PROTOCOL SPECIFICATION, TESTING, AND VERIFICATION, C. Sunshine (ed.)
North-Holland Publishing Company
© IFIP, 1982

ON TIME-DEPENDENT COMMUNICATION PROTOCOLS AND THEIR PROJECTIONS

A. Udaya Shankar and Simon S. Lam

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

Time-dependent systems are distributed systems that cannot be adequately modeled without measures of time. Communication protocols offer numerous examples of such systems. To model time-dependent systems, we introduce time variables, to measure elapsed times between system events, and time events, to age the time variables. Time variables in a distributed system model may correspond to timers that are actually implemented in the system. Time variables may also be auxiliary variables that are used for stating assertions about the system's behavior and for representing time constraints of system modules. Two protocol examples exhibiting time-dependent behavior are used to illustrate our model. The first is a simplex stop-and-wait protocol. The second example is a protocol with the functions of connection management and full-duplex data transfer. We also apply the method of protocol projections to this second example, and show well-formed image protocols for the functions of connection management and one-way data transfer. With the help of the image protocols, we state some logical correctness properties of the original protocol with respect to the projected functions.

1. INTRODUCTION

We view a distributed system as a set of modules that interact by exchanging messages [1, 2]. Each event in the system is either an internal event of a module or a message transfer between two adjacent modules. The occurrence of an internal event of a module depends on, and affects the state of that module alone. A message transfer event transports a message from one module to an adjacent module. Its occurrence depends only on the states of the two modules. The message transfer is assumed to occur instantaneously. In this model, a communication channel with non-zero delay is modeled as a module. Non-adjacent modules may interact by exchanging messages via a path of intermediate modules.

We distinguish two categories of distributed systems: time-independent systems and time-dependent systems. Time-independent systems can be adequately modeled by using event counts instead of measures of time [3, 4]. Time-dependent systems, on the other hand, cannot be adequately modeled with event counts only. Some measures of time are needed [5]. Communication protocols offer numerous examples of time-dependent systems.

In a time-independent distributed system, nothing is assumed about the execution speeds of the different modules (except that they are non-zero). Consequently, if an event e_1 in one module is to precede an event e_2 in another module, then the occurrence of e_2 is made to depend upon the reception of a message stating that e_1 has occurred. This message reception would be the last event in a chain of system events leading from e_1 to e_2. (Recall that each system event is either a message transfer between adjacent modules, or an event internal to a module.) Thus, in a time-independent system, the logical correctness properties of interest can be verified without including measures of time in the modeling. (Measures of time may be needed for its performance analysis.) Many models of distributed systems reflect this time-independent behavior.

In a time-dependent system, certain modules are obliged to satisfy some time constraints. For example, given events e_1 and e_2 involving a module, the module ensures that the elapsed time between the occurrences of e_1 and e_2 satisfies certain bounds. Because (physical) time elapses at the same rate everywhere, these time constraints satisfied locally by modules give rise to precedence relations between remote events in different modules. Unlike time-independent systems, such precedence relations are not established through a chain of system events. If such precedence relations are vital to the proper functioning of the distributed system, then many logical correctness properties of importance cannot be verified without including measures of time in the modeling. An example of a time-dependent system involving the synchronization of physical clocks is shown in [3].

In section 2 we introduce our basic protocol model without time variables and time events. A simple communication protocol example illustrates the model and demonstrates the need to include measures of time in modeling. In section 3 we introduce time variables and time events. In section 4 we add time variables and time events to our basic protocol model, and illustrate by modeling the example protocol. In section 5 we consider the modeling of a larger example of a communication protocol with the functions of connection management and full-duplex data transfer. Using this example, we shall also illustrate an application of the method of protocol projections to time-dependent protocols.

2. A TIME-DEPENDENT PROTOCOL

In this paper, we will start with the basic protocol model described in [6, 7]. Later, we will add time variables and time events. Let P_1 and P_2 be two protocol entities that communicate with each other. P_1 sends messages to P_2 through channel C_1, and P_2 sends messages to P_1 through channel C_2. (See Fig. 1.) Messages transmitted through a channel may be reordered, duplicated and/or lost (due to corruption by noise). In addition, messages in the channels have a bounded lifetime. A message that stays in channel C_1 for longer than a specified time constant, denoted by MaxDelay1, is lost. Similarly, the lifetime of every message in channel C_2 is bounded by MaxDelay2.

A channel from one entity to another consists of all buffers and communication media between the entities. We assume that a message can be sent into the channel without any constraint by the channel - i.e., unblocked sends. (Note that most communication protocols have some measure of flow control. As a result, their buffer requirements for messages in transit between entities are bounded. Hence, the assumption of unblocked sends is equivalent to satisfying those buffer requirements.)

At any time let the variable CHANNEL1 denote the sequence of messages in channel C_1. When P_1 sends a message into C_1, that message is appended to the tail of the message sequence in CHANNEL1. When P_2 receives a message from C_1, the first message in CHANNEL1 is removed and transferred to P_2. When messages in C_1 are reordered or duplicated, CHANNEL1 is appropriately updated. When a message in C_1 is lost, that message is deleted from CHANNEL1.

Similarly, the variable CHANNEL2 denotes the sequence of messages in channel C_2, at any time.

Example

Consider the following simplex stop-and-wait protocol [8]. P_1 sends a POLL message to P_2 and receives an ACK message as an acknowledgement. P_1 uses a timer to measure the time elapsed since the POLL was sent. If the POLL remains unacknowledged for longer than a specified time duration, denoted by TimeoutValue, P_1 assumes that either the POLL or the ACK was lost and stops waiting for an acknowledgement. This event is known as a timeout. P_1 can then retransmit the POLL. We assume that P_2 satisfies the following time constraint: the elapsed time between receiving a POLL and sending the ACK is always less than a specified time constant, denoted by MaxReactionTime. TimeoutValue at P_1 satisfies the following condition: TimeoutValue > MaxDelay1 + MaxReactionTime + MaxDelay2.

For this protocol, it is desirable to prove the following Timeout Condition: when there is a timeout at P_1, there is no POLL in C_1, no ACK in C_2, and P_2 is not about to send an ACK.

We will first attempt to model this protocol using only message transfer and internal events. Let the timer used in P_1 be modeled as a variable TIMER taking values from {Off,0,1,2,...}. The value of TIMER indicates how long the timer has been running. TIMER=Off corresponds to the timer being stopped.

The events of P_1 are shown in Table 1. put(CHANNEL1, POLL) appends a POLL message to the tail of the sequence of messages in CHANNEL1. TIMER_TICK represents the aging of the timer. first(CHANNEL2) indicates the first message in the sequence of messages in CHANNEL2. When the first message in CHANNEL2 is ACK, get(CHANNEL2,ACK) removes that message.

P_2 has the Boolean variable ACK_DUE which is true if (and only if) a POLL has been received and the ACK has not been sent. The events of P_2 are shown in Table 2.

By examining this model, we observe that it is possible for the TIMEOUT event in P_1 to occur even though there is a POLL in C_1, or an ACK in C_2, or ACK_DUE in P_2 is true. Thus, we cannot decide from the above model whether or not the simplex stop-and-wait protocol satisfies the Timeout Condition. This model is an inadequate representation of the simplex stop-and-wait protocol.

We conclude that any adequate modeling of the above example has to relate the elapsed times between events in P_1 with the elapsed times between events in C_1, P_2 and C_2. We cannot do this using only message transfer and internal events.

3. TIME VARIABLES AND TIME EVENTS

In the previous section, we observed that in order to model a time-dependent system, the elapsed times at various locations in the distributed system have to be related. In order to relate these elapsed times, we augment our model with time variables and time events.

Time variables are variables that measure elapsed time in integer ticks. Each time variable takes its values from {Off,0,1,2,...}. A time variable is termed inactive if its value is Off, else it is termed active. In modeling a distributed system, whenever it becomes necessary to measure an elapsed time at a module, we include a time variable in the module.

Each time variable has a time event associated with it. If a time variable is active, the occurrence of its associated time event increments the value of the time variable by 1 tick. If the time variable is inactive, the occurrence of its associated time event has no effect on the time variable. The effect on a time variable due to the occurrence of its time event is referred to as aging.

The value of a time variable in a module can also be changed due to the occurrence of message transfer and internal events of that module. Such changes to a time variable are referred to as resets. For an active time variable, the difference between its current value and the value it was last reset to, indicates the time elapsed since the last reset.

We distinguish between two types of time variables: global time variables and local time variables.

In our model, all global time variables are aged by the same time event. That time event is referred to as the global time event. Thus, we assume that all active global time variables are coupled. For an active global time variable the difference between its current value and the value it was last reset to, indicates the global time elapsed since the last reset. The global time event can be used to model physical time (or a clock that is accessible by all modules).

In general, a global time variable does not correspond to an actual timer implemented in the distributed system. Global time variables are typically used as auxiliary variables that are needed to model time constraints satisfied by modules, or for stating assertions about the system's behavior.

Local time variables are used to model the timers and clocks that are implemented in system modules. Every local time variable t is associated with a unique time event, referred to as the <u>local time event</u> of t. t can be aged only by its local time event, and the local time event of t can age only t. This decouples the aging of t from the aging of any other time variable.

The error between the local time measured by t and the corresponding elapsed global time is characterized by an <u>accuracy axiom</u>. For example, with a local time variable t, we can associate a global time variable t*. t* is affected by the global time event just like any other global time variable. Whenever t is reset, t* is reset to the same value. Note that t is active if (and only if) t* is active. Therefore, at any time, t* indicates the value that t should have if t were aged by the global time event. The accuracy axiom of local time variable t can be specified by a bound on t-t* at any time (Off-Off is treated as 0).

For example, the condition $|t-t^*| \leq 1$ specifies a local time variable with no accumulating error. The condition $|t-t^*| \leq \max(a(t^*-t_0),1)$, where t_0 is the value that t was last reset to, specifies a local time variable with maximum relative error a in the clock tick. In this model, neither the local time event of t nor the global time event can occur, if such an occurrence would violate the accuracy axiom. (The meaning of this will be made clear in the discussion on time constraints below.)

The accuracy axiom for each local time variable t cannot be chosen arbitrarily. For example, the accuracy axiom $|t-t^*| \leq 0$ would deadlock the local time event of t and the global time event. In this paper, we insist that the accuracy axiom for any local time variable t is an upper bound on $|t-t^*|$ which is monotonically non-decreasing both in t and in t*, and whose minimum value is 1. This ensures that the time events do not get deadlocked due to unreasonable accuracy axioms. (There are weaker sufficient conditions.)

<u>Time constraints</u>

One of the uses of time variables and time events is to model time constraints in modules. Let e_1 and e_2 denote two message transfer or internal events of a module. Let t be a time variable (local or global) that is reset to 0 by e_1 and reset to Off by e_2. Let D denote a specified time period.

In this situation, one example of a time constraint is that e_2 will not occur until D time units have elapsed since the occurrence of e_1. This can be modeled by including $(t \geq D)$ as part of the enabling condition of e_2.

Another example of a time constraint is that e_2 will occur not later than D time units after the occurrence of e_1. This cannot be modeled by including $(t \leq D)$ in the enabling condition of e_2. (This is because in our model an enabled event does not have to occur.) However, it can be modeled by including $(t < D)$ in the enabling condition of the time event for t. This particular interpretation does not mean that in the distributed system, time comes to a halt whenever t equals D. Rather, it models the guarantee that the module will execute the event e_2 before (or as soon as) t reaches D. Such guarantees by a module are modeled by enabling conditions for time events, and will be referred to as the module's local time axioms.

Note that in the above situation, the module cannot guarantee the local time axiom if e_2 is not enabled some time before t reaches D, or if e_2 is a receive event, or if e_2 is a send event which may be blocked. In short, a local time axiom, like the accuracy axioms, cannot be any arbitrary time constraint. It must be a constraint that the module can guarantee without any cooperation from its environment. If that is the case, then in the modeling of the distributed system, the time events will never be deadlocked.

Note that the accuracy axiom for a local time variable t, considered earlier, is a special case of a local time axiom; the module guarantees that the variable t is incremented at (more or less) the same rate as the global time event occurs. We shall only consider systems whose modules satisfy their local time axioms.

4. PROTOCOL MODEL WITH TIME VARIABLES

We now expand the original protocol model by allowing entities P_1 and P_2 to have both global and local time variables. For notational convenience, define the successor function next on {Off,0,1,2,...} as follows: next(Off)=Off, and next(i)=i+1. Then, the aging of time variable t can be concisely stated by t := next(t).

We now model the bounded lifetimes of messages in channel C_1. With every message in channel C_1, we associate a time variable that indicates the time spent by the message in the channel. At any time, let the variable \$CHANNEL1 denote the sequence of these time variable values in channel C_1. Recall that variable CHANNEL1 denotes the sequence of messages in C_1. Initially, both CHANNEL1 and \$CHANNEL1 are empty sequences. When P_1 sends a message into C_1, that message is appended to the tail of CHANNEL1, and the time value 0 is appended to the tail of \$CHANNEL1. When P_2 receives a message from C_1, the first message in CHANNEL1 is removed and transferred to P_2, and the first time value in \$CHANNEL1 is deleted. When CHANNEL1 undergoes a transformation due to a reordering, duplication or loss channel event, \$CHANNEL1 undergoes an identical transformation. The state of channel C_1 is given by the values of CHANNEL1 and \$CHANNEL1.

The bounded lifetime of messages in channel C_1 is modeled by the following local time axiom: no time value in \$CHANNEL1 exceeds MaxDelay1. We abbreviate this as: \$CHANNEL1 $\leq$ MaxDelay1. Finally, for notational convenience, we assume that all the time variables in the channels are global time variables. Then, the action of the global time event would be to increment every value in \$CHANNEL1 by 1 tick. This is abbreviated as \$CHANNEL1 := next(\$CHANNEL1).

Similarly, in channel C_2, we have the variable sequence of time values in \$CHANNEL2, and the local time axiom \$CHANNEL2 $\leq$ MaxDelay2. The action of the global time event is \$CHANNEL2 := next(\$CHANNEL2).

Example

We now illustrate this model by modeling the simplex stop-and-wait protocol example discussed earlier. Let global time variable \$REACTIONTIME in P_2 indicate the elapsed time between reception of a POLL and transmission of the ACK. The time constraint satisfied by P_2 is modeled by the local time axiom (\$REACTIONTIME $\leq$ MaxReactionTime). Let local time variable TIMER model the timer used in P_1 to measure the time elapsed since sending a POLL. Let \$TIMER denote the global time variable associated with TIMER. We assume that TIMER satisfies the accuracy axiom |TIMER-\$TIMER| $\leq$ max(a(\$TIMER),1), where a is the relative error in each tick of TIMER.

The events of entity P_1 and entity P_2 are shown in Table 3 and Table 4 respectively. The time events are shown in Table 5. TIMER_TICK is the local time event for TIMER. GLOBAL_TICK is the global time event for the protocol system. (\$CHANNEL1 < MaxDelay1) and (\$CHANNEL2 < MaxDelay2) are from the local time axioms of the channels. ($REACTIONTIME < MaxReactionTime) is from the local time axiom of P_2.

In this model, we can prove the following: If TimeoutValue > (1+a).(MaxDelay1 + MaxReactionTime + MaxDelay2), then when there is a timeout at P_1, there is no POLL in CHANNEL1, there is no ACK in CHANNEL2, and ACK_DUE = False. Thus, this model of the simplex stop-and-wait protocol is adequate in that it allows us to verify the Timeout Condition, and enables us to calculate the proper timeout value.

5. A PROTOCOL WITH CONNECTION MANAGEMENT AND FULL-DUPLEX DATA TRANSFER FUNCTIONS

In this section, we illustrate the application of protocol projections to a time-dependent protocol. The method of protocol projections has been described elsewhere [6, 7] and will not be repeated here. The protocol to be considered performs connection management and full-duplex data transfer between entities P_1 and P_2. The protocol has the following three distinguishable functions: connection management between P_1 and P_2, one-way data transfer from P_1 to P_2, and one-way data transfer from P_2 to P_1. We will show faithful image protocols for the first two functions. We emphasize that this protocol is provided here to illustrate the usefulness of projections, and not as an example of a "good" communication protocol.

The connection management function of the protocol is responsible for opening/closing the data link between P_1 and P_2. Data transfer is possible only when the data link is open. The link is initially closed. P_1 can open the link by sending the connect command message (CONN) to P_2, and receiving the response message (RESP). The data transfer variables (to be described later) at P_1 and P_2 are initialized each time the link is opened. P_1 can close the link by sending the disconnect command message (DISC) to P_2, and receiving the response message (RESP). In both cases P_2 guarantees to respond within a specified time, denoted by MaxReactionTime. Because the channels can lose messages, a command message that is unacknowledged for longer than a specified time, denoted by TimeoutValue, is retransmited.

When the data link between P_1 and P_2 is open, P_1 sends data blocks to P_2 and receives acknowledgements, and P_2 sends data blocks to P_1 and receives acknowledgements. Each entity has at most one data block unacknowledged at any time. A data block that has not been acknowledged within a time duration equal to TimeoutValue is retransmited. To avoid duplicate delivery, each data block sent is accompanied by a sequence number which is either 0 or 1. The acknowledgement for a received data block can be sent alone as an (ACK) message, or it can be piggy-backed with a data block. Each entity guarantees to acknowledge a received data block within MaxReactionTime.

A data block d accompanied by sequence number ns can be sent as either the message (DATA,ns,d) or as the message (DATA&ACK,ns,d). Here, DATA is a (character string) constant identifying the message as a sequenced data block; DATA&ACK is a (character string) constant identifying the message as a sequenced data block with a piggy-backed acknowledgement for a data block received in the opposite direction. Let DATASET denote the set of data blocks that can be sent in the protocol. The set of messages that can be sent by P_1 is given by

$$\begin{aligned} M_1 = {} & \{(\text{CONN}),\ (\text{DISC})\} \cup \{(\text{ACK})\} \\ & \cup \{(\text{DATA},ns,d) : ns \in \{0,1\},\ d \in \text{DATASET}\} \\ & \cup \{(\text{DATA\&ACK},ns,d) : ns \in \{0,1\},\ d \in \text{DATASET}\} \end{aligned}$$

The set of messages that can be sent by P_2 is given by

$$\begin{aligned} M_2 = {} & \{(\text{RESP})\} \cup \{(\text{ACK})\} \\ & \cup \{(\text{DATA},ns,d) : ns \in \{0,1\},\ d \in \text{DATASET}\} \\ & \cup \{(\text{DATA\&ACK},ns,d) : ns \in \{0,1\},\ d \in \text{DATASET}\} \end{aligned}$$

We now examine the variables in entity P_1. P_1 has an infinite array of data blocks, SOURCE[i] for i = 0,1,2,..., destined for P_2, and an infinite array SINK[i] for i = 0,1,2,..., to store data blocks received from P_2. Additionally, P_1 has the following variables: MODE which takes values from {Open, Closed, Opening, Closing}, a local time variable TIMER, VS and VR which are nonnegative integers, ACK_DUE which is a Boolean variable, and global time variable $REACTIONTIME. MODE indicates the status of the data link as perceived by P_1. TIMER is active if (and only if) either a CONN, DISC, DATA or DATA&ACK message has been sent but not yet acknowledged. It indicates the time duration for which the message sent has been outstanding. We denote its associated global timer by $TIMER, and assume an accuracy axiom $|\text{TIMER}-\$\text{TIMER}| \leq \max(a(\$\text{TIMER}),1)$. VS points to the data block in SOURCE to be sent next. VR points to the position in SINK to be next filled. ACK_DUE is true if (and only if) a received data block has to be acknowledged. $REACTIONTIME is active if (and only if) ACK_DUE is True and indicates the time elapsed since reception of the data block. The events of entity P_1 are shown in Table 6.

The variables of entity P_2 are similar to those of P_1. That is, P_2 has the variables SOURCE, SINK, MODE, TIMER, \$TIMER, VS, VR, ACK_DUE and \$REACTIONTIME. For convenience, we have omitted qualifiers (1 or 2) for these variables and we shall omit them as long as it is clear whether we are referring to P_1 or P_2. The meaning of MODE, SOURCE, SINK, VS, VR and ACK_DUE in P_2 are similar to that in P_1. TIMER is used to measure the time for which a sent data block is unacknowledged. \$REACTIONTIME is used to measure the time between reception of either a (CONN), (DISC) or data block, and the transmission of the appropriate acknowledgement. The events of P_2 are shown in Table 7.

Initially, in both P_1 and P_2, MODE = Closed. At P_1, when the link is opened (i.e., a (RESP) acknowledgement is received for an outstanding (CONN) command), the data transfer variables are initialized as follows: VS=VR=0, ACK_DUE=Off, \$REACTIONTIME=Off. A corresponding initialization is done at P_2 when a (RESP) acknowledgement to a received (CONN) is sent. At this time, in each entity, SINK is empty and SOURCE equals some infinite array of data blocks. SOURCE does not change its value while the link is open.

The time events of the protocol are shown in Table 8. The qualifiers (1 or 2) for the time variables in Table 8 identify the entity to which they belong.

The example protocol has three functions corresponding to connection management between P_1 and P_2, one-way data transfer from P_1 to P_2, and one-way data transfer from P_2 to P_1. We shall show image protocols for the first two functions.

Image protocol for one-way data transfer from P_1 to P_2

For one-way data transfer from P_1 to P_2, observe that the variables VR, ACK_DUE, \$REACTIONTIME and SINK in P_1, and the variables VS, TIMER (and its associated \$TIMER), and SOURCE in P_2 are not needed. The variable MODE in P_1 and the variable MODE in P_2 are required since they are involved in the initialization of data transfer. Thus, the variables of the projected function (referred to as function variables) are MODE, TIMER (and its associated \$TIMER), VS and SOURCE in P_1 and MODE, ACK_DUE, \$REACTIONTIME, VR and SINK in P_2. At any time, the image state (i.e. state of the projected function) of P_1 is given by the values of MODE, TIMER, \$TIMER, VS and SOURCE. Let S_1' denote the image state space of P_1. Similarly, the image state of P_2 is given by the values of MODE, ACK_DUE, \$REACTIONTIME, VR and SINK. Let S_2' denote the image state space of P_2.

By examining the state changes in the image spaces S_1' and S_2' due to send and receive events in Tables 6 and 7, the following can be shown about messages in M_1. The message (ACK) has a null image. The messages (DATA,ns,d) and (DATA&ACK,ns,d) are equivalent; let (DATA′,ns,d) denote their image. No other messages in M_1 are equivalent. Let (CONN′) and (DISC′) denote the images of (CONN) and (DISC). Thus,
$M_1' = \{(\text{CONN}'),(\text{DISC}')\} \cup \{(\text{DATA}',ns,d) : ns \in \{0,1\},\ d \in \text{DATASET}\}$.

Similarly, the following can be shown about messages in M_2. All (DATA,ns,d) messages have the null image. All (DATA&ACK,ns,d) messages are equivalent to the ACK message; denote their image by (ACK′). Let (RESP′) denote the image of (RESP). Thus $M_2' = \{(\text{RESP}'), (\text{ACK}')\}$.

The system events of the image protocol for the projected function are shown in Tables 9 and 10. The time events for the image protocol are shown in Table 11.

For this image protocol, if we assume that the channels do not duplicate messages, and (TimeoutValue > (1+a) (MaxDelay1 + MaxReactionTime + MaxDelay2)) then the following logical correctness properties can be shown to hold.

If MODE = Open in P_1 then the following assertions hold:

1. SINK[i] = SOURCE[i] for $0 \leq i < \text{VR}$

2. $\text{VR} \geq \text{VS} \geq \text{VR} - 1$

```
3. (DATA′,ns,d) in CHANNEL1 => (TIMER ≠ Off)
            and (TIMER ≤ 1 + (1+a)(MaxDelay1))
            and (d = SOURCE[VS]) and (ns = VS mod 2)
            and (exactly one DATA′ message in CHANNEL1)
            and (not ACK_DUE) and (VS = VR or VS = VR - 1)
            and (no ACK′ message in CHANNEL2)

4. ACK_DUE => (TIMER ≠ Off)
            and (TIMER ≤ 1
                + (1+a)(MaxDelay1 + MaxReactionTime))
            and (no DATA′ message in CHANNEL1)
            and (VS = VR-1) and (no ACK′ message in CHANNEL2)

5. (ACK′) in CHANNEL2 => (TIMER ≠ Off)
            and (TIMER ≤ 1
                + (1+a)(MaxDelay1 + MaxReactionTime + MaxDelay2))
            and (no DATA′ message in CHANNEL1)
            and (VS = VR - 1)
            and (exactly one ACK′ message in CHANNEL2)
```

It can also be shown that the image protocol is well-formed. As a result, under the same conditions stated on channels and timeout value, these assertions are also valid for the original protocol with respect to the function of one-way data transfer.

<u>Image protocol for connection establishment</u>

For the function of connection management between P_1 and P_2, we are interested in assertions involving the variables MODE, TIMER and $TIMER in P_1, and MODE and $REACTIONTIME in P_2. However, an image protocol involving only these function variables would not be well-formed. In order to make it well-formed, ACK_DUE in P_2 has to be included as a function variable. Thus the function variables are MODE, TIMER and $TIMER in P_1 and MODE, $REACTIONTIME and ACK_DUE in P_2. These variables define the image state spaces of P_1 and P_2.

By examining the state changes in the image state spaces due to send and receive events, the following can be shown about messages in M_1. The message (ACK) has a null image. All DATA and DATA&ACK messages are equivalent; let (DATA′) denote their image. Note that DATA messages do not leave a null image because they affect TIMER. Let the image of (CONN) be (CONN′) and the image of (DISC) be (DISC′). Thus, $M_1' = \{(\text{CONN}'), (\text{DISC}'), (\text{DATA}')\}$.

Similarly, the following can be shown about the messages in M_2. All DATA messages have a null image. All DATA&ACK messages are equivalent to the ACK message; let (ACK′) denote their image. Note that DATA&ACK messages and the ACK message do not have a null image because they affect ACK_DUE. Let the image of (RESP) be (RESP′). Thus $M_2' = \{(\text{RESP}'), (\text{ACK}')\}$.

The events of the image protocol for the projected function are shown in Tables 12 and 13. The time events for the image protocol are identical to those shown in Table 11.

For this image protocol, if we assume that the channels do not duplicate messages, and (TimeoutValue > (1+a) (MaxDelay1 + MaxReactionTime + MaxDelay2)), then the following logical correctness properties can be shown to hold.

1. If MODE in P_1 is Open then MODE in P_2 is Open
2. If MODE in P_1 is Closed
   then (MODE in P_2 is Closed)
   and (both CHANNEL1 and CHANNEL2 are empty)
   and (ACK_DUE in P_2 is False)

The image protocol can also be shown to be well-formed. Hence, under the same conditions stated on channels and timeout value, these assertions are also valid for the original protocol with respect to the function of connection management.

6. CONCLUSIONS

Time-dependent systems are distributed systems that cannot be adequately modeled without measures of time; in particular, system event counts cannot be substituted for measures of elapsed time. Communication protocols offer numerous examples of time-dependent protocols. To model time-dependent systems, we have introduced time variables, to measure elapsed times between system events, and time events to age the time variables. Using time variables and time events, we can model modules that satisfy time constraints and state assertions about the time-related behavior of a system of modules. Two types of time variables have been distinguished: global time variables and local time variables. Global time variables are coupled and aged under the action of a single global time event. They are typically used as auxiliary variables for stating assertions and time constraints. On the other hand, local time variables correspond to timers that are actually implemented in the distributed system being modeled. Local time variables are not coupled, and each is aged by its own local time event. The elapsed time measured by a local time variable can differ from the elapsed time measured by the global time event. The difference is guaranteed to be within certain bounds that are specified formally by an accuracy axiom.

We have incorporated time variables and time events into a protocol model where protocol entities are connected by channels. Messages travelling within a channel have a bounded lifetime. The channels can also lose, reorder and duplicate messages in transit. Two protocol examples exhibiting time-dependent behavior were employed to illustrate the model. The first example is a simplex stop-and-wait protocol. The second example is a protocol with the functions of connection management and full-duplex data transfer. We have also applied the method of protocol projections to this second example, and found well-formed image protocols for the functions of connection management and one-way data transfer. With the help of the image protocols, we stated some logical correctness properties of the original protocol with respect to the projected functions.

REFERENCES

[1] Hoare, C. A. R., "Communicating Sequential Processes," Comm. ACM, August 1978.

[2] Keller, R. M., "Formal Verification of Parallel Programs," Comm. ACM, July 1976.

[3] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Comm. ACM, July 1978.

[4] Ricart, G. and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Comm. ACM, January 1981.

[5] Wirth, N., "Toward a Discipline of Real-Time Programming," Comm. ACM, August 1977.

[6] Lam, S. S. and A. U. Shankar, "Verification of communication protocols via protocol projections," Proc. INFOCOM ´82, Las Vegas, April 1982.

[7] Lam, S. S. and A. U. Shankar, "An illustration of protocol projections," Proc. Second International Workshop on Protocol Specification, Testing, and Verification, Idyllwild, May 1982.

[8] Tanenbaum, A. S., Computer Networks, Prentice-Hall, New Jersey, 1981, (pp. 143-144).

ACKNOWLEDGEMENT

The research in this paper was supported by National Science Foundation Grant No. ECS78-01803.

Event Name	Enabling Condition	Action
1. SEND_POLL	TIMER = Off	put(CHANNEL1, POLL); TIMER := 0
2. TIMER_TICK	TIMER ≠ Off	TIMER := TIMER + 1
3. TIMEOUT	TIMER > TimeoutValue	TIMER := Off
4. REC_ACK	first(CHANNEL2) = ACK	get(CHANNEL2, ACK); TIMER := Off

TABLE 1. Events of P_1 for simplex stop-and-wait protocol

1. REC_POLL	first(CHANNEL1) = POLL	get(CHANNEL1, POLL); ACK_DUE := True
2. SEND_ACK	ACK_DUE = True	put(CHANNEL2, ACK); ACK_DUE := False

TABLE 2. Events of P_2 for simplex stop-and-wait protocol

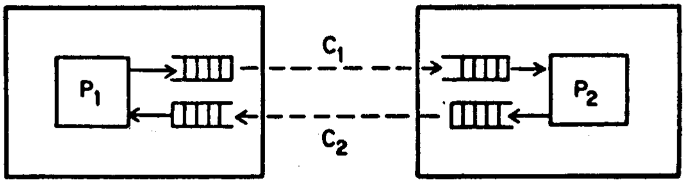

Fig. 1. Components of the protocol model.

Event Name	Enabling Condition	Action
1. SEND_POLL	TIMER = Off	put(CHANNEL1, POLL); TIMER := 0 {\$TIMER := 0}
2. TIMEOUT	TIMER > TimeoutValue	TIMER := Off {\$TIMER := Off}
3. REC_ACK	first(CHANNEL2) = ACK	get(CHANNEL2, ACK); TIMER := Off {\$TIMER := Off}

TABLE 3. Events of P_1 for simplex stop-and-wait protocol

1. REC_POLL	first(CHANNEL1) = POLL	get(CHANNEL1, POLL); ACK_DUE := True {\$REACTIONTIME := 0}
2. SEND_ACK	ACK_DUE	put(CHANNEL2, ACK); ACK_DUE := False {\$REACTIONTIME := Off}

TABLE 4. Events of P_2 for simplex stop-and-wait protocol

1. TIMER_TICK	(TIMER-\$TIMER) < max(a(\$TIMER),1)	TIMER := next(TIMER)
2. GLOBAL_TICK	(TIMER-\$TIMER) > -max(a(\$TIMER),1) and (\$CHANNEL1 < MaxDelay1) and (\$CHANNEL2 < MaxDelay2) and (\$REACTIONTIME < MaxReactionTime)	\$TIMER := next(\$TIMER); \$CHANNEL1 := next(\$CHANNEL1); \$CHANNEL2 := next(\$CHANNEL2); \$REACTIONTIME := next(\$REAC-TIONTIME)

TABLE 5. Time events for simplex stop-and-wait protocol

Event Name	Enabling Condition	Action
1. SEND_CONN	TIMER = Off	put(CHANNEL1, (CONN)); MODE := Opening; TIMER := 0 {$TIMER := 0}
2. SEND_DISC	TIMER = Off	put(CHANNEL1, (DISC)); MODE := Closing; TIMER := 0 {$TIMER := 0}
3. REC_RESP	first(CHANNEL2) = RESP	get(CHANNEL2, (RESP)); TIMER := Off {$TIMER := Off}; If MODE = Closing then MODE := Closed; If MODE = Opening then begin MODE := Open; VS := 0; VR := 0; ACK_DUE := False {$REACTIONTIME := Off} end
4. TIMEOUT	TIMER > TimeoutValue	TIMER := Off {$TIMER := Off}
5. SEND_DATA	MODE = Open and TIMER = Off	SDATA := SOURCE[VS]; NS := VS MOD 2; put(CHANNEL1, (DATA, NS, SDATA)); TIMER := 0 {$TIMER := 0}
6. SEND_DATA&ACK	MODE = Open and TIMER = Off and ACK_DUE	SDATA := SOURCE[VS]; NS := VS MOD 2; put(CHANNEL1, (DATA&ACK, NS, SDATA)); TIMER := 0 {$TIMER := 0}; ACK_DUE := False {$REACTIONTIME := Off}
7. SEND_ACK	MODE = Open and ACK_DUE	put(CHANNEL1, (ACK)); ACK_DUE := False {$REACTIONTIME := Off}
8. REC_DATA	MODE = Open and first(CHANNEL2)=DATA	get(CHANNEL2, (DATA, NR, RDATA)); ACK_DUE := True {$REACTIONTIME := 0}; If NR = VR MOD 2 then begin SINK[VR] := RDATA; VR := VR + 1 end
9. REC_DATA&ACK	MODE = Open and first(CHANNEL2)=DATA&ACK	get(CHANNEL2, (DATA&ACK, NR, RDATA)); ACK_DUE := True {$REACTIONTIME := 0} If NR = VR MOD 2 then begin SINK[VR] := RDATA; VR := VR + 1 end; VS := VS + 1; TIMER := Off {$TIMER := Off}
10. REC_ACK	MODE = Open and first(CHANNEL2)=ACK	get(CHANNEL2, (ACK)); VS := VS + 1; TIMER := Off {$TIMER := Off}

TABLE 6. Events of P_1 in the connection management and data transfer protocol

Event Name	Enabling Condition	Action

1. REC_CONN	first(CHANNEL1) = CONN	get(CHANNEL1, (CONN));
		MODE := Opening
		{$REACTIONTIME := 0}

2. REC_DISC	first(CHANNEL1) = DISC	get(CHANNEL1, (DISC));
		MODE := Closing
		{$REACTIONTIME := 0}

3. SEND_RESP	MODE = Opening
	or MODE = Closing
		put(CHANNEL2, (RESP))
		{$REACTIONTIME := Off}
		If MODE = Closing
		then MODE := Closed;
		If MODE = Opening
		then begin MODE := Open;
		VS := 0; VR := 0;
		ACK_DUE := False;
		TIMER := Off
		{$TIMER := Off}
		end

4-10. Events SEND_DATA, SEN_DATA&ACK, SEND_ACK, REC_DATA, REC_DATA&ACK, REC_ACK and TIMEOUT are as described in Table 6 (with CHANNEL1 and CHANNEL2 interchanged).

TABLE 7. Events of entity P_2 in the connection management and data transfer protocol

Event Name	Enabling Condition	Action
1. TIMER1_TICK	TIMER1-$TIMER1 < max(a($TIMER1),1)	TIMER1 := next(TIMER1)
2. TIMER2_TICK	TIMER2-$TIMER2 < max(a($TIMER2),1)	TIMER2 := next(TIMER2)
3. GLOBAL_TICK	(TIMER1-$TIMER1 > -max(a($TIMER1),1) and (TIMER2-$TIMER2 > -max(a($TIMER2),1) and ($REACTIONTIME1<MaxReactionTime) and ($REACTIONTIME2<MaxReactionTime) and ($CHANNEL1 < MaxDelay1) and ($CHANNEL2 < MaxDelay2)	$TIMER1 := next($TIMER1); $TIMER2 := next($TIMER2); $REACTIONTIME1 := next($REACTIONTIME1); $REACTIONTIME2 := next($REACTIONTIME2); $CHANNEL1 := next($CHANNEL1); $CHANNEL2 := next($CHANNEL2);

TABLE 8. Time events for the connection management and data transfer protocol

Event Name	Enabling Condition	Action

1. SEND_CONN′	TIMER = Off	put(CHANNEL1, (CONN′));
		MODE := Opening;
		TIMER := 0 {$TIMER := 0}

2. SEND_DISC′	TIMER = Off	put(CHANNEL1, (DISC′));
		MODE := Closing;
		TIMER := 0 {$TIMER := 0}

3. REC_RESP′	first(CHANNEL2) = RESP′	get(CHANNEL2, (RESP′));
		TIMER := Off {$TIMER := Off};
		If MODE = Closing
		   then MODE := Closed;
		If MODE = Opening
		   then begin MODE := Open;
		               VS := 0
		        end

4. TIMEOUT′	TIMER > TimeoutValue	TIMER := Off {$TIMER := Off}

5. SEND_DATA′	MODE = Open
	and TIMER = Off
		SDATA := SOURCE[VS];
		NS := VS MOD 2;
		put(CHANNEL1, (DATA′, NS, SDATA));
		TIMER := 0 {$TIMER := 0}

6. REC_ACK′	MODE = Open
	and first(CHANNEL2)=ACK′
		get(CHANNEL2, (ACK′));
		VS := VS + 1;
		TIMER := Off {$TIMER := Off}

TABLE 9. Events of P_1' in image protocol for one-way data transfer

Event Name	Enabling Condition	Action

1. REC_CONN´	first(CHANNEL1) = CONN´	get(CHANNEL1, (CONN´));
		MODE := Opening
		{$REACTIONTIME := 0}

2. REC_DISC´	first(CHANNEL1) = DISC´	get(CHANNEL1, (DISC´));
		MODE := Closing
		{$REACTIONTIME := 0}

3. SEND_RESP´	MODE = Opening
	or MODE = Closing
		put(CHANNEL2, (RESP´))
		{$REACTIONTIME := Off}
		If MODE = Closing
		   then MODE := Closed;
		If MODE = Opening
		   then begin MODE := Open;
		              VR := 0;
		              ACK_DUE := False
		         end

4. REC_DATA´	MODE = Open
	and first(CHANNEL1)=DATA´
		get(CHANNEL1, (DATA´, NR, RDATA));
		ACK_DUE := True {$REACTIONTIME := 0};
		If $\overline{\text{NR}}$ = VR MOD 2
		   then begin SINK[VR] := RDATA;
		              VR := VR + 1
		         end

5. SEND_ACK´	MODE = Open and ACK_DUE	put(CHANNEL2, (ACK´));
		ACK_DUE := False {$REACTIONTIME := Off}

TABLE 10. Events of P_2' in image protocol for one-way data transfer

Event Name	Enabling Condition	Action
1. TIMER1_TICK	TIMER1-$TIMER1 < max(a($TIMER1),1)	TIMER1 := next(TIMER1)
2. GLOBAL_TICK	(TIMER1-$TIMER1 > -max(a($TIMER1),1) and ($REACTIONTIME2<MaxReactionTime) and ($CHANNEL1 < MaxDelay1) and ($CHANNEL2 < MaxDelay2)	$TIMER1 := next($TIMER1); $REACTIONTIME2 := next($REAC-TIONTIME2); $CHANNEL1 := next($CHANNEL1); $CHANNEL2 := next($CHANNEL2)

TABLE 11. Time events for image protocol of one-way data transfer

Event Name	Enabling Condition	Action

1. SEND_CONN´	TIMER = Off	put(CHANNEL1, (CONN´));
		MODE := Opening;
		TIMER := 0 {$TIMER := 0}

2. SEND_DISC´	TIMER = Off	put(CHANNEL1, (DISC´));
		MODE := Closing;
		TIMER := 0 {$TIMER := 0}

3. REC_RESP´	first(CHANNEL2) = RESP´	get(CHANNEL2, (RESP´));
		TIMER := Off {$TIMER := Off}
		If MODE = Closing
		   then MODE := Closed;
		If MODE = Opening
		   then MODE := Open

4. TIMEOUT´	TIMER > TimeoutValue	TIMER := Off {$TIMER := Off}

5. SEND_DATA´	MODE = Open
	and TIMER = Off
		put(CHANNEL1, (DATA´));
		TIMER := 0 {$TIMER := 0}

6. REC_ACK´	MODE = Open
	and first(CHANNEL2)=ACK´
		get(CHANNEL2, (ACK´));
		TIMER := Off {$TIMER := Off}

TABLE 12. Events of P_1' in image protocol for connection management function

Event Name	Enabling Condition	Action

1. REC_CONN´	first(CHANNEL1) = CONN´	get(CHANNEL1, (CONN´));
		MODE := Opening
		{$REACTIONTIME := 0}

2. REC_DISC´	first(CHANNEL1) = DISC´	get(CHANNEL1, (DISC´));
		MODE := Closing
		{$REACTIONTIME := 0}

3. SEND_RESP´	MODE = Opening
	or MODE = Closing
		put(CHANNEL2, (RESP´))
		{$REACTIONTIME := 0}
		If MODE = Closing
		   then MODE := Closed;
		If MODE = Opening
		   then MODE := Open

4. REC_DATA´	MODE = Open
	and first(CHANNEL1)=DATA´
		get(CHANNEL1, (DATA´))
		{$REACTIONTIME := 0}}
		ACK_DUE := True

5. SEND_ACK´	MODE = Open
	and ACK_DUE
		put(CHANNEL2, (ACK´))
		{$REACTIONTIME := Off}
		ACK_DUE := False

TABLE 13. Events of P_2' in image protocol for connection management function