Synthesizing Protocol Specifications from Service Specifications in Timed Extended Finite State Machines *

Jun-Cheol Park   Raymond E. Miller

Department of Computer Science
University of Maryland, College Park, MD 20742
{jcpark, miller}@cs.umd.edu

September 23, 1996

Abstract

We propose a specification model and present a method to algorithmically derive a protocol specification from a service specification based on the model. Unlike the previous models based on finite state machines, the proposed model can explicitly express concurrency, synchronization, and timing requirements such as delays and timeouts. We assume that there exists a reliable communication channel between any two protocol entities and the maximum delay for each channel is bounded by a positive constant. Because of the variable nature of the communication delays along with the time constraints associated with events, no protocol specification can fully simulate the service specification. The proposed method derives a protocol specification that is optimal in the sense that it provides the largest possible subset of the service specification under the communication delay constraints. We also give a method to derive a sub specification from a service specification and a maximum communication delay of each channel such that the sub specification, but no superset of it, can be simulated by the derived protocol specification.

1 Introduction

There are two common approaches for designing communication protocols: analysis and synthesis [4]. In the analysis method, the protocol designer begins with a preliminary version of the protocol usually obtained by ad hoc methods. This approach usually results in an incomplete and erroneous design, which is followed by an analysis and redesign process. The sequence of redesign, analysis, and error correction is applied iteratively until an error-free design is obtained. In the synthesis method, a partially specified or incomplete protocol design is completed incrementally, or automatically, without any interaction by the designer such that as the synthesis process proceeds correctness is maintained. The process ends with a design that provides the set of specified services. Therefore, no further verification of the protocol design is necessary as in the analysis approach.

Much research has been done in the area of protocol synthesis. The reader may refer to [1] for a survey and assessment of several synthesis methods. The synthesis methods can be

*This research was supported by NASA Grant No. NAG 5-2648 and NSF Grant No. NCR9506039.

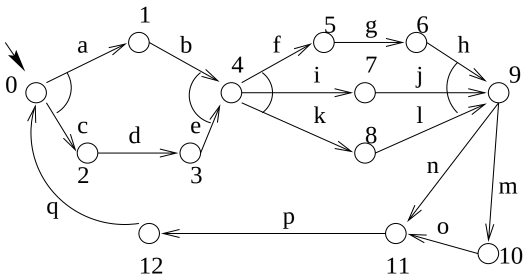
classified by the modeling formalism. The models include finite state machines [2, 3, 7], Petri nets [5], LOTOS-like models [8, 9], etc. However, these methods do not provide or represent the notion of time, which is important for the proper functioning of communication systems. Recently, a few methods [10, 11] have been proposed that derive protocol specifications from timed service specifications. In [10], a model based on finite state machines has been proposed for specifying timing requirements by using a global clock, timers, and counters. The method derives the protocol and medium specifications from a service specification written as a set of timed transitions. The model represents temporal requirements between remote as well as consecutive events, which necessarily introduces an exponential increase in the number of timers. On the other hand, [11] has proposed a model based on LOTOS that restricts the time constraints of service specifications while fixing the maximum delay of the communication media in the sense that the model can specify a complicated order of events in a structural way.

In this paper, we propose a model called timed extended finite state machine(TEFSM) based on the extended finite state machine(EFSM) model to deal with timed operations between consecutive events. Delay and timeout are certainly two of the most useful timed operations. To represent these events, we use the notion of a timed transition in our model by associating a time interval $[l, u]$ with the transition. The lower and upper time limits are measured with respect to a global clock, and can thus be used in modeling timed properties including delays and timeouts. The notion of a timed transition is not new, and our model is in fact inspired by a few previous works [12, 13]. The main difference is that our model can express concurrency and synchronization among protocol entities explicitly while these previous models could not. For synthesis, we assume that each communication channel is error-free and has a propagation delay bounded by a constant, as in [11]. We present an algorithm that derives a protocol specification from a service specification modeled as a TEFSM when an upper bound of delay for each channel is given.

The paper is organized as follows. Section 2 describes our TEFSM model. Section 3 formalizes the protocol synthesis problem and gives some notation. In section 4, we present an algorithm for deriving the protocol specification of a protocol entity from a service specification and prove the correctness of the algorithm by investigating the relationship between the service specification and the protocol specification. In section 5, we demonstrate the applicability of our synthesis method by giving an example. Section 6 gives some concluding remarks and discusses areas requiring future work.

2 The Model

The TEFSM model is designed as a method for the formal description of service and protocol specifications. A TEFSM M is defined by a tuple $< S, FJ, V, T, \delta, s_0 >$, where (1) S is a nonempty set of states and for each $s \in S$, s is a choice, fork, join, or fork/join state. To represent a possible parallel execution among protocol entities, M explicitly uses a pair $(fork, join)$ of states such that the control flows(directed paths) from $fork$ to $join$ can be executed concurrently and independently. If a join state is also a fork state which is matched with another join state, we call it fork/join state. Note that for each fork state, there exists a unique join state and vice versa. All states other than fork, join, or fork/join states are choice states. If more than one outgoing transition exists for a choice or a join state, M can arbitrarily choose one transition and execute it. See Figure 1 for an example of the classification of states; (2) FJ is a finite(possibly empty) set of (fork,join) pairs in M; (3) V is a set of variables including input, output, and local variables, denoted by I, O, and L, respectively; (4) T is a set of transitions and each transition $t \in T$ is a 6-tuple $< head(t), tail(t), P(t), E(t), host(t), [min_t, max_t] >$, where $head(t)$ and $tail(t)$ are respectively the head and the tail state of t, $P(t)$ is the enabling predicate in V associated with t, $E(t)$ is the event on V associated with t, $host(t)$ is the protocol entity that executes t, and $[min_t, max_t]$ is the time interval associated with t such that t can be executed after min_t, but no later than max_t has passed since the time of visit to $head(t)$ state. If a time interval is not specified explicitly, the default interval $[0, \infty)$ is assumed. An event is a partial function: $E(t) : L \times I \rightarrow L \times O$. We denote $a^i[min_t, max_t]$ as the transition t with the action a and the protocol entity i when the other components of t are of no concern; (5) δ is a partial state transition function such that $\delta : S \times T \rightarrow S$, and (6) s_0 is an initial state.



fork state(s) : 0
join state(s) : 9
fork/join state(s) : 4
choice state(s) : all other states

(a, b) and (c, d, e) can be executed concurrently.
(f, g, h), (i, j) and (k, l) can be executed concurrently.

Figure 1: The Classification of States : An Example

The execution of a transition t is an instantaneous action in which both the event associated with t and the state change to the tail state of t occur simultaneously. A transition t in a TEFSM M must be executed within its time interval if (1) M is in $head(t)$; (2) A finite time interval is associated with t; and (3) t is enabled throughout the time interval.

A protocol is specified as a set of processes $< PS_0, PS_1, ..., PS_n >$ where each process PS_i is a TEFSM that can communicate with other processes through $FIFO$ channels. Note that each process PS_i has only choice states since no concurrent execution is allowed.

A channel from PS_i to PS_j has a maximum delay $D_{i,j}$ such that the message transmissions are carried out within $D_{i,j}$, i.e., $0 < delay(i,j) \leq D_{i,j}$.

A sending transition $s_{ij}(m)$ denotes the nonblocking transmission of the message m from PS_i to PS_j and a receiving transition $r_{ij}(m)$ denotes the blocking reception of the message m coming from PS_i to PS_j. Note that $s_{ij}(m)$ and $r_{ij}(m)$ are dual events.

Since a TEFSM can be described as a labeled directed graph, we will use state and node, and event and transition interchangeably for the rest of the paper.

3 Synthesis Problem

We assume that there is a global digital clock that ticks at a constant frequency and all of the relative times of the protocol entities refer to this clock.

Notation 1 (1) Given a finite sequence σ, $first(\sigma)$ and $last(\sigma)$ denote the first and the last element of σ, respectively. Denote $\cdot$ for concatenation. ϵ denotes an empty sequence, $|\epsilon| = 0$. (2) Given a sequence of events σ, we denote $\sigma \downarrow_i$ for the projection of σ onto the events of PS_i.

(3) For a state s in a TEFSM, $t(s)$ denotes the time when the machine has visited the state. (4) Given a state s in a TEFSM, $IN(s)$ and $OUT(s)$ denote the sets of the incoming and the outgoing transitions of s, respectively.

Definition 1 For a join or a fork/join state s in a TEFSM, $t(s) \stackrel{\text{def}}{=} max_{1 \leq i \leq k}\{t_i|$ where t_i is the time when the incoming event e_i of s has occurred, $1 \leq i \leq k\}$. For all other states s in a TEFSM, $t(s)$ is defined to be the time when an incoming event e of s has occurred provided the machine has executed e to reach the state s.

Definition 2 [10] A *timed sequence* S in a TEFSM M is a finite or infinite sequence of pairs $< e_i, t_i >$, where $t_i < t_{i+1}$ if $host(e_i) = host(e_{i+1})$ and $t_i \leq t_{i+1}$ otherwise and each pair $< e_i, t_i >$ denotes that an event e_i of M has occurred when the time is equal to t_i.

Definition 3 A timed sequence S in a TEFSM M is *valid* if each e_i has been executable at $head(e_i)$, i.e., $P(e_i)$ was true at $head(e_i)$ and has remained to be true till the execution of e_i, and $t(head(e_i)) + min_{e_i} \leq t_i \leq t(head(e_i)) + max_{e_i}$.

Definition 4 Let $\{seq_1, \ldots, seq_n\}$ be a set of sequences, where seq_i is a valid sequence in a TEFSM $PS_i, 1 \leq i \leq n$. A *merged sequence* $seq(\sum_{i=1}^{n} i)$ from the set is a sequence of pairs $< e_i, t_i >$, where e_i is in the union of the events of $PS_j, 1 \leq j \leq n$, such that $seq(\sum_{i=1}^{n} i) \downarrow_j = seq(j)$, for each $j, 1 \leq j \leq n$, and $t_i \leq t_{i+1}$.

Notation 2 (1) $\{PS_i, 1 \leq i \leq n\}$ denotes the set of the merged sequences $\{seq(\sum_{i=1}^{n} i) | seq(\sum_{i=1}^{n} i)$ is a merged sequence from $\{seq_1, \ldots, seq_n\}$, where seq_i is a valid sequence in a TEFSM $PS_i, 1 \leq i \leq n\}$. (2) $\{SS\}$ denotes the set of valid sequences in a service specification SS.

The protocol synthesis problem is basically to derive a protocol specification for the protocol entities from a given service specification such that each protocol entity would be able to execute events in exactly the same order as specified in the service specification. However, since the specification is modeled by a TEFSM, the problem now is to consider time constraints as well as the relative order of the events in the service specification. Along with the time constraints associated with events, the variable nature of the communication delays make it impossible to derive a protocol specification which would be able to fully simulate the service specification. Therefore, to cope with the discrepancy between protocol and service specifications, we define the protocol synthesis problem as follows. Derive a protocol specification from a given service specification which satisfies the following conditions.

Definition 5 A derived protocol specification $PS_i, 1 \leq i \leq n$, is correct with respect to the service specification SS if (1) every merged sequence $seq(\sum_{i=1}^{n} i)$ from $\{seq(1), \ldots, seq(n)\}$, where $seq(i)$ is a valid sequence in PS_i, $1 \leq i \leq n$, is a valid sequence in $\{SS\}$; and (2) every valid sequence σ in $\{SS\}$ is a merged sequence from $\{SS \downarrow_1, \ldots, SS \downarrow_n\}$, where $SS \downarrow_i$ preserves the order of events as specified in $PS_i, 1 \leq i \leq n$.

Condition (2) of Definition 5 means that the derived protocol specification should preserve the order of events, but not necessarily simulate the same time stamp of the events in the service specification.

4 Synthesis Algorithm

We present an algorithm that derives the maximal protocol specification among the correct protocol specifications from a service specification. Moreover, we also give an algorithm for finding the maximal subset of a service specification which can be represented by the derived protocol specification.

Since we assume that each protocol entity is modeled by a TEFSM, no (fork, join) pair in a service specification should contain a set of control flows that might be able to cause a conflict, i.e., two or more concurrent events with the same host(protocol entity) and the same time stamp. To cope with the problem, we provide a sufficient condition for a service specification to be conflict-free. We believe that the condition given in Lemma 1 does not severely restrict the modeling power of TEFSM.

Lemma 1 A TEFSM M with nonempty FJ is conflict-free if for each $(f, j) \in FJ$, any two sequences s_1 and s_2 from f to j that can be executed concurrently by M do not share a host, i.e., $host(s_1) \cap host(s_2) = \emptyset$, where $host(s_i) = \{m | a^m \text{ is an event in } s_i\}$.

We also impose a restriction R_1 to the service specification SS as follows: for every choice state s in SS, $|host(OUT(s))| = 1$. R_1 means that when a choice is possible during the execution of a concurrent protocol system, the choice should be made locally by the same protocol entity to avoid possible deadlocks.

For the sake of algorithm presentation, we denote (x, e^i, y) as the event e^i with the head state x and the tail state y, respectively. The following algorithm generates PS_i, the specification for protocol entity i, and we can get the protocol specification $PS_i, 1 \leq i \leq n$, by running the algorithm n times with different i each time.

Synthesis

- Input: Service specification SS with the condition in Lemma 1 and R_1 represented by a TEFSM and $D_{i,j}, \forall i, j, 1 \leq i, j \leq n$. Note that $D_{i,i} = 0, \forall i, 1 \leq i \leq n$.
- Output: Protocol entity specification PS_i in a TEFSM

1. For each state s with $|IN(s)| > 0$ in SS do the following:
   Let $IN(s) = \{(u_1, e^{in_1}, s), \ldots, (u_k, e^{in_k}, s)\}$ and $OUT(s) = \{(s, f^{out_1}, v_1), \ldots, (s, f^{out_l}, v_l)\}$.
   (a) (Append send and/or receive transitions appropriately.)
      i. s is a choice state: Note that $out_1 = \ldots = out_l \stackrel{\text{let}}{=} j$.
         for each transition $(u_x, e^{in_x}, s), 1 \leq x \leq k$, do:
         - if $(in_x \neq i \wedge j = i)$, then append a receive transition to the transition as in Figure 2(a);
         - else if $(in_x = i \wedge j \neq i)$, then append a send transition to the transition as in Figure 2(b);
         - else if $(in_x = i \wedge j = i) \vee (in_x \neq i \wedge j \neq i)$, then do nothing;
      ii. s is a fork, but not a join state:
         for each transition $(u_x, e^{in_x}, s), 1 \leq x \leq k$, do:
         - if $(in_x = i)$, then append a set of send transitions to the transition as in Figure 3(a);

Figure 2: Case(i) s is a choice state

- else if $(in_x \neq i \wedge i \in \{out_1, \ldots, out_l\})$, then append a receive transition to the transition as in Figure 3(b);
- else if $(in_x \neq i \wedge i \notin \{out_1, \ldots, out_l\})$, then do nothing;

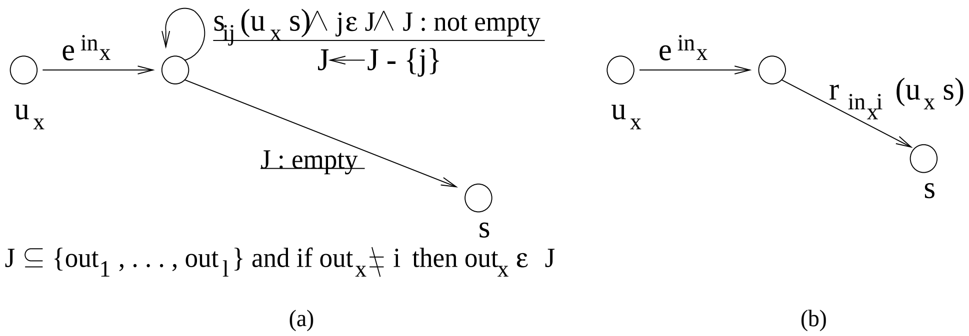


Figure 3: Case(ii) s is a fork, but not a join state

iii. s is a join or a fork/join state:
- if $(i \in \{in_1, \ldots, in_k\}) \wedge (i \in \{out_1, \ldots, out_l\})$, then append a set of send and receive transitions to the transition as in Figure 4(a);
- else if $(i \in \{in_1, \ldots, in_k\}) \wedge (i \notin \{out_1, \ldots, out_l\})$, then append a set of send transitions to the transition as in Figure 4(b);
- else if $(i \notin \{in_1, \ldots, in_k\}) \wedge (i \in \{out_1, \ldots, out_l\})$, then append a set of receive transitions to the transition as in Figure 4(c);
- else if $(i \notin \{in_1, \ldots, in_k\}) \wedge (i \notin \{out_1, \ldots, out_l\})$, then do nothing;

(b) (Adjust the time intervals associated with the outgoing transitions of s, if necessary.) if $i \in \{out_1, \ldots, out_l\}$, then [for each transition $t^y \stackrel{\text{let}}{=} (s, f^{out_y}, v_y), 1 \leq y \leq l$, such that $out_y = i$, do: $[min_{t^y}, max_{t^y}] \leftarrow [min_{t^y}, max_{t^y} - max_{1 \leq x \leq k}\{D_{in_x,i}\}]$[1]]

2. (Project the SS from the Step 1 onto PS_i.) For every event $e^z, z \neq i$, replace the event with an ϵ transition.

3. For each pair $(s_f, s_j) \in FJ$, remove all ϵ paths from s_f to s_j, if any. If all the transitions and states except s_f and s_j are removed, then merge s_f and s_j into a single state $s_{f,j}$.

4. Remove ϵ transitions by the standard algorithm given in [6].

[1]If $min_{t^y} > max_{t^y} - max_{1 \leq x \leq k}\{D_{in_x,i}\}$, then the execution of f^{out_y} may not be possible under the delay constraint.

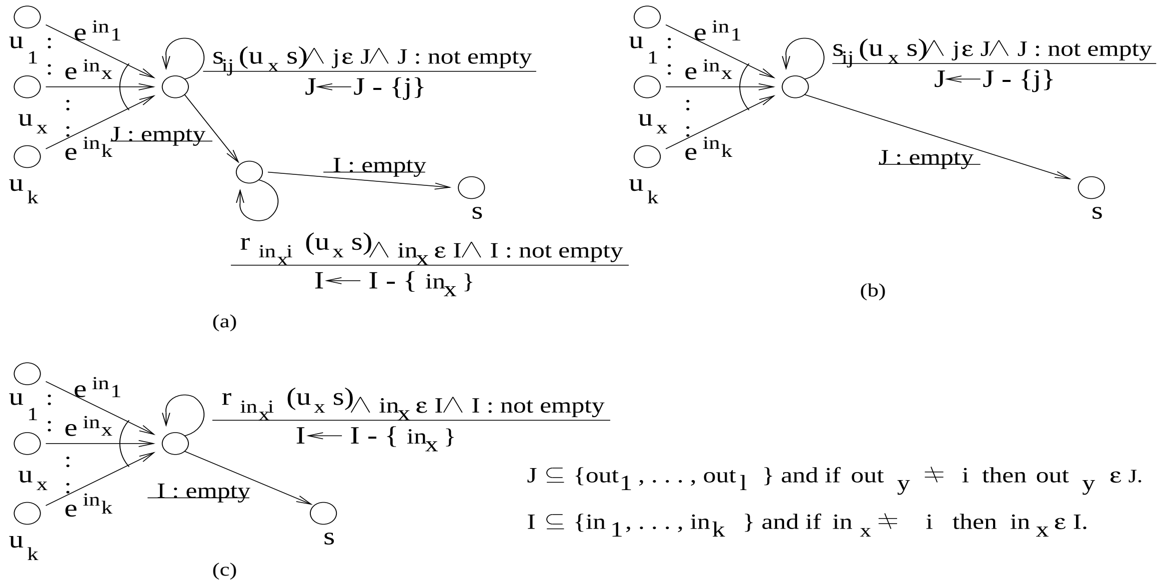


Figure 4: Case(iii) s is a join or a fork/join state

Lemma 2 Let $PS_i, 1 \leq i \leq n$, be the derived protocol specification from SS under the delay constraints $D_{i,j}, 1 \leq i, j \leq n$. Then $\{PS_i, 1 \leq i \leq n\} \subseteq \{SS\}$. Moreover, $\{PS_i, 1 \leq i \leq n\}$ is maximal in the sense that any extension of a time interval in any PS_i might be able to generate sequences which are not in $\{SS\}$ under some specific delay constraints.

Proof: We first show that $\{PS_i, 1 \leq i \leq n\} \subseteq \{SS\}$. Let $\{seq(i), 1 \leq i \leq n\}$ be a set of sequences such that $seq(i)$ is a valid sequence in PS_i for each $i, 1 \leq i \leq n$. The proof is by induction on $|\sigma|$, where σ is a merged sequence from $\{seq(i), 1 \leq i \leq n\}$. *Base Case* $|\sigma| = 1$. It is clear that σ is a valid sequence in SS. *Induction Hypothesis*(IH for short) Assume the claim holds for $|\sigma| = k > 0$. Let $\sigma' = \sigma \cdot < a^i, t >, |\sigma| = k$, be a merged sequence from $\{seq(i), 1 \leq i \leq n\}$. Let $s \overset{\text{let}}{=} head(a^i)$. Suppose s is a choice state. By the Step 1(a) of the algorithm Synthesis, PS_i can execute a^i either after having received a message from one of $PS_{in_x}, in_x \neq i$ or after having executed $e^{in_x}, in_x = i$. In either case, we know from the algorithm Synthesis that an event in $IN(s)$ must have occurred in σ. Let e^{in_x} be the latest event from $IN(s)$ in σ. Then, the subsequence $e^{in_x}, \ldots, last(\sigma)$ of σ does not have any event from $OUT(s)$ since otherwise a^i would not have occurred in σ' by the nature of choice state. Thus, we showed that a^i had been executable at $last(\sigma)$. Suppose s is a fork, but not a join state. The only difference here from the above case(s:a choice state) is that the subsequence $e^{in_x}, \ldots, last(\sigma)$ of σ might have some events from $OUT(s)$, but no a^i's since otherwise a^i would not have occurred at $last(\sigma)$. Suppose s is a join, but not a fork state. By the construction of PS_i in Step 1(a) of the algorithm Synthesis, we know that PS_i can execute a^i only after having received a set of messages from $\{PS_{in_x} |$ where $(u_x, e^{in_x}, s) \in IN(s), in_x \neq i\}$, which implies that $\{e^{in_x}, in_x \neq i\}$ had occurred in σ by PS_{in_x}, respectively. Also, we know that e^{in_y}, where $(u_y, e^{in_y}, s) \in IN(s)$, and $in_y = i$, if any, had occurred in σ. Let the most recently occurred event from $OUT(s)$ in σ be e^{in_x}, i.e., $t(s)$ is equal to the time when e^{in_x} has occurred. Then the subsequence $e^{in_x}, \ldots, last(\sigma)$ of σ does not have any event from $OUT(s)$ since otherwise a^i would not have occurred. Thus, a^i had been executable at $last(\sigma)$. Suppose

s is a join/fork state. As above, we know that $\{e^{in_x}, 1 \leq x \leq k\}$ had occurred in σ. We also know that the subsequence $e^{in_x}, \ldots, last(\sigma)$ might have some events from $OUT(s)$, but no a^i's since otherwise a^i would not have occurred at $last(\sigma)$, where e^{in_x} is the most recently occurred event from $OUT(s)$ in σ. Now, it is straightforward that a^i had been executable at $last(\sigma)$. Thus, we conclude that a^i had been executable at $last(\sigma)$ for all cases. Next we show that $t(s) + min_{a^i} \leq t \leq t(s) + max_{a^i}$, where $[min_{a^i}, max_{a^i}]$ is the time interval associated with a^i in SS. The time interval associated with a^i in PS_i, by the algorithm Synthesis Step 1(b), becomes $[min_{a^i}, max_{a^i} - max_{1 \leq x \leq k}\{D_{in_x,i}\}]$. Since $\sigma' \downarrow_i = \sigma \downarrow_i \cdot < a^i, t >$ is a valid sequence in PS_i, we know that $t(s) + min_{a^i} + d_{in_x,i} \leq t \leq t(s) + d_{in_x,i} + max_{a^i} - max_{1 \leq x \leq k}\{D_{in_x,i}\}$, where $0 < d_{in_x,i} \leq D_{in_x,i}$. Thus we have that $t(s) + min_{a^i} < t(s) + min_{a^i} + d_{in_x,i} \leq t \leq t(s) + max_{a^i} - \{max_{1 \leq x \leq k}\{D_{in_x,i}\} - d_{in_x,i}\} \leq t(s) + max_{a^i}$. Therefore, since σ is a valid sequence in SS by IH, σ' is also a valid sequence in SS from the above argument. To prove the maximality of $\{PS_i, 1 \leq i \leq n\}$, consider a sequence ψ in $\{SS\} - \{PS_i, 1 \leq i \leq n\}$. It is clear that $|\psi| > 1$, since any sequence in $\{SS\}$ with length 1 should also be in $\{PS_i\}$, for some i. We know that there exists a pair of events $< e^i, t_e >, < f^j, t_f >$ in ψ such that $e^i \in IN(s), f^j \in OUT(s), i \neq j$, and $t(s) = t_e$ for some state s in SS, since otherwise ψ would not be in $\{SS\} - \{PS_i, 1 \leq i \leq n\}$. Note that e^i and f^j might not be adjacent in ψ. By the algorithm Synthesis, the time interval associated with the event f^j is adjusted into $[min_{f^j}, max_{f^j} - max_{1 \leq x \leq k}\{D_{in_x,j}\}]$ in PS_j, where $e^{in_x} \in IN(s), 1 \leq x \leq k$. Assume the interval associated with the event f^j in PS_j is extended to $[min_{f^j} - \epsilon_1, max_{f^j} - max_{1 \leq x \leq k}\{D_{in_x,j}\} + \epsilon_2]$, where ϵ_1 and ϵ_2 are positive constants. Then $t_e + min_{f^j} - \epsilon_1 + \eta < t_e + min_{f^j}$ for a positive constant η such that $\eta < \epsilon_1 \leq D_{i,j}$. Hence, if the actual delay from PS_i to PS_j is η, then $< e^i, t_e >, < f^j, t_e + min_{f^j} - \epsilon_1 + \eta >$ would not be possible in any sequence in $\{SS\}$, since $t_e + min_{f^j} - \epsilon_1 + \eta < t_e + min_{f^j}$. Similarly, $t_e + max_{f^j} - max_{1 \leq x \leq k}\{D_{in_x,j}\} + \epsilon_2 + max_{1 \leq x \leq k}\{D_{in_x,j}\} = t_e + max_{f^j} + \epsilon_2 > t_e + max_{f^j}$. Thus, $< e^i, t_e >, < f^j, t_e + max_{f^j} - max_{1 \leq x \leq k}\{D_{in_x,j}\} + \epsilon_2 + max_{1 \leq x \leq k}\{D_{in_x,j}\} >$ would not be possible in any sequence in $\{SS\}$, if $max_{1 \leq x \leq k}\{D_{in_x,j}\} = D_{i,j}$ and the actual delay from PS_i to PS_j is $D_{i,j}$. Note that if $max_{1 \leq x \leq k}\{D_{in_x,j}\} > D_{i,j}$, the validity of the pair $< e^i, t_e >, < f^j, t_e + max_{f^j} - max_{1 \leq x \leq k}\{D_{in_x,j}\} + \epsilon_2 + D_{i,j} >$ in SS depends upon the sign of the value $D_{i,j} + \epsilon_2 - max_{1 \leq x \leq k}\{D_{in_x,j}\}$. ■

On the other hand, it should be clear that $\{SS\} \not\subseteq \{PS_i, 1 \leq i \leq n\}$ because of the adjustment in Step 1(b) of the algorithm Synthesis. However, we can restrict SS to get a sub specification SS^* such that $\{SS^*\} \subseteq \{PS_i, 1 \leq i \leq n\}$. Here, we give an algorithm to generate such a sub specification SS^* which is maximal in the sense that $\{SS'\} \subseteq \{PS_i, 1 \leq i \leq n\}$ implies $\{SS'\} \subseteq \{SS^*\}$.

Restriction

- Input: Service specification SS with the condition in Lemma 1 and R_1 represented by a TEFSM and $D_{i,j}, \forall i, j, 1 \leq i, j \leq n$. Note that $D_{i,i} = 0, \forall i, 1 \leq i \leq n$.
- Output: Restricted service specification SS^* in a TEFSM

For each state s with $|IN(s)| > 0$ in SS do the following:

1. Let $IN(s) = \{(u_1, e^{in_1}, s), \ldots, (u_k, e^{in_k}, s)\}$ and $OUT(s) = \{(s, f^{out_1}, v_1), \ldots, (s, f^{out_l}, v_l)\}$.
2. For each $i, 1 \leq i \leq n$, where n is the number of the protocol entities, do the following:

- if $i \in \{out_1, \ldots, out_l\}$, then [for each transition $t^y \stackrel{\text{let}}{=} (s, f^{out_y}, v_y), 1 \leq y \leq l$, such that $out_y = i$, do: $[min_{t^y}, max_{t^y}] \leftarrow \bigcap_{0<d\leq max_{1\leq x\leq k}\{D_{in_x,i}\}} \{[min_{t^y} + d, max_{t^y} - max_{1\leq x\leq k}\{D_{in_x,i}\} + d]\}$.[2]

Lemma 3 Let $PS_i, 1 \leq i \leq n$, be the derived protocol specification from SS and SS^* be the restricted service specification of SS. Then every valid sequence σ in $\{SS\}$ is a merged sequence from $\{SS \downarrow_1, \ldots, SS \downarrow_n\}$, where $SS \downarrow_i$ preserves the order of events as specified in $PS_i, 1 \leq i \leq n$. Moreover, $\{SS^*\} \subseteq \{PS_i, 1 \leq i \leq n\}$, and $\{SS^*\}$ is maximal in the sense that any extension of a time interval in SS^* might be able to generate sequences which are not in $\{PS_i, 1 \leq i \leq n\}$ under some specific delay constraints.

Proof: Since SS and SS^* are equivalent if timing is ignored, we know that it suffices to show that $\{SS^*\} \subseteq \{PS_i, 1 \leq i \leq n\}$ to guarantee that the order of events specified in SS is preserved in each $PS_i, 1 \leq i \leq n$. We first show that $\{SS^*\} \subseteq \{PS_i, 1 \leq i \leq n\}$. The proof is by induction on $|\sigma|$, where σ is a valid sequence in SS^*. *Base Case* $|\sigma| = 1$. It is easy to see that σ is a merged sequence from $\{PS_i\}$, where $\sigma =< a^i, t >$. *Induction Hypothesis*(IH for short) Assume the claim holds for $|\sigma| = k > 0$. Let $\sigma' = \sigma \cdot < a^i, t >, |\sigma| = k$, be a valid sequence in SS^*. By IH, we know that for each $j, 1 \leq j \leq n$, $\sigma \downarrow_j$ is a valid sequence in PS_i. We show that $\sigma \downarrow_i \cdot < a^i, t >$ is also a valid sequence in PS_i, where we assume for the sake of the proof that PS_i is the protocol specification obtained in Step 3, i.e., one with ϵ transitions. Note that $\sigma \downarrow_j = \sigma' \downarrow_j$, for $j \neq i, 1 \leq j \leq n$. Let $s \stackrel{\text{let}}{=} head(a^i)$. Since σ' is a valid sequence in SS^*, it is clear that a^i had been executable at $last(\sigma)$. We let $last(\sigma \downarrow_i) = first(\sigma)$, if $\sigma \downarrow_i = \epsilon$. By Step 2 of the algorithm Synthesis, the transitions after $last(\sigma \downarrow_i)$ through $last(\sigma)$, if any, in PS_i would be ϵ transitions. It is not hard to verify that, by investigating Step 1 and 2 of the algorithm Synthesis, the sequence σ would be able to lead PS_i into the state s and moreover s is reachable from either $head(last(\sigma \downarrow_i))$, if any, or the start state of PS_i, otherwise, via only ϵ, send, and/or receive transitions. Next we show that the inequalities $t(s) + min_{a^i}^{PS_i} + d_{in_j,i} \leq t \leq t(s) + max_{a^i}^{PS_i} + d_{in_j,i}$ hold regardless of the value of the actual delay $d_{in_j,i}$ as long as $0 < d_{in_j,i} \leq D_{in_j,i}$, where $[min_{a^i}^{PS_i}, max_{a^i}^{PS_i}]$ is the time interval associated with a^i in PS_i and $< e^{in_j}, t_e >$ is an incoming event of s such that $t(s) = t_e$. Note that $[min_{a^i}^{PS_i}, max_{a^i}^{PS_i}] = [min_{a^i}, max_{a^i} - max_{1\leq j\leq k}\{D_{in_j,i}\}]$. Since σ' is a valid sequence in SS^*, we have that $t(s) + min_{a^i}^{SS^*} \leq t \leq t(s) + max_{a^i}^{SS^*}$, where $[min_{a^i}^{SS^*}, max_{a^i}^{SS^*}] = \bigcap_{0<d\leq max_{1\leq j\leq k}\{D_{in_j,i}\}} \{[min_{a^i} + d, max_{a^i} - max_{1\leq j\leq k}\{D_{in_j,i}\} + d]\}$ is the time interval associated with a^i in SS^*. We have that $t(s) + min_{a^i}^{PS_i} + d_{in_j,i} = t(s) + min_{a^i} + d_{in_j,i} \leq t(s) + min_{a^i} + D_{in_j,i} \leq t(s) + min_{a^i} + max_{1\leq j\leq k}\{D_{in_j,i}\} \leq t(s) + min_{a^i}^{SS^*} \leq t$. Similarly, $t \leq t(s) + max_{a^i}^{SS^*} < t(s) + max_{a^i} - max_{1\leq j\leq k}\{D_{in_j,i}\} + \eta$, where the last inequality holds for any positive constant η. Hence, by choosing η sufficiently small, we have $t(s) + max_{a^i}^{PS_i} + \eta \leq t(s) + max_{a^i}^{PS_i} + d_{in_j,i}$, which completes the other half. Therefore, since $\sigma \downarrow_i$ is a valid sequence in PS_i (by IH), $\sigma' \downarrow_i = \sigma \downarrow_i \cdot < a^i, t >$ is also a valid sequence in PS_i. To prove the maximality of $\{SS^*\}$, consider a sequence ψ in $\{SS'\} - \{SS^*\}$, where SS' is SS^* with a time interval in SS^* extended. We know that $|\psi| > 1$, since any sequence in $\{SS^*\}$ with length 1 must have an unadjusted time interval, which implies that any extension of the interval would generate a sequence not in $\{SS\}$, a contradiction. We know that there exists a pair of events $< e^i, t_e >, < f^j, t_f >$ in ψ such that $e^i \in IN(s), f^j \in OUT(s), i \neq j$, and $t(s) = t_e$ for some state s in SS and the time interval

[2]If the intersection for any transition does not exist, SS^* does not, either.

associated with f^j is extended in SS', since otherwise ψ would not be in $\{SS'\} - \{SS^*\}$. Note that e^i and f^j might not be adjacent in ψ. By the algorithm Restriction, the time interval associated with the event f^j in SS^* is adjusted into $[l, u] \stackrel{\text{let}}{=} \bigcap_{0<d\leq max_{1\leq x\leq k}\{D_{in_x,j}\}}$ $\{[min_{f^j} + d, max_{f^j} - max_{1\leq x\leq k}\{D_{in_x,j}\} + d]\}$. Assume the extended interval associated with the event f^j in SS' is $[l - \epsilon_1, u]$ or $[l, u + \epsilon_2]$, where ϵ_1 and ϵ_e are positive constants. Then $l - \epsilon_1 = min_{f^j} + max_{1\leq x\leq k}\{D_{in_x,j}\} - \epsilon_1 < min_{f^j} + D_{i,j}$, if $max_{1\leq x\leq k}\{D_{in_x,j}\} = D_{i,j}$. Hence, if the actual delay from PS_i to PS_j is $D_{i,j}$ and $max_{1\leq x\leq k}\{D_{in_x,i}\} = D_{i,j}$, then $< e^i, t_e >, < f^j, t_e + l - \epsilon_1 >$ would not be possible in $\{PS_i, 1 \leq i \leq n\}$. Similarly, $u + \epsilon_2 > max_{f^j} - max_{1\leq x\leq k}\{D_{in_x,j}\} + \epsilon_2$. Thus, if the actual delay from PS_i to PS_j is less than ϵ_2, $< e^i, t_e >, < f^j, t_e + u + \epsilon_2 >$ would not be possible in $\{PS_i, 1 \leq i \leq n\}$, either. ■

By lemmas 2 and 3, we have the following theorem which proves the correctness of the algorithm Synthesis.

Theorem 1 A derived protocol specification $PS_i, 1 \leq i \leq n$, is correct with respect to the service specification SS.

5 An Example

To demonstrate the synthesis method, we show the protocol specification after each step of the algorithm Synthesis when the service specification SS in Figure 5 is given. Figure 6 (a),(b), and (c) describe the protocol specification PS_1 after each step of the algorithm Synthesis. After removing ϵ transitions, we have the final protocol specification PS_1, which is given in Figure 7 along with the final protocol specifications PS_2 and PS_3.

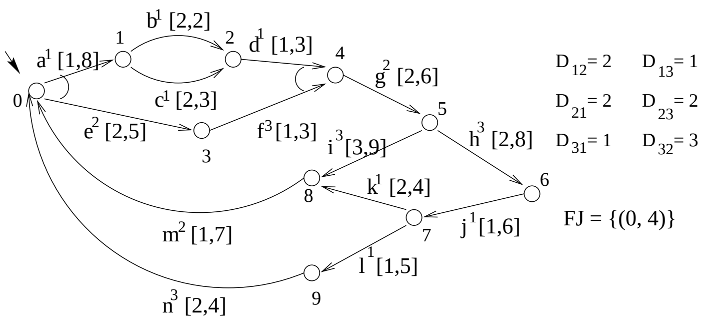


Figure 5: A Service Specification SS

6 Conclusion

We proposed a model based on EFSM that can represent concurrency, synchronization, and timing requirements explicitly, and presented a method to synthesize protocol specifications from timed service specifications based on the model. The proposed method appropriately inserts send and/or receive transitions between the events in the service specification so that the event orderings in the service specification are preserved. The time intervals associated with transitions are also adjusted by the method to incorporate the delay between protocol

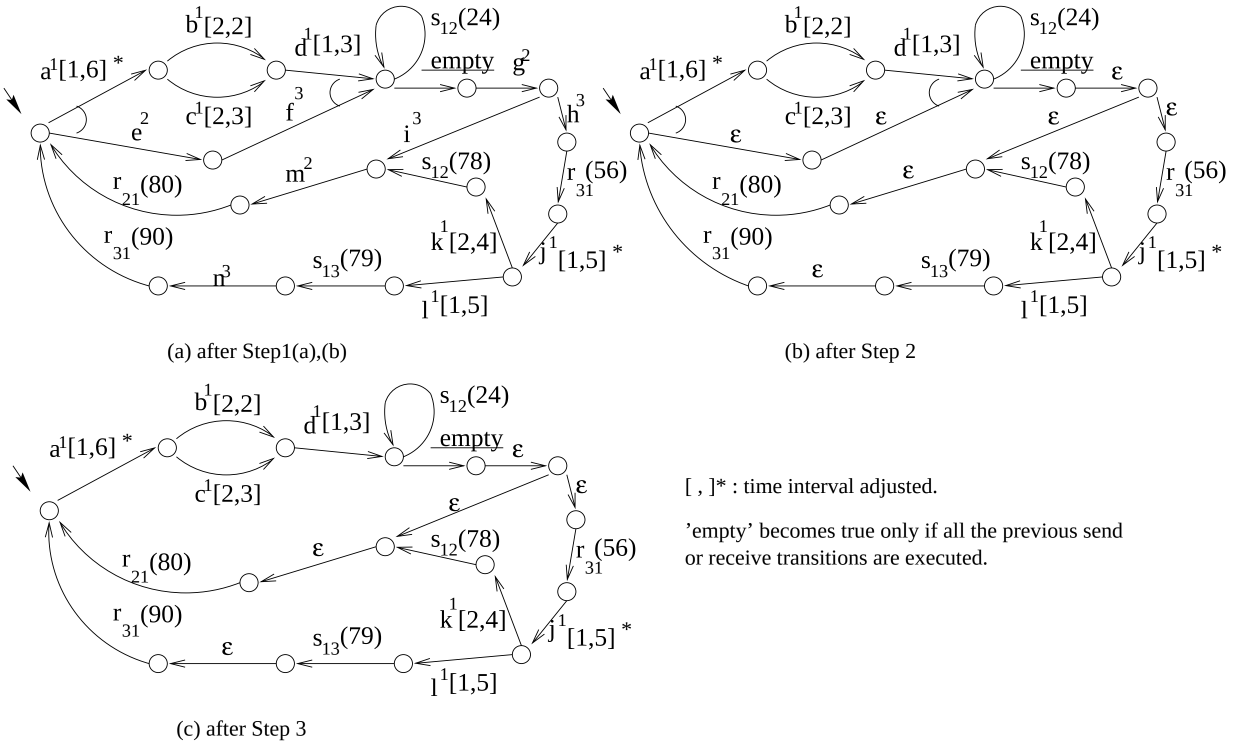


Figure 6: PS_1 after each step of the algorithm Synthesis

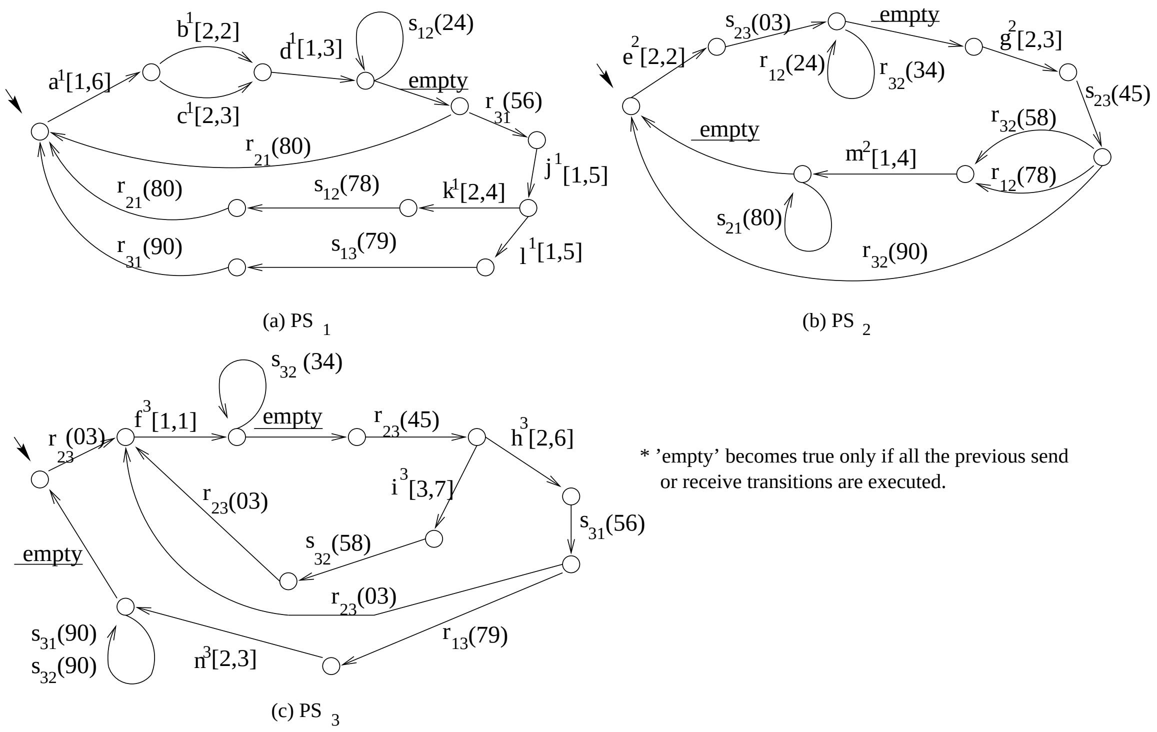


Figure 7: Protocol Specifications PS_1, PS_2, and PS_3

entities for synchronization. We proved that the derived protocol specification is optimal in the sense that any superset of the protocol specification would necessarily include specifications which are not attainable from the service specification under some specific delay constraints. We also presented a method to derive a sub specification from a service specification and a maximum communication delay of each channel such that the sub specification, but no superset of it, can be simulated by the derived protocol specification.

A formalization of logical errors in a TEFSM would be required to further investigate the relationship between the derived protocol specification and the service specification, as far as insuring the absence of design errors.

References

[1] R. L. Probert and K. Saleh *Synthesis of Communication Protocols: Survey and Assessment* IEEE Trans. Comput., vol. 40, no. 4, pp. 468-476, April 1991.

[2] P. M. Chu and M. T. Liu *Synthesizing Protocol Specification from Service Specification in the FSM model* Proc. Comput. Networking Symp., pp. 173-182, April 1988.

[3] P. M. Chu and M. T. Liu *Protocol Synthesis in a State-Transition Model* Proc. COMPSAC'88, pp. 505-512, October 1988.

[4] P. Zafiropulo, C. H. West, H. Rudin, D. D. Cowan, and D. Brand *Towards Analyzing and Synthesizing Protocols* IEEE Trans. Commun., vol. COM-28, no. 4, pp. 651-661, April 1980.

[5] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi *Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri net Model with Registers* Proc. IEEE Int'l Conf. Dist. Comp. Syst., pp. 510-517, May 1995.

[6] W. A. Barrett and J. D. Couch *Compiler Construction: Theory and Practice* Chapter 3, Science Research Associates, 1979.

[7] K. Saleh and R. L. Probert *Automatic Synthesis of Protocol Specifications from Service Specifications* Proc. IEEE Phoenix Conf. Comput. and Commun., pp. 615-621, March 1991.

[8] C. Kant, T. Higashino, and G. v. Bochmann *Deriving Protocol Specifications from Service Specifications Written in LOTOS* Distributed Computing, vol. 10, pp. 29-47, 1996.

[9] R. Langerak *Decomposition of Functionality: A Correctness-Preserving LOTOS Transformation* Proc. X IFIP Symp. Protocol Specification, Testing and Verification, pp. 229-242, June 1990.

[10] A. Khoumsi, G. v. Bochmann, and R. Dssouli *On Specifying Services and Synthesizing Protocols for Real-time Applications* Proc. XIV IFIP Symp. Protocol Specification, Testing and Verification, pp. 177-192, June 1994.

[11] A. Nakata, T. Higashino, and K. Taniguchi *Protocol Synthesis from Timed and Structured Specifications* Proc. Int'l Conf. Network Protocols, pp. 74-81, November 1995.

[12] C.-M. Huang and S.-W. Lee *Timed Protocol Verification for Estelle-Specified Protocols* ACM SIGCOMM Comput. Commun. Review, vol. 25, no. 3, pp. 5-32, July 1995.

[13] K. Naik and B. Sarikaya *Protocol Conformance Test Case Verification Using Timed-Transitions* Proc. XIV IFIP Symp. Protocol Specification, Testing and Verification, pp. 98-113, June 1994.