message, until it receives the next one.

The proofs of these formulas are similar to the proofs for the sender properties and they are omitted.

C. Safety of the System

The safety of the system is expressed by :

• $\Box(ur(\sigma)=<ur_i>_{i=0}^{n} \supset$ (S1)

$\forall i \leq n((us_i \in \sigma) \wedge data(us_i)=data(ur_i)))$

which states that if n+1 messages have been received by the user2 these are the first n+1 messages sent by user1 and the order is preserved.

Proof of S1

1.UR $\equiv ur(\sigma)=<ur_i>_{i=0}^{n}$

2. UR $\supset \forall \leq n((rd_i \in \sigma) \wedge data(ur_i)=data(rd_i))$ R1 def ur, dr

3. UR $\supset \forall i \leq n((sd_i \in \sigma) \wedge data(sd_i)=data(ur_i))$ 2, N1

4. UR $\supset \forall i \leq n((us_i \in \sigma) \wedge data(ur_i)=data(us_i))$ T1, def sd, us

Since U1 holds implies that the us_i's in σ are the only messages the user1 has sent.

d. Liveness of the System

The liveness of the system is given by:

• $\Box<>sd \wedge \Box<>rd \wedge \Box<>sa \wedge \Box<>ra$ (L1)

• $\Box(|ur(\sigma)|=n \supset <>(|ur(\sigma)|>n))$ (L2)

The first formula denotes that the system is starvation free; that is, infinitely many messages are transfered through each one of the four system channels.The second formula expresses the system liveness; that is, at any time if user2 has received n messages then he will definately receive the next message at some future point.

We now give the proofs of L1 and L2.

Proof of L1

1. UC	U1, tp4
2. $\Box\langle\rangle sd$	1, T3, mp
3. $\langle\rangle rd$	2, N5, mp
4. $\Box\langle\rangle rd$	3, gen
5. $\Box\langle\rangle sa$	4, R3, mp
6. $\langle\rangle ra$	5, N6, mp
7. $\Box\langle\rangle ra$	6, gen
8. L1	2, 4, 5, 7 $\wedge$-Intr

Proof of L2

First we prove the hypotheses of T4, T3' and R4 :

1. $ra_k \vee ra_k \in\sigma \supset sa_k \in\sigma$	N2
2. $sa_k \in\sigma \supset \lvert ur(\sigma)\rvert \geq k+1$	R2
3. $\lvert ur(\sigma)\rvert \geq k+1 \supset rd_k \in\sigma$	R1
4. $rd_k \in\sigma \supset sd_k \in\sigma$	N1
5. $sd_k \in\sigma \supset \lvert us(\sigma)\rvert \geq k+1$	T1
6. $\forall k(ra_k \vee ra_k \in\sigma \supset \lvert us(\sigma)\rvert \geq k+1)$	1,...,5, $\supset$-Trans, gen
7. $\lvert us(\sigma)\rvert \geq k+1 \supset ra_{k-1} \in\sigma$	T2
8. $ra_{k-1} \in\sigma \supset sa_{k-1} \in\sigma$	N2
9. $sa_{k-1} \in\sigma \supset \lvert ur(\sigma)\rvert \geq k$	R2
10. $\forall k(rd_k \vee rd_k \in\sigma \supset \lvert ur(\sigma)\rvert \geq k)$	4,5,7,8,9,$\supset$-Trans, gen

Now we prove L2:

11. $\forall j(ra_j \in \sigma \supset \Box\Diamond(sd_{j+1}) \vee \Diamond(ra_{j+1} \in \sigma))$	6, U1, T4, mp
12. $\forall j(rd_j \in \sigma \supset \Diamond(\lvert ur(\sigma)\rvert \geq j+1) \wedge (\Box\Diamond sa_j \vee \Diamond(rd_{j+1} \in \sigma)))$	L1, 10, R4, mp
13. $\lvert ur(\sigma)\rvert = n \supset rd_{n-1} \in \sigma$	R1
14. $\lvert ur(\sigma)\rvert = n \supset \Box\Diamond sa_{n-1} \vee \Diamond(rd_n \in \sigma)$	13, 12, sub, $\supset$-Trans
15. $\Box\Diamond sa_{n-1} \supset \Diamond(ra_{n-1} \in \sigma)$	N6, U1, T3', $\supset$-Trans
16. $\Diamond(ra_{n-1} \in \sigma) \supset \Box\Diamond sd_n \vee \Diamond(ra_n \in \sigma)$	11,sub,D1,Th6,Th15
17. $\Box\Diamond sd_n \supset \Diamond(rd_n \in \sigma)$	N3, R3'
18. $\Diamond(rd_n \in \sigma) \supset \Diamond(\lvert ur(\sigma)\rvert \geq n+1)$	12, sub
19. $\Diamond(ra_n \in \sigma) \supset \Diamond(sa_n \in \sigma)$	N2, D1
20. $\Diamond(sa_n \in \sigma) \supset \Diamond(\lvert ur(\sigma)\rvert \geq n+1)$	R2, D1
21. L2	14, ..., 20, Proof by Cases, gen.

8. CONCLUSIONS

It is a general belief that state transition oriented specifications are easier to understand since the behaviour of a system is described by an abstract program. Such specifications, although they are closer to an implementation of the system, provide little or no means for expressing and reasoning about the properties (correctness) of the system. On the other hand, temporal logic seems to be a suitable tool for this purpose. However, specification methods using temporal logic tend to produce complicated expressions which are not only hard to understand but also difficult to reason with.

The Significant Event Temporal Logic(SIGETL) method we have developed can be viewed as a generalized transition oriented specification technique as well as a sound tool for verifying protocols specified in any transition oriented method. We believe that SIGETL bridges the gap between the two general categories mentioned earlier, in a very natural way. The only disadvantage of SIGETL is the inherent undesidability of first order temporal logic. However, since theorem proving techniques are quite advanced nowadays, we believe that implementation of a SIGETL semi-automatic verification system is feasible and worthwhile. We have also applied the SIGETL technique on other protocols including a data transfer protocol with finite sequence number, conditional events, buffers and FIFO medium (a version of the alternating bit protocol) [13].

Our research effort presented in this paper aims at adding a verification capability to an integrated set of tools under development at the University of British Columbia, which currently provides validation and synthesis facilities via tools called VALIRA and VALISYN [15] and an automatic implementation capability for protocols specified in Estelle via an Estelle-C compiler [16,17].

REFERENCES

[1] M. Ben, "Temporal Logic Proofs of Concurrent Programs", ACM TOPLAS, 1980.

[2] B. Hailpern, "Verifying Concurrent Processes Using Temporal Logic", Technical Report 195, Computer Systems Lab., Stanford Univ., Aug. 1980, also in LNCS 129, Springer-Verlag.

[3] B. Hailpern, S. Owicki, "Verifying Network Protocols Using Temporal Logic", NBS Trends and Appl. Symp., May 1980.

[4] B. Hailpern, S. Owicki, "Modular Verification of Computer Protocols", IBM Research Report RC 8726, 1981.

[5] C. Hoare, "Communicating Sequential Processes",CACM, Aug 78, V21, N8.

[6] ISO/TC97/SC16/WG1 Subgroup B, "A FDT Based on an Extended State Transition Model", March 1984.

ISO/TC97/SC21/WG1/DIS9074, "Estelle - A Formal Description Technique Based on an Extended State Transition Model, 1987.

[7] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specificatons", Tech. Report CS-81-872, Comp. Science, Stanford Univ., 1981.

[8] Z. Manna, A. Pnueli, "Verification of Concurrent Programs, Part II: Temporal Proof Principles", Tech. Report CS-81-843, Comp. Science, Stanford Univ.,1981.

[9] S. Owicki, L. Lamport, "Proving Liveness Properties of Concurrent Programs", ACM Trans. on Prog. Lang. 4, 1982.

[10] A. Pnueli, "The Temporal Semantics of Concurrent Programs", Intern. Symp. on the Semantics of Concurrent Computation, Evian, Springer-Verlag, July 1979.

[11] R. Schwartz, P. Melliar-Smith, "Temporal Logic Specification of Distributed Systems", Proceedings of the IEEE Conference on Distributed Systems, April 1981.

[12] R. Schwartz, P. Melliar-Smith, "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", Protocol Specification Testing and Verification, C. Sunshine (ed.), North-Holland, 1982

[13] G. Tsiknis, "Specification-Verification of Protocols - The Significant Event Temporal Logic Technique", M. Sc. Thesis, University of British Columbia, April 1985.

[14] F. Vogt, "Event-Based Temporal Logic Specifications of Services and Protocols", Protocol Specification Testing and Verification, C. Sunshine (ed.) North-Holland, 1982.

[15] S. Vuong, et al., "VALIRA - A Tool for Protocol Validation via Reachability Analysis", 6th IFIP Workshop on Protocol Specification Testing and Verification, Sarikaya and Bochmann (Eds.), June 1986.

[16] S. Vuong, A. Lau, "A Semi-Automatic Aproach to Protocol Implementation - The ISO Class 2 Transport Protocol as an Example", IEEE INFOCOM'87, SanFrancisco, April 1987.

[17] S. Vuong, A. Lau, I. Chan, "Semiautomatic Implementation of Protocols Using an Estelle-C Compiler", IEEE Trans. on Software Engineering, March 1988.

[18] L. Lamport, "Specifying Concurrent Program Modules", TOPLAS 5 2, April 1983.

APPENDIX 1
Axioms and Derived Rules of Standard Logic

$A \vdash A$	Assump Axiom
If $\vdash B$ then $A \vdash B$	Assump Intro
If $A \vdash B$ and $\sim A \vdash B$ then $\vdash B$	Assump Elim
$\vdash T$	T Axiom
$\sim\vdash F$	F Axiom
If $\vdash A$ then $\vdash A \vee B$	V Intr
If $A \vdash C$ and $B \vdash C$ and $\vdash A \vee B$ then $\vdash C$	V Elim
If $\vdash A$ and $\vdash B$ then $\vdash A \wedge B$	$\wedge$ Intr
If $\vdash A \wedge B$ then $\vdash A$	$\wedge$ Elim
If $A \vdash B$ then $\vdash A \supset B$	$\supset$ Intr
If $\vdash A$ and $\vdash A \supset B$ then $\vdash B$	$\supset$ Elim
If $A \vdash B$ and $A \vdash \sim B$ then $\vdash \sim A$	$\sim$ Intr
If $\vdash A$ then $\vdash \sim\sim A$	$\sim\sim$ Intr
If $\vdash \sim\sim A$ then $\vdash A$	$\sim\sim$ Elim
If $\vdash A \supset B$ and $\vdash B \supset A$ then $\vdash A \equiv B$	$\equiv$ Intr
If $\vdash A \equiv B$ then $\vdash A \supset B$	$\equiv$ Elim
$(A \supset (B \supset C)) \equiv ((A \wedge B) \supset C)$	$\supset$ to $\wedge$ Trans
$((A \vee B) \wedge (\sim A)) \supset B$	V Elim
$((A \supset B) \wedge (\sim B)) \supset (\sim A)$	$\supset$ Elim
$((A \supset C) \wedge (B \supset C)) \equiv ((A \vee B) \supset C)$	Proof by Cases
$((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$	$\supset$ Trans
$((A \equiv B) \wedge (B \equiv C)) \supset (A \equiv C)$	$\equiv$ Trans
$(A \supset B) \equiv (\sim A \vee B)$	$\supset$ to V Trans
$\sim(A \vee B) \equiv (\sim A \wedge \sim B)$	DM
$\sim(A \wedge B) \equiv (\sim A \vee \sim B)$	DM'
$(A \supset B) \equiv (\sim B \supset \sim A)$	CP
$(A \wedge B) \equiv (B \wedge A)$	$\wedge$ Commut
$(A \vee B) \equiv (B \vee A)$	V Commut
$(A \equiv B) \equiv (B \equiv A)$	$\equiv$ Commut

APPENDIX 2
The Temporal Logic System

AXIOMS

A1. $\Box(p \supset q) \supset (\Box p \supset \Box q)$
A2. $O(p \supset q) \supset (Op \supset Oq)$
A3. $\Box p \supset p \wedge Op \wedge O\Box p$
A4. $\Box p = \sim \Diamond \sim p$
A5. $\sim Op = O \sim p$
A6. $\Box(p \wedge \sim q \supset Op) \supset (p \supset p \text{ Until-After } q)$
A7. $p \text{ Until } q \equiv q \vee (p \wedge O(p \text{ Until } q))$
A8. $p \text{ Until } q \supset \Box(p \wedge \sim q) \vee \Diamond q$
A9. $p \text{ Until-After } q \equiv p \text{ Until } (p \wedge q)$

INFERENCE RULES

R1, (taut). If p is a (substitution of a) tautology then $\vdash p$.
R2, (mp). If $\vdash p$ and $\vdash p \supset q$ then $\vdash q$.
R3, (gen). If $\vdash p$ then $\vdash \Box p$.

DERIVED RULES

D1. If $\vdash p$ then $\vdash Mp$ where M is $\Box$ or $\Diamond$ or O
D2. If $\vdash p \supset q$ then $\vdash Mp \supset Mq$
D3. If $p \vdash q$ then $\vdash \Box p \supset q$. Deduction theorem (ded)
D4. If $\vdash p \supset (p \text{ Until-After } q)$ and $\vdash \Diamond q$
then $\vdash p \supset \Diamond(p \wedge q)$ (U-$\Diamond$ Rule)
D5. If $\vdash p \supset (p \text{ Until-After } q)$ and $\vdash r \wedge \sim q \supset \Diamond r$
then $\vdash p \supset \Box\Diamond(p \wedge r) \vee \Diamond(p \wedge q)$ (U-$\Box\Diamond$ Rule)
D6. If $\vdash p \supset (p \text{ Until-After } q)$ and $\vdash \Box \sim q$
then $\vdash p \supset \Box(p \wedge \sim q)$ (U-$\Box$ Rule)

THEOREMS

Th1. $\Box(p \supset Op) \supset (p \supset \Box p)$
Th2. $Op \supset \Diamond p$
Th3. $\Box(p \wedge q) \equiv \Box p \wedge \Box q$
Th4. $\Diamond(p \vee q) \equiv \Diamond p \vee \Diamond q$
Th5. $\Box\Box p \equiv \Box p$
Th6. $\Diamond\Diamond p \equiv \Diamond p$
Th7. $O(p \wedge q) \equiv Op \wedge Oq$
Th8. $Op \vee Oq \equiv O(p \vee q)$
Th9. $\Diamond(p \wedge q) \supset \Diamond p \wedge \Diamond q$
Th10. $\Box p \vee \Box q \supset \Box(p \vee q)$
Th11. $p \wedge O\Box p \equiv \Box p$
Th12. $\Diamond p \equiv p \vee O\Diamond p$
Th13. $\Box((p \vee \Box q) \wedge (q \vee \Box p)) \equiv \Box p \vee \Box q$
Th14. $\Diamond\Box p \equiv \Box\Diamond\Box p$
Th15. $\Box\Diamond p \equiv \Diamond\Box\Diamond p$
Th16. $\Diamond p \wedge \Box q \supset \Diamond(p \wedge \Box q)$
Th17. $\Diamond\Box p \wedge \Box q \supset \Diamond\Box(\Box p \wedge \Box q)$
Th18. $\Diamond\Box p \wedge \Diamond\Box q \supset \Diamond\Box(\Box p \wedge \Box q)$
Th19. $\Box\Diamond(p \vee q) \supset \Box\Diamond p \vee \Box\Diamond q$
Th20. $\Box\Diamond p \wedge \Box q \supset \Box\Diamond(p \wedge \Box q)$
Th21. $\Box\Diamond p \wedge \Diamond\Box q \supset \Box\Diamond(p \wedge \Box q)$
Th22. $(p \supset q) \text{ Until } r \supset (p \text{ Until } r \supset q \text{ Until } r)$
Th23. $p \text{ Until } q \supset (\sim q \text{ Until } r \supset p \text{ Until } r)$
Th24. $p \text{ Until } (q \wedge r) \supset (p \text{ Until } q) \text{ Until } r$
Th25. $O(p \text{ Until } q) \supset (Op) \text{ Until } (Oq)$

APPENDIX 3
SIGETL Specification of the Data Transfer Protocol

```
type

   data-type=...;
   seq-type=0...;
   id-type=(DATA,ACK);


(* Channel definitions *)

Channel UserTransmitter(User, Provider);
   By User:
      SENDreq(d:data-type);
   By Provider:
      SENDack;

Channel UserReceiver(User, Provider);
   By Provider:
      RECEIVEindic(d:data-type);

Channel EntityNetwork(User, Provider);
   By User:
      SEND(id:id-type, d:data-type, seq:seq-type);
   By Provider:
      RECEIVE(id:id-type,d:data-type, seq:seq-type);

Channel System Transmitter (User, Provider);
   By User:
      STARTTIMER;
      STOPTIMER;
   By Provider:
      TIMEOUT;


(* Abbreviations used in this specification * )
```

$us(\sigma)$ = P(σ, SENDreq), us_i = SENDreq(d_i), and us=any SENDreq.

$ua(\sigma)$=P(σ,SENDack), ua=any SENDreq.

$ur(\sigma)$=P(σ,RECEIVEindic), ur_i =RECEIVEindic(b_i), ur=any ur_i

$sd(\sigma)$=P(σ,[SEND, where id=DATA]),

sd_i =SEND(DATA,d_i ,i) and sd=any sd_i

$rd(\sigma)$=P(σ,[RECEIVE,where id=DATA]),

rd_i =RECEIVE(DATA,b_i ,i) and rd=any rd_i

$sa(\sigma)$=P(σ,[SEND,where id=ACK]), sa_i =SEND(ACK,-,i)

and sa=any sa_i

$ra(\sigma)=P(\sigma,[\text{RECEIVE, where id=ACK}])$, ra_i =RECEIVE(ACK,-,i)

and ra=any ra_i

$[\text{in ESTAB}] \equiv [\sigma 1=\lambda] \vee [ra_{|us(\sigma 1)|-1} \in \sigma 1]$ (state1)

$[\text{in ACK-WAIT}] \equiv [\sigma 1\neq\lambda] \wedge [ra_{|us(\sigma 1)|-1} \notin \sigma 1]$ (state2)

(* Module definitions *)

Module Transmitter (UserTransmitter(Provider);
EntityNetwork(User))

SES : $\sigma 1$: [us,ua,sd,ra,STARTTIMER,STOPTIMER,TIMEOUT]

Transition Axioms:

- $[\text{in ESTAB}] \wedge [\sigma 1=\sigma 0] \wedge [at(us)] \supset$ (t1)
  $O([\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0<us_{|us(\sigma 0)|}>$
  $<sd_{|us(\sigma 0)|},\text{STARTTIMER}>] \wedge$
  $[sd_{|us(\sigma 0)|}] \wedge [\text{STARTTIMER}])$
- $[\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0] \wedge [at(ra_{|us(\sigma 0)|-1})] \supset$ (t2)
  $O([\text{in ESTAB}] \wedge [\sigma 1=\sigma 0<ra_{|us(\sigma 0)|-1}$
  $><\text{STOPTIMER,ua}>] \wedge [\text{STOPTIMER}] \wedge [\text{ua}])$
- $[\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0] \wedge [at(\text{TIMEOUT})] \supset$ (t3)
  $O([\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0<\text{TIMEOUT}>$
  $<sd_{|us(\sigma 0)|-1},\text{STARTIMER}>] \wedge$
  $[sd_{|us(\sigma 0)|-1}] \wedge [\text{STARTIMER}])$
- $[\text{in ESTAB}] \wedge [\sigma 1=\sigma 0] \wedge \sim[at(us)] \supset$ (t4)
  $O([\text{in ESTAB}] \wedge [\sigma 1=\sigma 0])$
- $[\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0] \wedge \sim[at(ra_{|us(\sigma 0)|-1})]$ (t5)
  $\wedge \sim[at(\text{TIMEOUT})] \supset$
  $O([\text{in ACK-WAIT}] \wedge [\sigma 1=\sigma 0])$

• ([at(us)] ∧ ~[at(ra)] ∧ ~[at(TIMEOUT)]) (t6)

∨ (~[at(us)] ∧ [at(ra)] ∧ ~[at(TIMEOUT)])

∨ (~[at(us)] ∧ ~[at(ra)] ∧ [at(TIMEOUT)])

∨ ~([at(us)] ∨ [at(ra)] ∨ [at(TIMEOUT)])

Module Receiver (UserReceiver (Provider);
EntityNetwork (User))

SES : $\sigma 2$: [ur, rd, sa]

Transition Axioms:

• $[\sigma 2=\sigma 0] \wedge [at(rd_{|ur(\sigma 0)|})] \supset$ (r1)

$O([\sigma 2=\sigma 0 < rd_{|ur(\sigma 0)|} >$

$< ur_{|ur(\sigma 0)|} , sa_{|ur(\sigma 0)|} >] \wedge$

$[ur_{|ur(\sigma 0)|}] \wedge [sa_{|ur(\sigma 0)|}])$

• $[\sigma 2=\sigma 0] \wedge [at(rd_k \text{ where } k \neq |ur(\sigma 0)|)] \supset$ (r2)

$O([\sigma 2=\sigma 0 < rd_k >< sa_{|ur(\sigma 0)|-1} >] \wedge$

$[sa_{|ur(\sigma 0)|-1}])$

• $[\sigma 2=\sigma 0] \wedge \sim[at(rd)] \supset O([\sigma 2=\sigma 0])$ (r3)

(*Additional Modules*)

Module User1 (UserTransmitter(User))

Properties:

• $\Box([us] \supset [\sigma 1=\lambda] \vee [ua \in suff(\sigma 1, us)])$ (U1)

• $[\sigma 1=\lambda] \vee [ua \in suff(\sigma 1, us)] \supset <>[us]$ (U2)

Module User2 (UserReceiver(User))

Properties: none

Module System (SystemTransmitter(Provider))

Properties:

- [SETTIMER ϵ suff(σ1, [STOPTIMER,TIMEOUT])] $\supset$ Timer

<>[TIMEOUT]

Module Network (EntityNetwork(Provider)
EntityNetwork(Provider))

Properties:

- $\Box$([rd_i] V [rd_i $\epsilon\sigma$2] $\supset$ ([sd_i $\epsilon\sigma$1] $\wedge$ [data(rd_i)=data(sd_i)])) (N1)
- $\Box$([ra_i] V [ra_i $\epsilon\sigma$1] $\supset$ [sa_i $\epsilon\sigma$2]) (N2)
- $\Box$<>[sd_i] $\supset$ <>[rd_i] (N3)
- $\Box$<>[sa_i] $\supset$ <>[ra_i] (N4)
- $\Box$<>[sd] $\supset$ <>[rd] (N5)
- $\Box$<>[sa] $\supset$ <>[ra] (N6)

APPENDIX 4

Estelle Specification of the Same Protocol

```
(* Types and Channels are the same as in SIGETL specification *)

(* Module definitions *)

Module Transmitter (U:UserTransmitter(Provider);
                    N:EntityNetwork(User);
                    S:SystemTransmitter(User));

Var
   data:data-type;
   send-seq:seq-type;

Stateset
   [ESTAB,ACK-WAIT];

Initialize
   Begin
     state to ESTAB;
     send-seq:=0;
   end

(* transitions *)
trans

from ESTAB
    to ACK-WAIT
       when U.SENDreq(d)
    begin
      data:=d;
      out N.SEND(DATA,data,send-seq);
      out S.STARTTIMER;
    end;

from ACK-WAIT
    to ESTAB
       when N.RECEIVE(ACK,-,seq)
          provided (send-seq=seq)
    begin
      send-seq:=send-seq+1;
      out U.SENDack;
      out STOPTIMER;
    end;

from ACK-WAIT
    to SAME
      when S.TIMEOUT
    begin
      out N.SEND(DATA,data,send-seq);
      out S.STARTTIMER;
    end;
end Module
```

```
Module Receiver(U:UserReceiver(Provider)
                N:EntityNetwork(User))

Var
   recv-seq:seq-type;

Initialize
   begin   recv-seq:=0;  end

(* transitions *)
trans

when N.RECEIVE(DATA,d,seq)
   provided(seq=recv-seq)
   begin
     out N.SEND (ACK,-,seq);
     out U.RECEIVEind(d);
     recv-seq:=recv-seq+1;
   end;

provided otherwise
   begin
     out N.SEND(ACK,-,recv-seq);
   end;
end Module.
```