PAPER Special Section on Net Theory and Its Applications

Synthesis of Protocol Specifications from Service Specifications of Distributed Systems in a Marked Graph Model

Hirozumi YAMAGUCHI†, *Nonmember*, Kozo OKANO†, Teruo HIGASHINO† *and* Kenichi TANIGUCHI†, *Members*

SUMMARY In a distributed system, the protocol entities must exchange some data values and synchronization messages in order to ensure the temporal ordering of the events described in a service specification for the distributed system. It is desirable that a correct protocol specification can be derived automatically from a given service specification. In this paper, we propose an algorithm which synthesizes automatically a correct protocol specification from a service specification described as a Marked Graph with Registers (MGR) model and resources (registers and gates) allocation information. This model has a finite control modeled as a marked graph. Therefore, parallel events can be described. In our method, to minimize the number of the exchanged messages, we use a procedure to calculate an optimum solution for 0-1 integer linear programming problems. The number of the steps which each protocol entity needs to simulate one transition in the service specification is also minimized. Ways to avoiding conflict of registers are also described. Our approach has the following advantages. First, parallel events can be described in a service specification. Secondly, many practical systems can be described in the MGR model. Finally, at the protocol specification level, we can understand what events can be executed in parallel.

key words: *Petri net, marked graph, protocol synthesis, parallel events*

1. Introduction

If we regard a communication system as a black box, a description of the temporal ordering of input/output events observed at SAP's (Service Access Points) is regarded as a service specification of the communication system [2]. The services (input/output events) are provided by some cooperative protocol entities (or nodes) which exchange data and synchronization messages through communication channels. The behaviors of each protocol entity are described as a protocol entity specification, and the set of all protocol entities' specifications is called a protocol specification [2]. Some synthetic approaches have been proposed such as LOTOS based approaches [4]–[6], FSM based approaches [8]–[11], and so on (for survey, see Ref. [2]), in order to get a correct protocol specification satisfying a given service specification. Most of them do not handle data parameters. We have also proposed a synthetic technique for an extended FSM model with registers [13]. In this model, we can treat global variables such as state variables and registers. Each register can be allocated to more than one protocol entity of the distributed system while each gate must belong to one of the protocol entities. Such an allocation information is given by the designers.

However, parallel events can not be described in the model. Such a parallelism is very important for describing service specifications. In this paper, we introduce a Marked Graph model with a finite number of Registers (MGR model) in order to describe parallel events. And, we propose an algorithm to derive correct protocol specifications from a service specification in the MGR model and a resource allocation information. Each protocol entity specification is obtained from the service specification by replacing each transition with a suitable sub-marked graph that includes input/output events which exchange synchronization messages and data values among the protocol entities. In order to reduce the number of exchanged messages among the protocol entities under the simulating principle above, we use a similar method in Ref. [20]. In this method, a procedure to calculate optimum solutions for 0-1 integer linear programming problems is used. We also give how to construct the MGR using parallel events for reducing the number of steps which each protocol entity needs to simulate one transition of a service specification under the simulating principle. We also give the ways to avoiding conflict of registers that arises in parallel computing model. The construction methods and the ways for avoiding conflict of registers are not dealt with in Ref. [20].

Our approach has the following advantages. First, parallel events can be described in a service specification. Secondly, since the MGR model has registers and many functions to calculate the values of the registers are allowed, many practical systems can be described in the MGR model. Thirdly, we give the ways to avoiding conflict of registers. Finally, at the protocol specification level, we can understand what events can be executed in parallel, because the algorithm derives protocol specifications that have the most parallel transitions.

The paper is constructed as follows. In Sect. 2, we define our MGR model. In Sect. 3, the equivalence be-

Manuscript received April 12, 1994.
Manuscript revised June 10, 1994.

†The authors are with the Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan.

tween service specifications and protocol specifications is formally defined. An example of correct protocol specifications is also given. An algorithm to derive the protocol specifications, especially how to construct the MGR is described in Sect. 4. In Sect. 5, we explain techniques for avoiding the conflict of registers. In this section some extension model of MGR is used. In Sect. 6, an overview of a correctness proof of our algorithm is described.

2. MGR Model

2.1 Definition of Marked Graph Model with Registers

Marked graphs are known as a sub-class of the Petri nets for their well properties [1].

In this paper, a Marked Graph model with Registers (MGR model) is introduced. This model has a finite control modeled as a marked graph and a finite number of registers.

A MGR model M is defined as 10-tuple $M = (P, T, F, W, \mathcal{A}, G, R, C, \delta, M_I)$. The first four tuples correspond to a set of places, a set of transitions, a set of arcs and a function representing the weight of each arc, in the Petri Net model, respectively. For the marked graph, each place has exactly one incoming arc and exactly one outgoing arc with a unit weight. Thus the value of the function W of each arc is always 1.

The set $\mathcal{A}$ is a set of *events*, and the set G is a set of *guards*. The set R is a set of finite number of registers. The set C is a set of *register definition statements*. The function $\delta : T \rightarrow G \times \mathcal{A} \times C^*$ defines the content of each transition. The tuple $M_I = (M_0, INIT)$ gives an initial marking M_0 of the MGR M and initial values $INIT$ of the registers in R.

An event in the set $\mathcal{A}$ must have one of the following three forms: $a?x$, $a!E(\ldots)$ and i. The $a?x$ denotes an input event and the variable x represents an input value given from the gate a. More than one input values may be given for an input event. Such an input event is described like as "$a?x, y, z$". The event $a!E(r_{k_1}, \ldots, r_{k_t})$ denotes an output event and the value of the expression E is emitted from the gate a (more than one outputs may be emitted). The event i means an internal event which does not input nor output data from/to any gate.

Each element in the set G of guards is a predicate of input variables which used in an input event.

Each element in C^* is an n-tuple of register definition statements whose forms are $r_i \leftarrow f_1(r_{k_1}, \ldots, r_{k_t}, x)$ $, \cdots r_j \leftarrow f_n(r_{k_1}, \ldots, r_{k_{t'}}, x)$. For each transition t in T, t corresponds to an event a in $\mathcal{A}$, a guard g in G, and some register definition statements α in C^* where the event a is executable if the value of the guard g is true and the transition t can fire. If the event a is executed, then the new values of the registers are calculated based on the register definition statements α.

Here, we only treat safe and live marked graphs.

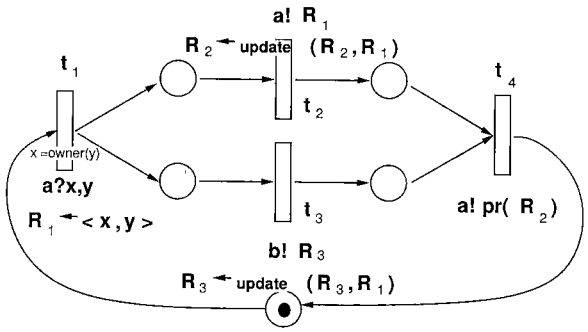


Fig. 1 An example of service specification SS in MGR model.

The safeness means that the number of the tokens for each place is at most one. The liveness allows for each transition to fire eventually from any reachable marking. This assures that there are no deadlocks.

We define the enable transition t as follows. A transition t is enable *iff* each input place of t has a token and the value of the guard of t is true. If an enable transition t fires, then the input (output) event of the t is executed and the values of the registers are modified based on the register definition statements of the t. For example, if a transition t has the following register definition statements $r_1 \leftarrow r_1 + r_2$, $r_2 \leftarrow r_1 - r_2$ and the current values of registers r_1 and r_2 are 4 and 1, respectively, then the values of registers r_1 and r_2 just after t fired, are 5 and 3, respectively.

Figure 1 is an example of a service specification SS which is described in MGR model. This is an example of a service specification describing a simple database machine with a sub-database for back up. In a given marking, if a guard of a transition t_1 is true, then the transition t_1 is enable to fire. In the example, the guard $x = \text{owner}(y)$ means that user-name x is an owner of the data y. If t_1 fires, the values of the input data x and y are input from the gate a (I/O terminal) and those values are stored into the register R_1 which is used for keeping the input data. Next, the value of the register R_1 is output from the gate a, and the database R_2 is updated by executing the transition t_2. On the other hand, on firing the transition t_3, the value of the sub-database R_3 is output from the gate b (display) and the sub-database R_3 is updated. After firing of the transitions t_2 and t_3 in parallel, a part of the contents of the database R_2 is emitted from the gate a (transition t_4). Guards of the transitions t_2, t_3 and t_4 are all "true" and they are omitted.

3. Protocol Specifications and Their Correctness

Here, we will implement a given service specification SS in a distributed system with p protocol entities (Fig. 2). We denote protocol specifications with p nodes by $\langle PE_1, \ldots, PE_p \rangle$ (or, simply PE^{1-p}), and a protocol entity specification of node k as PE_k.

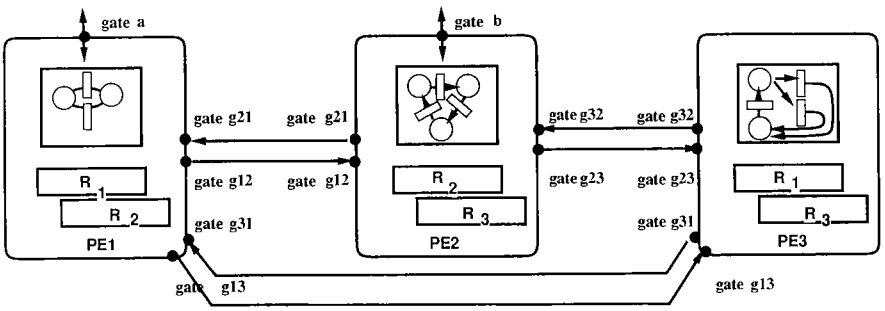


Fig. 2 Allocation of registers and gates to PE's.

In this paper, each protocol entity PE_k is described as the MGR model. We assume that each gate must belong to one of p protocol entities, and that each register may be allocated to more than one protocol entities. Figure 2 is an example of resource allocations. Let Alloc(SS) denote such a resource allocation.

Here, we assume that each communication channel from PE_i to PE_j is modeled as a FIFO queue (queue$_{ij}$) whose capacity is infinite, and that we call both side of the channel the gate g_{ij}. Therefore if the protocol entity PE_i executes "$g_{ij}!d$", then the data d is enqueued to the queue$_{ij}$. If the protocol entity PE_j executes "$g_{ij}?x$" and the first element of the queue$_{ij}$ is d, then the data d is dequeued from the queue$_{ij}$ and the value of d is assigned to the input variable x. If there are no elements in the queue$_{ij}$, then we assume that PE_j cannot execute the event $g_{ij}?x$.

3.1 Equivalence

For a protocol specification PE^{1-p}, we define the initial state of PE^{1-p} as follows. If each PE_k $(1 \leqq k \leqq p)$ is at its initial marking and its registers' values are initial values and each FIFO queue (communication channel) is empty, then such a situation is called the initial state of the PE^{1-p}.

We define the equivalence between a service specification SS and a protocol specification PE^{1-p} as follows.

Definition 1: [**Equivalence**] Suppose that all sending/receiving events $g_{ij}?x, g_{ij}!E(\ldots)$ for the communication channels and internal event i are unobservable. If a service specification SS and a protocol specification PE^{1-p} are observational congruent [10], [12], then we say that the SS and the PE^{1-p} are equivalent. The PE^{1-p} is called a correct protocol specification for the service specification SS. □

If SS and PE^{1-p} are observational congruent, then, all observational event sequences that SS (PE^{1-p}) can execute must be also executable in PE^{1-p} (SS). Also at the state after executing each observational event sequences, executable observational events for SS and PE^{1-p} must be the same.

3.2 Example of Protocol Entities Specifications

For example, suppose that the following resource allocation Alloc(SS) is given for the service specification SS in Fig. 1.

	PE_1	PE_2	PE_3
registers	r_1, r_2	r_2, r_3	r_1, r_3
gates	a	b	

For the protocol entities specifications PE_1, PE_2 and PE_3 in Fig. 3, the service specification SS in Fig. 1 and the protocol specification $\langle PE_1, PE_2, PE_3 \rangle$ are equivalent.

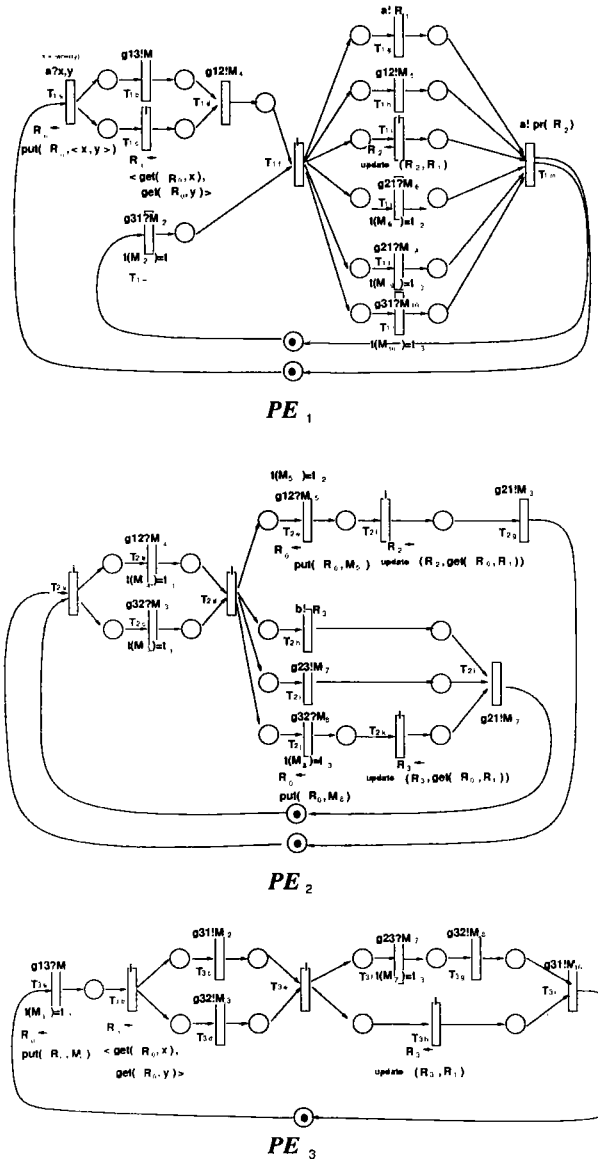
Note that each transition j of each PE_k has an auxiliary label T_{kj} for explanations in Fig. 3.

First, the PE_1 executes an input event at the transition T_{1a} and get values of the variables x and y from the gate a. These values must be assigned to the register R_1 based on the definition of the SS. Since the register R_1 belongs to PE_1 and PE_3, the PE_1 sends the values as a message M_1 to the PE_3 and changes the value of the register R_1 in the PE_1. On the other hand, at the transition T_{3a}, the PE_3 receives the message M_1 which has label corresponding to the transition t_1 in SS. And PE_3 changes the value of its register R_1 at the transition T_{3b}. At the transitions T_{3c} and T_{3d}, PE_3 sends messages to PE_1 and PE_2 in order to notify that PE_3 has finished to change the value of its register R_1. PE_1 sends a message M_4 to PE_2 at the transition T_{1d}. And PE_1 receives the message from PE_3 at the transition T_{1e}. Similarly, PE_2 receives the messages from PE_1 and PE_3 at the transitions T_{2b} and T_{2c}, respectively. Thus, the transition t_1 in SS is simulated by PE_1, PE_2 and PE_3.

3.3 Derivation Problem

Definition 2: [**Derivation Problem**] For a given SS, a set of nodes { node1, ..., node$p$ } and a resource allocation Alloc(SS), we consider the problem to derive a correct protocol specification PE^{1-p} □

Definition 3: [**Conflict of Registers**] Suppose that two transitions t_i and t_j $(t_i \neq t_j)$ are enable to fire at a reachable marking of a given service specification SS. If the transition t_i (or t_j) has a register definition



$M_1 : \mu(x,y)$, $M_2 : \gamma$, $M_3 : \gamma$, $M_4 : \gamma$, $M_5 : \mu(R_1)$,
$M_6 : \gamma$ $M_7 : \mu$, $M_8 : \beta(R_1)$, $M_9 : \gamma$, $M_{10} : \gamma$

Fig. 3 PE_1, PE_2, PE_3.

statement that changes the value of a register r and the transition t_j (or t_i) has a register definition statement or an ouput event that uses the register r, then we say that there is a conflict of the registers for the pair of the transitions t_i and t_j. □

If there is a conflict of the registers for two transitions t_i and t_j, the firing of t_i and t_j may make some conflict. Figure 4 shows an example of the conflict of the registers. Now, suppose that the values of the registers R_1 and R_2 are both 0. If the transition t_a fires before firing of the transition t_b, then the values of the registers R_1 and R_2 are both 1. And if t_b fires before t_a, then the values of the registers R_1 and R_2 are both 2. In a service specification level, there is no conflict. However, in a protocol specification level, wrong situations may occur if some mechanism for avoiding the conflict of the registers is not applied. Suppose that two different nodes p_1 and p_2 have the registers R_1 and R_2. Since the order of the reception of the messages may not be the same for the two nodes, one node p_1 may fire t_a and then t_b while the other node p_2 fires t_b and then t_a. Then, the values of the registers in the node p_1 become 1 while those in the node p_2 become 2. This makes a conflict. A mechanism for avoiding this conflict is described in Sect. 5.

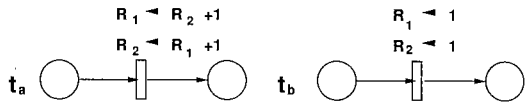


Fig. 4 Example of conflict of registers.

Here, for simplicity of discussion, we will give the following four restrictions on SS and Alloc(SS).

1. In a service specification SS, the internal event i is not used as the events.
2. There are no conflict of the registers.
3. Each gate must belong to one of p nodes.
4. If an output event $a!E(r_{h_1}, \ldots, r_{h_k})$ is described in a service specification SS, then each register r_{h_j} $(1 \leq j \leq k)$ must belong to the node which the gate a belongs. □

A specification satisfying the restriction 2 is called a conflict-free service specification. In a marked graph model, it is decidable whether given two transitions can fire simultaneously. Therefore, for a given specification SS, it is also decidable whether SS is a conflict-free or not.

If the restriction 3 does not hold, there may exist many nodes that can execute the same event simultaneously. If the restriction 4 does not hold, the node executing the event must get the values of the registers that it needs before it fires the transition.

3.4 Some Notations

Hereafter, we use the following some notations. Note that F denotes the set of arcs of the MGR model representing a given service specification SS. Each arc is represented as (t,p) or (p,t). The former denotes an arc from the transition t to the place p. The latter denotes its reverse. For each transition t, we define the set of output places for t as $t\bullet \equiv \{p \mid (t,p) \in F\}$. Similarly, for each place p, we define the set of output transitions for p as $p\bullet \equiv \{t \mid (p,t) \in F\}$. Also, for each transition t, we define the set of next transitions as $t\bullet\bullet \equiv \{s \mid s \in p\bullet \wedge p \in t\bullet\}$.

Since the restriction 3 holds, for each transition t in SS, only one node can execute the input/output event specified in the transition t. We call the node the responsible node for the transition t, and denote it by Snode(t).

For each transition t, $\text{Snode}(t \bullet \bullet) \equiv \{\text{Snode}(s) \mid s \in t \bullet \bullet\}$ denotes the set of the next responsible nodes. $\text{Rnode}(r_h)$ denotes the set of nodes where the register r_h is allocated (as we mentioned before, each r_h may be allocated more than one nodes). For each transition t, $\text{Cset}(t)$ denotes the set of the registers which $\text{Snode}(t)$ has.

4. Deriving Protocol Specifications

4.1 Basic Idea

Here, we explain the outline of the behaviors of protocol entities. In the same way as Ref. [20], in general, for a transition t in SS, each PE_k executes some transitions of a sub-marked graph that simulates the transition t by exchanging messages among them. In general, after simulating transition t, there are many responsible nodes in $\text{Snode}(t \bullet \bullet)$ which can simulate transitions $t \bullet \bullet$. This is different point from Ref. [20] using an EFSM model. Furthermore, in MGR model, we can get efficient protocol entities using a property that some transitions can fire simultaneously. In sub Sect. 4.4, we will explain how to get efficient protocol entities.

First, the responsible node of the transition t ($\text{Snode}(t)$) executes the event of t. Note that the responsible node is defined uniquely for each t. If a node must change its registers' values based on the corresponding register definition statement, then in general it needs some registers' values which it does not have. Here, we assume that such registers' values are sent from other nodes through communication channels and the node receiving those values changes its registers' values by itself.

Each PE_k simulates each transition in SS in following five phases. Now suppose that a transition t is enable to fire.

(1) The responsible node $\text{Snode}(t)$ executes the event of the transition t.
(2) The responsible node $\text{Snode}(t)$ sends request messages to the nodes which must send current values of registers to the other nodes. These messages are called type α messages.
(3) Each node receiving the type α message sends the current values of registers as type β messages to the nodes which need the type β messages
(4) If a node receives all messages necessary for changing its registers' values, then the node changes its registers' values.
(5) The nodes that have changed their registers' values send synchronization messages to all nodes in $\text{Snode}(t \bullet \bullet)$. These messages are called type γ messages. And each node in $\text{Snode}(t \bullet \bullet)$ receives all type γ messages transmitted to the node.

In phase (2), the $\text{Snode}(t)$ also sends type η messages to the nodes which can change its registers' values by itself in order to notify the timing of changing their registers since such nodes have received neither type α nor β messages.

The responsible node $\text{Snode}(t)$ may send the current values of registers belonging to $\text{Cset}(t)$ and the values of the input variables to the nodes which need them in phase (2). We also regard such messages as type η messages.

4.2 Contents of Messages

Here, we do not consider the sizes of exchanged messages. We only consider the number of the messages because the number of the messages affects the communication costs more than the size of the messages. Each message contains the following information: (1) the label of the current transition, (2) the message type, (3) the values of input variables of the current transition (if they exist), and (4) the values of registers' names and their values (if they exist).

From the label of the current transition, each node receiving a message can know which transition is now firing. Also, if each next responsible node receives the transmitted messages, it can know which transitions have fired.

The exchanged messages have their types. Each responsible node may send a type α message and a type η massage to the same node. For such a case, the type α and η messages can be merged into one message. Here, the merged message is called a type μ message. Using the merged type μ messages, we can reduce the total number of the exchanged messages.

4.3 Decision of Transitions at Each Phase and Messages

Here, we can consider the following sub-problem **SYN**.
[Inputs:] SS, $\text{Alloc}(SS)$ and a transition t in SS
[Outputs:] transitions(including their labels, events, guards and register definition statements) in each phase for simulating the t at PE_k and messages (including their types and contents), where the phases are defined in Sect. 4.1.
[Condition:] If in each PE_k, all transitions of PE_k fire starting from phase 1 then these sequences must simulate the transition t and the number of messages is minimum.

In Ref. [20], using a procedure to calculate optimum solutions for 0-1 integer linear programming problems, a similar sub-problem is solved. In MGR model, we can apply a similar method in Ref. [20] to the sub-problem. Recall that there are many responsible nodes in $\text{Snode}(t \bullet \bullet)$ for a transition t in MGR model, we have to modify some inequalities of 0-1 integer linear programming which represent the constraints about messages so that type γ messages are sent to all members of $\text{Snode}(t \bullet \bullet)$. Here, the detail is omitted.

If a PE_k is not concern with all phases for the transition t in SS, then the PE_k executes a transition with only an internal event i. Such a transition is called an ϵ transition.

In order to solve the sub-problem **SYN**, we assume that each node PE_k has an additional working register r_0 for keeping received messages and that the following two functions *put* (...) and *get* (...) can be used. The function put(r_0,m) means that the message m (containing the values of the input variables and registers) is stored into the register r_0. The function get(r_0,r_h) represents that the latest value of the register r_h is stored in r_0.

4.4 Construction of PE_k Using Parallel Transitions

Here, we explain how to make a sub marked graph from the transitions obtained in the previous section, in order to reduce the number of steps which each PE_k needs to simulate one transition in the SS.

In general, the sub marked graph corresponds to a di-graph $G(V, E)$ if we treat a transition as a vertex of G. To minimize its execution steps, we will use the di-graph G. The corresponded MGR is easily got from the obtained di-graph G.

For each transition v of a PE_k which is obtained as a solution for the **SYN** in the previous section, we introduce a di-graph $G(V, E)$, where $v \in V$. A set of arcs E is got from data dependencies among the transitions.

Hereafter, we assume that $v \in$ (Ph n) means the transition v is defined in phase n as a solution for the **SYN**. Let $?(v, r)$ ($?(v, x)$) be a predicate whose value is true *iff* the transition v executes an input event and receives the value of the register r (the input variable x). Similarly, let $!(v, r)$ be a predicate whose value is true *iff* the transition v executes an output event and transmits the value of the register r. If the input/output data is not interested, We use $!(v)$ and $?(v)$ instead of $!(v, r)$ and $?(v, x)$, respectively.

For a transition t, let $RV(r_{h_k}, t)$ ($RV(x, t)$) be a predicate whose value is true *iff* that the register $r_{h_k}(1 \leqq k \leqq j)$ (the input variable x) is occurred in right side of some register definition statement $r_i \leftarrow f(r_{h_1}, \ldots, r_{h_j}, x)$ of the t. Similarly, let $LV(r_i, t)$ be a predicate whose value is true *iff* that the register r_i is occurred in a left side of some register definition statement $r_i \leftarrow f(r_{h_1}, \ldots, r_{h_j}, x)$ of the t.

The set of arc E is the maximum set where each arc $(v, s) \in E$ satisfies some of conditions in Table 1.

The constraint (1) represents that the register definition statements using the input variables must be executed after the input event is executed. Similarly, the constraint (2) represents that the value of the register r emitted in the transition v must be emitted before it is changed. The constraint (5) represents that type β messages are sent after receiving type μ messages. The constraints (7),(8) and (10) represent that the values of registers and input variables must be received before they are referred in register definition statements. The constraints (6) and (9) represent that the values of registers must be sent before they are changed. The constraint (11) represents that type γ messages must be sent after all registers are changed their values.

Table 1 Conditions for arcs in G.

(1)	$v \in$ (Ph 1) $\wedge ?(v, x)$ and $s \in$ (Ph 4) $\wedge RV(x, s)$
(2)	$v \in$ (Ph 1) $\wedge !(v, r)$ and $s \in$ (Ph 4) $\wedge LV(r, s)$
(3)	$v \in$ (Ph 1) $\wedge ?(v, x)$ and $s \in$ (Ph 2) $\wedge !(s, x)$
(4)	$v \in$ (Ph 1) and $s \in$ (Ph 5) $\wedge !(s)$
(5)	$v \in$ (Ph 2) $\wedge ?(v)$ and $s \in$ (Ph 3) $\wedge !(s)$
(6)	$v \in$ (Ph 2) $\wedge !(v, r)$ and $s \in$ (Ph 4) $\wedge LV(r, s)$
(7)	$v \in$ (Ph 2) $\wedge ?(v, r)$ and $s \in$ (Ph 4) $\wedge RV(r, s)$
(8)	$v \in$ (Ph 2) $\wedge ?(v, x)$ and $s \in$ (Ph 4) $\wedge RV(x, s)$
(9)	$v \in$ (Ph 3) $\wedge !(v, r)$ and $s \in$ (Ph 4) $\wedge LV(r, s)$
(10)	$v \in$ (Ph 3) $\wedge ?(v, r)$ and $s \in$ (Ph 4) $\wedge RV(r, s)$
(11)	$v \in$ (Ph 4) and $s \in$ (Ph 5) $\wedge !(s)$
(12)	$v \in$ (Ph 2) $\wedge !(v)$ and $s \in$ (Ph 5) $\wedge !(s)$
(13)	$v \in$ (Ph 3) $\wedge !(v)$ and $s \in$ (Ph 5) $\wedge !(s)$

In general, there may be many vertices, $v_1, \ldots, v_n$ which has no incoming (outgoing) arcs. In such a case, we introduce a new vertex $v_i(v_o)$ and arcs $(v_i, v_1), \ldots, (v_i, v_n)$ $((v_1, v_o), \ldots, (v_n, v_o))$. The vertex $v_i(v_o)$ corresponds to an ε transition.

A brief explanation of correctness for above algorithm is following. (1) The sub marked graph satisfies above conditions is also correct for a given transition t in SS, i.e. the sub marked graphs of each PE_k can simulate the t correctly. (2) For a sub marked graph of PE_k, the maximum number of transitions in any paths of PE_k is the minimum number of steps for PE_k to be able to simulate the t correctly. The property (1) is proved by checking each condition keeps the correctness. The property (2) is proved by contradiction. If there is a sub marked graph which has the smaller number of steps than that of our sub marked graph, then there is a pair of transitions that does not satisfy some of the conditions. This fact implies the sub marked graph does not work correctly. □

Figure 5 shows PE_1 which is constructed by simply connecting transitions in (Ph 1) to (Ph 5) in Sect. 4.3 sequentially. Using this method, we can derive a correct protocol entity specification PE_k. This is a simple solution. However, these protocol entities' specifications are less efficient than the protocol entities' specifications derived by the method given in this section. In Fig. 6, PE_1 derived by the method given in this section is shown. Figure 6 contains more parallelism than Fig. 5. PE_1 has six ε transitions.

4.5 Deletion of Redundant Transitions and Places

These ε transitions have no influence on the equivalence between SS and PE^{1-p} in the MGR model. However, to simplify a control part of each PE_k, some transformation rules are needed, which delete clearly redundant

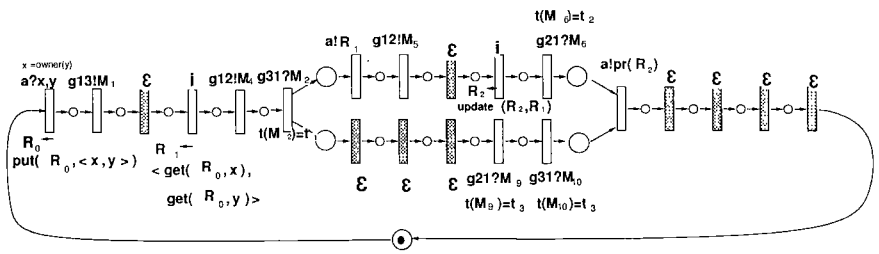


Fig. 5 PE_1 constructed in sequential style with ε transition.

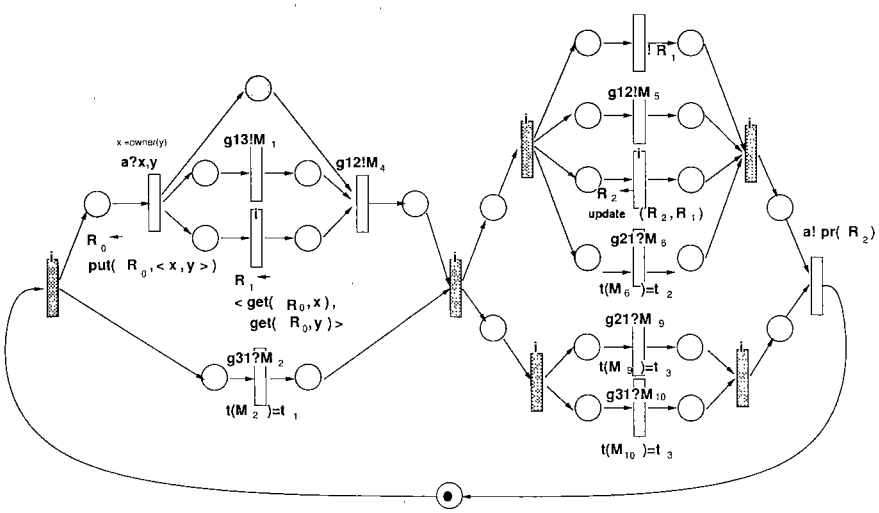


Fig. 6 PE_1 which has ε transitions.

transitions and places.

(S1) If there is a ε transition t that has only one input place p, then remove the place p, incoming/outgoing arcs of p and the transition t. And connect starting points of outgoing arcs of t to a transition t_0 that is an input transition of p (Fig. 7 (a)).

(S2) If there is a ε transition t that has only one output place, then remove the place p, incoming/outgoing arcs of p and the transition t. And connect ending points of incoming arcs of t to a transition t_0 that is an output transition of p (Fig. 7 (b)).

(S3) For a place p, if there is a direct path DP from an input transition t_a of p to an output transition $t_b(\neq t_a)$ of p, and if the direct path DP does not have the place p, then remove the place p and incoming/outgoing arcs of p (Fig. 7 (c)).

(S4) If there is a self loop with a place p having a token at the initial marking M_0, then remove the place p and incoming/outgoing arcs of p (Fig. 7 (d)).

By applying the rules (S 1) and (S 2), we can obtain more efficient protocol entities specifications. The rule (S 3) removes redundant places and the rule (S 4) is a special case of the rule (S 3).

We get PE_1 in Fig. 3 by applying the rules (S 1), (S 2) and (S 3) to PE_1 in Fig. 6.

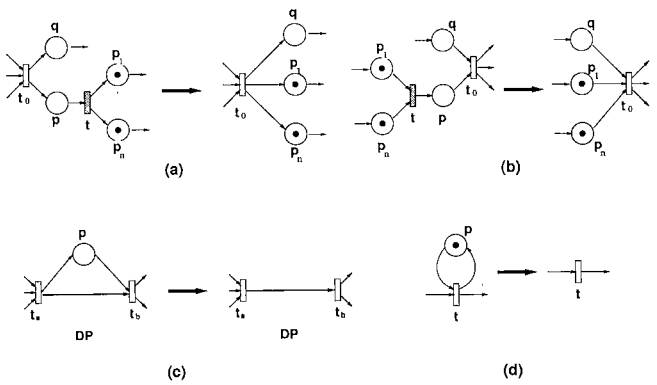


Fig. 7 Removing ε transitions and erdundant places.

4.6 Correctness of Our Method

The basic idea implementing each transition in a service specification by some cooperated nodes is based on the method for the EFSM model which we have proposed in Refs. [13], [20]. For the EFSM model, the correctness proof of the derivation algorithm is described in Ref. [19]. Using the similar technique, the correctness of the derivation algorithm for our MGR model can be also proved. Here, we only describe the difference between two proofs. In the MGR model, two transitions can be fired simultaneously. So, two types of the

messages may be exchanged simultaneously. However, those messages contain the labels for distinguishing the transitions each other. Therefore, each node can recognize which transition has been fired when it receives a message. Another difference is the number of tokens. In EFSM model, the number of tokens is always one. However, in MGR model, the number may become two or more. It may become zero in general. So, the deadlocks and/or livelocks may occur. However, since we assume safe and live marked graphs, such problems do not occur. This is the outline of the proof for the correctness of our algorithm.

5. Solutions for Avoiding Conflict of Registers

In this paper we have considered only conflict-free service specifications. Here, we discuss the service specifications with conflicts. Several methods for solving the conflicts of registers have been investigated [15], [16] (For survey see Ref. [14]). Here, we give two solutions.

5.1 Solution Using the Lamport's Algorithm

First, we consider the case that there are few conflicts in the service specifications. We use the Lamport's algorithm [15]. To make the discussion simply, we assume that each node has its own clock and all the clocks are correct and they show the same time. Note that Ref. [15] shows logical clocks using the counter registers and time stamps and so on can be substituted for these clocks.

The algorithm uses three kinds of messages "req", "reply" and "release", and each of these messages includes a pair (T_h, h) where h is the id of the transition which may occur conflicts of registers and T_h is the generation time of the message (time stamp). Each node has a queue Q_k which can sort entries by ordering of their time stamps. Now, suppose that responsible node Snode(t) (PE_k) will fire a sub marked graph which simulating the transition t in SS but may occur conflicts of registers. A "process" h for the transition t of PE_k exchanges some messages among other nodes in order to confirm the timing of executing the sub-marked graph for t, and then executes the sub-marked graph. Hereafter we call the sub-marked graph for t, the *critical section* for t.

(1) A process h which will enter the critical section for t in PE_k sends a message req(T_h, h) to all the nodes which are responsible nodes of transitions making the conflicts of registers with the transition t. And the process h enqueues the message req(T_h, h) to a queue Q_k.
(2) Each PE_i which has received the message req (T_h, h) enqueues the req(T_h, h) to its own queue Q_i and sends a message reply (T_i, i) to the PE_k.
(3) If PE_k receives all messages reply (T_i, i) which satisfy $T_i > T_h$ and the front entry of queue Q_k has id h, then the process h enters to a critical section for t. If the process h has gone out from the critical section, h sends messages release (h) to the nodes and dequeues the entry having the id h from its own queue Q_k.
(4) Each PE_i which has received the message release (h) dequeues the entry having the id h from its own queue Q_i.

We have to decide which nodes(PEs) send the messages among them. We can find the set of such transitions (i.e. nodes) by checking the concurrent firability [17]. For this purpose, some algorithms are known for many sub-classes of the Petri nets [18]. Especially, for the class of the live marked graphs used in this paper, a simple necessary and sufficient condition for deciding the concurrent firability is known [17].

An implement of the algorithm in an Extended MGR model (EMGR model) is shown in Fig. 8, where the EMGR model has many control parts modeled as marked graph in the same node. The marked graphs run independently, but they share same registers and same gates. The reason why this extension is needed in order to implement the algorithm is that the steps 2 and 4 of the algorithm run independently from the main stream of the algorithm (sending the req, receiving the reply, executing the critical section and sending the release). Note that the "release" messages are sent from one of the next responsible nodes of the transition t (PE_m). The PE_m is an arbitrary node of the next responsible nodes. This implement is different from the Lamport's original algorithm. This modification is needed, because the PE_k does not know when all PEs have finished the simulation of the t in general. The PE_m must know the id h, so, we assume that messages for simulating the t include also the id h when transition t makes the conflict of registers.

The algorithm has some advantages. It keeps fairness and the loads are shared by all the nodes impartially. On the other hand, the following disadvantages are considered. Let N be the number of nodes which has transitions that make the conflict of registers. Then the algorithm needs $3(N-1)$ messages. This algorithm does not use the information about the registers which should be locked.

5.2 Solution Using Another Algorithm

By using above information, more efficient solutions can be made. The following is an another solution. In the second algorithm, only one node controls the mutual exclusion. That is, the centralized controlled method is considered. This algorithm uses the information of registers. If a PE_k wants to execute a critical section that may make the conflicts, then the PE_k sends a request message to the PE_N. The request message includes the names of all registers appeared in the register definition statements used in the transition and time stamp. If the PE_N sends a reply message to the PE_k then the PE_k

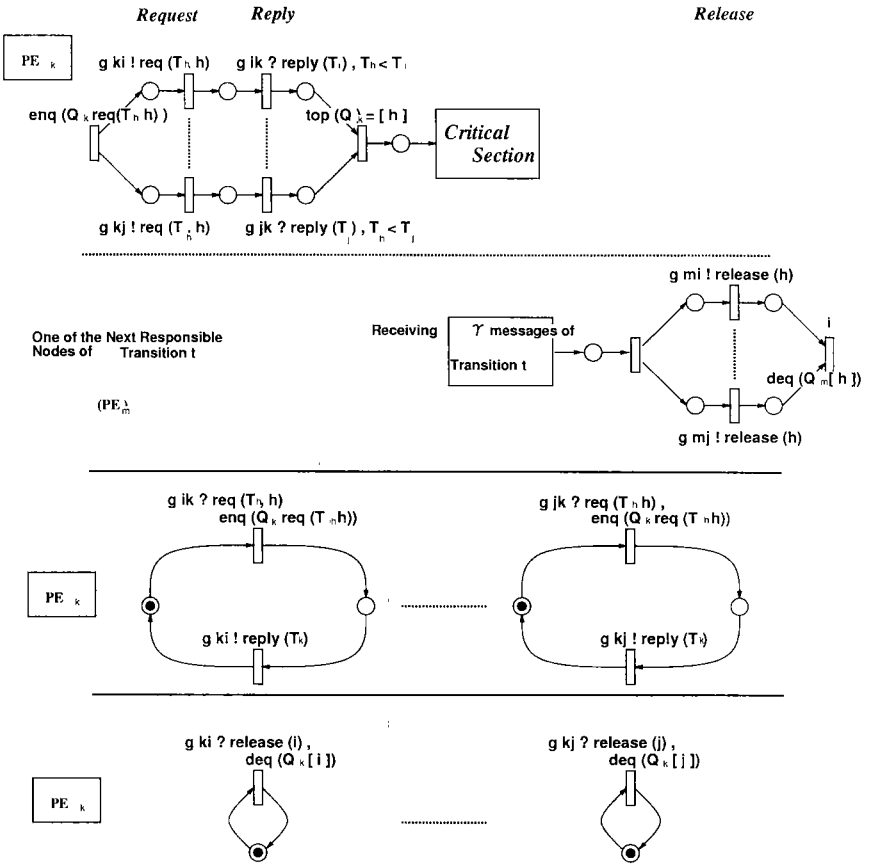


Fig. 8 Algorithm by Lamport.

executes the critical section and then one of the next responsible node sends the PE_N a release message that includes the same registers' names.

The PE_N has a table T of all registers' names for checking "lock" or "unlock" and a queue Q. If the PE_N receives the request message m, then it enqueues the message m to the queue Q. For the front message l of Q, the PE_N checks whether all registers' names in l are "unlock" in the table T. If they are all "unlock", then the PE_N changes them to "lock", dequeues the l from Q and sends a reply message to the sender of the l. If the PE_N receives the release message, then it changes the corresponding registers' names to "unlock" in T.

Although the method is a centralized controlled method, the number of the messages necessary for avoiding the conflict is small. And if the method is used, the deadlock does not occur because all requested resources are locked at the same time.

6. Correctness of Algorithm for SS with Conflicts

An overview of a correctness proof for the derivation algorithm using the Lamport's algorithm is following.

For the correctness of the simulating part, see Sect. 4.6. For the conflict of registers, see Ref. [15]. We also have to make sure that there are no deadlocks and no confusing, especially following points. (1) There are no deadlocks, even if messages for simulating and mutual exclusion are sent through the same communication channel. (2) There are no deadlocks caused by the guards for receiving reply messages. The point (1) is guaranteed by the types and identifiers of messages. The point (2) can be proved as follows. If such a message (message which does not satisfy the guard condition) is arrived to a PE_k, then the message is either (i) a message for simulating some transition t, (ii) the "req" message, (iii) the "release" message, or (iv) the "reply" message which has an old time stamp. For the first case, there is a transition which receives the message in the PE_k. For the second and third cases, the message is always received because the MGRs for receiving such messages run independently (see Fig. 8). For the last case, there must be a transition which has sent a "req" message corresponding to the "reply" message. Then there must be a transition which can receive the "reply" message, because of the construction in EMGR. We have to proof the fact that the "release" messages are sent from PE_m (in general, $PE_m \neq PE_k$) does not cause no harmful influences. The PE_m knows the message id h, because the h is included by the γ messages which the PE_m received. Therefore, the release(h) are sent at the end of the critical section for the transition t. Because we

select one node from the next responsible nodes, each "release" message is sent to a node which needs it at exactly once.

The correctness of the second algorithm is not given here, it is one of the future works.

7. Conclusion

In this paper, an algorithm to derive a correct protocol specification from a given service specification modeled as a MGR model.

The basic idea for implementing each transition of a service specification is similar to the method in Refs. [13], [19]. However, in MGR model, the conflict of register may occur. In this paper, we give solutions for solving the problem. When each node implements a given transition in MGR model, some events may be able to be executed in parallel. In Refs. [13], [19], such parallelism is not considered. However, in this paper, we describe how to minimize the simulation steps using parallel transitions. Those are new ideas in this paper.

In our MGR model, we can describe parallelism in a service specification. However, since the number of the arcs from each place is always one in our MGR model, the model can not treat selective events where one of the outgoing arcs is selected depending on the values of the registers and input variables.

One of the future works are get more efficient algorithm than shown in Sect. 5 and give a correctness proof of it. And to extend the model so that the selective actions can be expressed is also considered. A free-choice net can be considered as such a model.

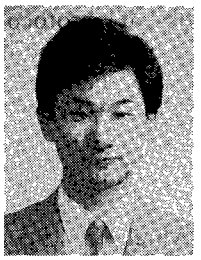
Acknowledgement

The authors would like to thank Professor T. Murata and T. Araki of Osaka University for their useful comments for improving our paper. They also would like to thank referees, whose comments are useful for improving this paper.

References

[1] Murata, T., "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE*, vol.77, no.4, pp.541–580, 1989.
[2] Probert, R. and Saleh, K., "Synthesis of Communication Protocols: Survey and Assessment," *IEEE Trans. on Comp.*, vol.40, no.4, pp.468–476, 1991.
[3] Khendek, F., Bochmann, G.v. and Kant C., "New results on deriving protocol specifications from service specifications," *Proc. of ACM SIGCOMM '89*, pp.136–145, 1989.
[4] Bochmann, G.v. and Gotzhein, R., "Deriving protocol specifications from service specifications," *Proc. of ACM SIGCOMM '86*, pp.148–156, 1986.
[5] Langerak, R., "Decomposition of functionality; a correctness-preserving LOTOS transformation," *Proc. of Tenth IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification*, North Holland, pp.229–242, 1990.
[6] Higashino, T., "Service Specification and Its Protocol Specifications in LOTOS," *IEICE Trans. on Fundamentals*, vol.E75-A, no.3, pp.330–338, Mar. 1992.
[7] ISO, "Information Processing System, Open Systems Interconnection, LOTOS–A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," IS 8807, Jan. 1989.
[8] Chu, P.-Y.M. and Liu, M.T., "Synthesizing Protocol Specifications from Service Specifications in FSM model," *Proc. Computer Networking Symp. '88*, pp.173–182, Apr. 1988.
[9] Chu, P.-Y.M. and Liu, M.T., "Protocol Synthesis in a State-Transition Model," *Proc. COMPSAC'88*, pp.505–512, 1988.
[10] Park, D., "Concurrency and automata on infinite sequences," *Theoretical Computer Science, Lecture Notes in Computer Science*, vol.104. pp.167–183, Springer-Verlag, 1981.
[11] Igarashi, H., Kakuda, Y. and Kikuno, T., " Synthesis of Protocol Specifications for Design of Responsive Protocols," *IEICE Trans. on Information and Systems*, vol.E76-D, no.11, pp.1375–1385, Nov. 1993.
[12] Milner, R., *Communication and Concurrency*, Prentice-Hall, 1989.
[13] Higashino, T., Okano, K., Imajo, H. and Taniguchi, K., "Deriving Protocol Specifications from Service Specifications in Extended FSM Models," *Proc. of 13th IEEE Inter. Conf. on Distributed Computing Systems*, pp.141–148, May 1993.
[14] Lamport, L. and Lynch, N.A., "Distributed Computing: Modes and Methods," *in Hand Book of Theoretical Computer Science, B: Formal Models Semantics*, van Leeuwen, J. et al. Ed., The MIT Press/Elsevier, pp.1157–1200, 1990.
[15] Lamport, L., "Time, Clocks and Ordering of Events in a Distributed System," *Comm. ACM*, vol.21, no.7, pp.558–564, 1978.
[16] Maekawa, M., "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Trans. on Computer Systems*, vol.3, no.2, pp.145–159, 1985.
[17] Leu, D. and Murata, T., "Properties and applications of the token distance matrix of a marked graph," *Proc. 1984 IEEE Int. Symp. Circuits Syst.*, vol.3, pp.1381–1385, 1984.
[18] Takahashi, M., Araki, T. and Kashiwabara, T., "Concurrent Firability of Transitions on Restricted Petri Net," *Trans. IEICE*, vol.J77-D-I, no.1, pp.1–11, Jan. 1994.
[19] Okano, K., Imajo, H., Higashino, T. and Taniguchi, K., "Synthesis of Protocol Entities' Specifications from Requirement Specification for Distributed System in EFSM Model.," *IPSJ Tech. Rep.*, 92–PRG–8, pp.211–218, 1992.
[20] Okano, K., Imajo, H., Higashino, T. and Taniguchi, K., "Synthesis of Protocol Entities' Specifications from Service Specification of Distributed System in Extended Finite State Machine Model," *IPSJ*, vol.34, no.6, pp.1290–1301, 1993.

Hirozumi Yamaguchi was born in Nagoya, Japan on July 18, 1971. He received the B.E. degree in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1994. He has been with the Department of Information and Computer Sciences in Osaka University, where he currently belongs in a postgraduate course. His research interests are design and analysis of distributed computing in the petri net model.



Kozo Okano was born in Osaka, Japan on June 12, 1967. He received the B.E. and M.E. degrees in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1990 and 1992, respectively. He joined the faculty of Engineering Sciences, Osaka University in 1993. Since then, he has been a Research Associate in the Department of Information and Computer Sciences at Osaka University. His current research interests include design and analysis of distributed systems and specification and verification of program design. He is a member of IPS of Japan.

Teruo Higashino was born in Osaka, Japan on June 8, 1956. He received the B.E., M.E., and Ph.D. degrees in Information and Computer Sciences from Osaka University, Osaka, Japan, in 1979, 1981 and 1984, respectively. He joined the faculty of Osaka University in 1984. Since 1991 he has been an Associate Professor in the Department of Information and Computer Sciences at Osaka University. In 1990, he was a Visiting Researcher of Dept. I.R.O. at University of Montreal, Canada. His current research interests include design and analysis of distributed systems, specification and verification of communication protocols, and formal approach of program design. He is a member of IEEE-CS, ACM, and IPS of Japan.

Kenichi Taniguchi was born in Wakayama, Japan on March 17, 1942. He received the B.E., M.E. degree in electronics engineering and Ph.D. degrees in control engineering from Osaka University, Osaka, Japan, in 1965, 1967 and 1970, respectively. He joined the faculty of Osaka University in 1970. Since 1986 he has been a Professor in the Department of Information and Computer Sciences at Osaka University. His current research interests include design methods for distributed systems and communication protocols and algebraic methods for software development and hardware design and verification.