# A Design Method for Communications Software Evolution

Hironobu OKUYAMA, Kenji MORIYASU, Yutaka HIRAKAWA,
E-mail: { okuyama, moriyasu, hirakawa}@slab.ntt.jp
3-9-11 Midori-Cho, Musashino-Shi, Tokyo, 180, Japan
NTT Software Laboratories

## Abstract

*This paper discusses a verification method for communication services on a network in which new service functions are added to only some of the network nodes.*

*Communication networks are becoming larger, and new functions are constantly required for communication services. In a large-scale network, however it is very difficult to update all node functions simultaneously. In this case, in a single network there will be nodes that have both new and existing functions and other nodes having only existing functions. This situation might result in illegal actions in communications between functionally updated nodes and non-updated nodes.*

*We clarify the fundamental properties of illegal actions, present an algorithm for detecting them and show that the algorithm is practical enough from the viewpoint of the order of complexity.*

**Key words** – *Network, Verification, Communications Software, Communication Protocols.*

## 1 Introduction

Communication networks are becoming larger, and new functions are constantly required for communication services. To satisfy user demands, new functions must be added to each network node. However, in a large-scale network, it is very difficult to update all node functions simultaneously.

It would be useful if new functions could be added to only part of the network. However, this would result in the coexistence of nodes having both new and previously existing functions and nodes having only existing functions on the same network. When nodes having different functions communicate with each other, illegal actions are likely to occur.

Similar problem has been discussed from various points of view. A system modification or extension method [8] has been proposed in which the range of processes that must stop synchronously is clarified by utilizing server and client relationships. Also, an execution mechanism for on-line system extension [9] has been proposed synchronous process stop is not necessary. This method uses parallel execution of old and new processes and rollback. However, there is no report from the verification point of view.

There are many research reports for verification of these methods [3, 4, 5, 6]. In the conventional method, however, the objective is to design a correct specification assuming there is an unique version of specification. Conventional methods are also applicable to a service where multiple service specification versions exist. However, such methods requier combinatorially large number of verification execution, when service specification consists of multiple processes, and when it is unknown whether each process follows old specification or new specification. Thus, conventional methods are not directly applicable to the above problem.

The paper clarifies the fundamental properties of illegal actions that occur in a heterogeneous network and proposes a fast algorithm for detecting these illegal actions.

This paper is organized as follows: Section 2 gives an example of an illegal action. Section 3 models a communications system and introduces some definitions. Section 4 presents properties of illegal actions and section 5 presents our algorithm for detecting illegal actions.

## 2 Example

Let us first consider the case where illegal actions do not occur when all the network nodes are given a new function but do occur when only some of the nodes are given it.

We will use as an example the "electric notice board" service. Its key specifications are shown in Figure. 1.



0-8186-6685-4/94 $4.00 

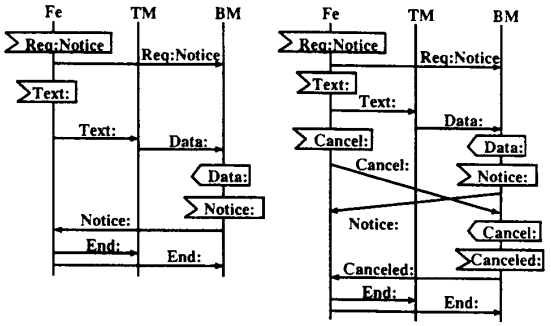


Figure 1: Electric Notice Board (Existing functions)

The electric notice board service consists of three processes: a front end (FE), a text manager (TM) and a board manager (BM). The service is used within a company. Each floor of the building has each type of process and usually the service area is restricted to one floor.

When the FE receives a text string from a user, the FE sends it to the the TM where it is transformed into formatted data. The TM sends it to the BM, and the board manager posts it on a database (board), which is outside the specifications in Figure. 1. There is only one database for the service, so all the board managers on every floor use the same database.

To cancel the text string, the user sends a message "Cancel:" to the FE. Then the posted data is erased and a message "Canceled:" is sent back to the FE for confirmation. (Here the function for reading is omitted.)

Now consider the addition of a new function, as shown in Figure. 2.

The new function is as follows: if the message "Cancel:" is received earlier than the message "Data:" by the BM, the "Data:" is canceled before being posted. But if the message "Data:" is received earlier by the BM, the data is canceled after being posted according to the existing function shown in Figure. 1.

By verifying this specification with a new function assuming that the specification is used only on one floor, we can confirm that illegal actions do not occur. Now consider what happens if the BM is out-of-order on a floor that does not have the new function. In that case, to use the electric notice board service, the FE must access the process on another floor. If the BM has the new function (see Figure. 3), an illegal action may occur as follows:

Step 1: Input message "Cancel:" to an FE that does not have the new function.

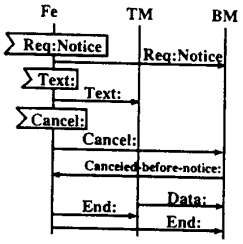


Figure 2: Electric Notice Board (New function)

Step 2: The message "Cancel:" is received before the message "Data:" by a BM that has the new function.

Step 3: The BM sends a message "Canceled-before-notice:" to the FE.

Step 4: This FE, which does not have the new function, cannot receive the message "Canceled-before-notice:" (i.e., unspecified reception occurs).

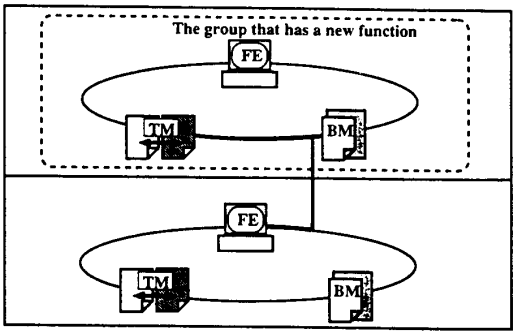


Figure 3: Addition to only the pert of the network

In the following, we discuss a method for detecting illegal actions when new functions are added to only some of the network processes.

## 3 System Model

The service example described in the previous section is executed by three communicating processes, the FE, TM, and BM, and one process not shown in Figure. 1. The members of the service processes vary but the number of service processes is always fixed. This type of service is conventionally modeled by a fixed number of communicating processes. We use the term "system" to indicate a fixed number of communicating processes used for service execution.

In this paper, specifications are assumed to be given in the form of message sequence charts (MSCs). We use a transformation method besed on using MSCs to communicating finite state machines (FSMs), which is proposed in [3].

### 3.1 System

The system can be modeled as communicating finite state machines consisting of processes and FIFO (first-in, first-out) channels between the processes.

Here, $N$ represents the number of processes, $Q_h$ represents the set of states of process $p_h$, $o_h$ represents the initial state of process $p_h$, $F_h = \{f_{h1}, ..., f_{h\tau(h)}\}$ represents the set of final states of process $p_h$, $M_{hk}$ represents the set of messages that can be carried through the channel from process $p_h$ to process $p_k$, and $succ_h(s_h, x)$ represents the state of process $p_h$ after message $x$ is sent at state $s_h \in Q_h$ if $x \in M_{hk}$ , or the state of process $p_h$ after message $x$ is received at state $s_h \in Q_h$ if $x \in M_{kh}$.

Definition 1 (System)

System P is formally defined as a quadruple $P = (Q, o, M, succ)$, where $Q = Q_1 \cup ... \cup Q_N$, $o = (o_1, ..., o_N)$, $M = \bigcup_{h \neq k} M_{hk}$, and $succ = (succ_1, ..., succ_N)$ where $succ_h$ is a partial function mapping for each $h$ and $k$ $(h \neq k)$, and $Q_h \times (M_{hk} \cup M_{kh}) \rightarrow Q_h$. Note that $o_h \in Q_h$, $Q_h \cup Q_k = \phi$ for each $h, k (h \neq k)$.

Definition 2 (Global State)

Global state G is defined as a pair $G = (S, C)$, where $S = (s_1, ..., s_N), s_h \in Q_h$, $C = (c_{11}, ..., c_{1N}, c_{21}, ..., c_{NN})$, and $c_{hk}$ is a sequence of messages in $M_{hk}^* (1 \leq h, k \leq N, h \neq k)$. Here, $M_{hk}^*$ represents $\phi$ or $M_{hk} \times M_{hk} \times ... \times M_{hk}$.

Also, $G_0 = (S_0, C_0)$, where $S_0 = (o_1, ..., o_N)$ and $C_0 = (\varepsilon, ..., \varepsilon)$ is called the initial global state, and $\varepsilon$ denotes an empty sequence of messages.

$G = (S', C_0)$ , where $S' = (f_{1\lambda(1)}, ..., f_{N\lambda(N)})$ and $C_0 = (\varepsilon, ..., \varepsilon)$ is called a finite global state.

Definition 3 (Global state transition)

A global state transition is defined as a binary relation $\vdash$ on global states such that $(S, C) \vdash (S', C')$ if and only if there exist $h, k$, and $x$ satisfying either of the following conditions:

(1) All the elements of $(S, C)$ are equal to those of $(S', C')$ except $s'_h = succ_h(s_h, x), x \in M_{hk}$, and $c'_{hk} = c_{hk} \cdot x$.

(2) All the elements of $(S, C)$ are equal to those of $(S', C')$ except $s'_h = succ_h(s_h, x), x \in M_{kh}$, and $c_{kh} = x \cdot c'_{kh}$.

Here, the global state transition $\vdash$ means that one global state produces the other by the transmission or reception of a message. The symbol " $\cdot$ " denotes a concatenation of two sequences of messages.

Definition 4 (Reachable)

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$. Then, a global state $G = (S, C)$ is reachable if and only if $G_0 = (S_0, C_0) \vdash^* G$ , where $S_0 = (o_1, ..., o_N)$ and $C_0 = (\varepsilon, ..., \varepsilon)$.

Definition 5 (Illegal action)

A sequence of global state transitions $G^0 \vdash^* G$ is called an illegal action if and only if $G$ is not a final global state and no global state $G'$ such that $G \vdash^* G'$ exists. Here, $G$ is called an illegally stopped global state. At the illegally stopped global state, if there is a channel $c_{hk} \in C$ in which a message remains (i.e., $c_{hk} \neq \varepsilon$), and the state of process $p_k$ is $s_k$, we say that unspecified reception occurs at $s_k$ that is a state of process $p_k$ .

### 3.2 Causal Relationship

A MSC defines a partial ordering of the events. The partial order "$(\rightarrow)$" can be defined as follows:

The relation "$(\rightarrow)$" on the set of events of a system is the smallest relation satisfying the following three conditions:

1. If $e_1$ and $e_2$ are events in the same process, and $e_1$ comes before $e_2$, then $e_1 \rightarrow e_2$.
2. If $e_1$ is the sending event of a message by one process and $e_2$ is the reception event of the same message by another process, then $e_1 \rightarrow e_2$.
3. If $e_1 \rightarrow e_2$ and $e_2 \rightarrow e_3$ then $e_1 \rightarrow e_3$.

Two distinct events $e_1$ and $e_2$ are said to be concurrent if and only if $e_1 \not\rightarrow e_2$ and $e_2 \not\rightarrow e_1$. We assume that $e \not\rightarrow e$ for any event e. When $e_1 \rightarrow e_2$, we say "event $e_1$ happens before event $e_2$", or "event $e_2$ happens after $e_1$".

We call this relationship a "causal relationship". It is the same concept as "happened before" proposed by Lamport [1].

## 4 Properties of an Illegal Action

### 4.1 Concepts and Symbols

We use the following concepts and symbols.

The specification of existing functions is given by a set of MSCs $\{MSC_1, ..., MSC_l\}$, and the system defined by it is called the system E. In the same way, the specification of new functions is given by a set of MSCs $\{MSC_{l+1}, ..., MSC_m\}$, and the system defined by the specification $\{MSC_1, ..., MSC_l, MSC_{l+1}, ..., MSC_m\}$ is called E+A.

A system that is given new functions on some of its processes is called E+(A). Note that there are many different systems called E+(A). In general, the number of different systems E+(A) is $2^N - 1$, because the number of the processes is $N$ and each process could have new functions added.

Assume that process $p_h$ does not have new functions in system E+(A). State $s_h$ of process $p_h$ is reachable by existing functions if and only if there is a reachable global state $G = (S, C)$ where $S = (s_1, ..., s_h, ..., s_N)$.

Existing event: An event defined by existing functions.

New event: An event defined by new functions and not defined by existing functions.

New trigger: A new event defined at a state that is reachable by existing functions.

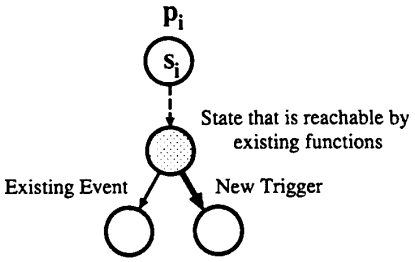


Figure 4: New Trigger

## 4.2 Assumptions

The given specification satisfies all of the following conditions:

(A1) There are no illegal actions in system E.

(A2) There are no illegal actions in system E+A.

(A3) Every event sequence of E or E+A is involved in the specifications.

(A4) A process state is one of the following:

(i) A state where no event is defined (i.e., a final state).

(ii) A state where only one message-sending event is defined.

(iii) A state where one or more message-receiving events are defined.

(A5) No event sequence of the system contains a loop.

The properties of an FSM depend on the transformation method. Assumption (A4) is based on the method proposed in [3].

We use the verification method proposed in [3] for verifying (A1), (A2), and (A3). The verification method also assumes (A4) but allows restricted loops. These two restrictions are key assumptions for fast verification. Applicability of the verification method is already shown in [7] where the method is applied to commercial PBX software design. The assumption (A5) is able to be relaxed but loops in specifications are not treated in this paper.

## 4.3 Properties

**Property 1**

Assuming (A1)-(A5), the following property holds: If an illegal action will occur in system E+(A), at least one unspecified reception occurs in the processes.

[Proof] Suppose that system E+(A) involves an illegal action, but unspecified reception does not occur. Assume that system E+(A) illegally stops at a global state $G$ where $G = (S, C), S = (s_1, ..., s_N), C = (\varepsilon, ..., \varepsilon)$, and at least one state $s$ is not a final state. Note that $s_h$ $(1 \leq h \leq N)$ must be a final state or a state where one or more receive events are defined. Now $G$ is also reachable in system E+A. Notice that for existing states, the differences between E+A and E+(A) are additional receive events definitions at existing state where one or more receive events are defined. As all channels are empty at $G$, there is no transition from $G$ in system E+A. This is inconsistent with assumption (A2). □

**Property 2**

Assuming (A1)-(A5), the following property holds: If unspecified reception occurs in system E+(A), there exists a state $s$ such that unspecified reception occurs in system E+(A) and new triggers are defined at the same state $s$ in system E+A.

[Proof] Assume that global state $G$ is reachable in system E+(A). Thus, $G$ is also reachable in system E+A.

By the definition of illegal action, there is no transition from $G$ in system E+(A). On the other hand, by assumption (A2), there is no illegal action in system E+A. This requires that there is at least one executable event from $G$ in system E+A. Assume that the state of process $p_k$ is $s_k$ when system E+A is in global state $G$. As there is no executable event from $G = (S, C)$ where $S = (s_1, ..., s_h, ..., s_N)$ in system E+(A), each $s_h$ is a final state or a state where one or more receive events are defined. However, there is an executable event $e_k$ of process $p_k$ from $G$ in system E+A and process $p_k$ does not have new functions in system E+(A), so the followings holds:

- $e_k$ is a new receive event defined at $s_k$ of process $p_k$.
- $s_k$ is reachable by existing functions.

Thus, $e_k$ is a new trigger.

As $e_k$ is a receive event of process $p_k$, there is a non-empty receive channel of process $p_k$ in $G$. This message should be received by the new trigger. This indicates that unspecified reception occurs at $s_k$ of process $p_k$ in $G$ of system E+(A). □

From properties 1 and 2, to detect illegal actions, it is sufficient to detect whether unspecified reception occurs in system E+(A) at the state where the new trigger is defined in E+A.

We define new concepts as follows: Here we assume that the new trigger e is defined at the state $s_h$ of process $p_h$.

ET(e): A set of messages of existing events which is defined at the state $s_h$.
QM(e,i): A set of messages $x$ satisfying the following two conditions:
(1) Event $e_r$ of process $p_h$ which receives message $x$ is the first receive event from process $p_k$ and $e \rightarrow e_r$ on $MSC_i$.
(2) For the new trigger e and the event $e_s$ of process $p_k$ where event $e_s$ sends message $x$ which is received by event $e_r$ of process $p_h$ on $MSC_i$, $e \not\rightarrow e_s$ holds.

We make the meaning of QM clear in the following property 3.

**Property 3**

Assuming (A1)-(A5), the following property holds: When the system follows $MSC_i$, the set QM(e,i) is equal to the set of messages that can exist at the head of the receive channel just before event e of process $p_h$ is executed.

[Proof] Assume that the system follows $MSC_i$, process $p_h$ reaches state $s_h$ where event e is defined, and a message $x$ sent by event $e_s$ is received by event $e_r$ of process $p_h$. If $e \rightarrow e_s$ holds, $e_s$ can not occur. Otherwise, if $e \not\rightarrow e_s$ holds, $e_s$ is executable and sends message $x$ to process $p_h$.

Because e and $e_r$ are events of process $p_h$, the relationship between them is $e \rightarrow e_r$ or $e_r \rightarrow e$. If $e_r \rightarrow e$, the message $x$ is consumed until the process $p_h$ reaches the state at which event e is defined, and message $x$ does not remain in the channel. Otherwise, if $e \rightarrow e_r$, message $x$ can remain in the channel.

And it is clear that message $x$ can exist at the head of the channel $c_{kh}$ when process $p_h$ is at the state $s_h$ if and only if $e_r$ is the first receive event that satisfies $e \rightarrow e_r$ on $MSC_i$. □

**Property 4**

Assuming (A1)-(A5), when a system follows $MSC_i$, the following property holds: Here at state $s$ of process $p$, a new trigger is defined in system E+A.

In system E+(A), an unspecified reception occurs at state $s$.
$\Longleftrightarrow \exists$ i s.t. $ET(e) \cap QM(e,i) = \phi$
[Proof]
($\Longleftarrow$) From property 3, $ET(e) \cap QM(e,i) = \phi$ means that the intersection of the set of messages received by the existing events defined at state $s$ and the set of messages which can exist at the heads of the receive channels at state $s$ is empty.

Thus, when a system follows $MSC_i$ and new functions are not added to process $p$, process $p$ can no longer transit in system E+(A). Thus, the message that should be received by the new trigger causes unspecified reception to occur at process $p$.
($\Longrightarrow$) We will show the contraposition
( i.e., $ET(e) \cap QM(e,i) \neq \phi \Longrightarrow$ An unspecified reception does not occur at state $s$ in system E+(A)).

If $ET(e) \cap QM(e,i)$ is not empty, process $p$ receives a message and transits to the next state. Therefore, unspecified reception does not occur at state $s$. □

Using properties 1-4, we can get the next theorem.

**Theorem**

Assuming (A1)-(A5), the following holds:

An illegal action occurs in system E+(A).
$\Longleftrightarrow$ There exists i such that $ET(e) \cap QM(e,i) = \phi$ is true for $MSC_i$ where an execution of a new trigger e is specified.
[Proof]
($\Longrightarrow$) When an illegal action occurs in system E+(A), from property 1 there is at least one process where un-

specified reception occurs. Assume that an unspecified reception occurs at state $s$ of process $p$. From property 2, a new trigger is defined at $s$ in system E+A. Now, we get the right side of the above equation.
($\Longleftarrow$) When we assume the right side of the above equation, it is clear from property 4 that an unspecified reception occurs in system E+(A). □

Using this theorem, we can judge whether or not illegal actions will occur when we add new functions to only a part of the network.

# 5 Algorithm for Detecting Illegal Actions

## 5.1 Algorithm

This subsection describes how the above theorem can be used in an algorithm that detects illegal actions.

**Step 1:** Transform given MSC specifications into process specifications.

**Step 2:** Classify events (existing events or new events) by comparing the specifications of each process with the MSC specifications. In this step, we can get the set of new triggers $\Sigma$ and the set of MSCs where an execution of a new trigger e is specified. The latter set is called SM(e).

**Step 3:** Extract a new trigger e from $\Sigma$. Here, e is assumed to be an event defined at state $s$ of the process $p_h$.

**Step 4:** (Calculate set ET(e)) Collect all the messages of existing events defined at state $s$. The set of those messages is called ET(e).

**Step 5:** (Calculate set QM(e,i)) Extract an $MSC_i$ from set SM(e) and get all the messages that can exist at the heads of the receive channels of process $p_h$ when the state of $p_h$ is $s$. The set of those messages is called QM(e,i).

(Further details are described in the following subsection)

**Step 6:** Check the intersection of ET(e) and QM(e,i). If it is empty, the algorithm terminates and reports "an illegal action occurred in the specification". If it is not empty and set SM(e) is not empty, go to step 5. If the set SM(e) is empty, go to step 7.

**Step 7:** If $\Sigma$ is not empty, go to Step 3. If the set $\Sigma$ is empty, the algorithm terminates and reports "in the specification, no illegal actions occur".

**[Algorithm for calculating QM]**

Here, we explain the algorithm for calculating QM in detail as mentioned in step 5 of the previous algorithm.

Suppose we examine the new trigger defined at state $s$ of process $p_h$ on $MSC_i$.

**Step 0:** To store the messages that may exist at the heads of the receive channels, we use an array H of size N (N is the number of processes) in which each initial value is null. We manage the creation of tokens for detection by using an array I of size N × N in which each initial value is null. When a new token is created, a new identification number is given to it. Identification numbers and the places of the tokens (i.e., their process name and event name) are registered in order in a list T.

At the end of the algorithm, the set of messages in array H is equal to the set QM(e,i).

The (d,k) element of array I (i.e., I(d,k)) indicates whether a message from process $p_d$ to process $p_k$ has been sent or not after the new trigger occurs.

**Step 1:** For process $p_h$, do the following operation. The operation starts from state $s$ and terminates at the final state.

(Case 1.1) In the case of encountering an event receiving message $x$ from process $p_k$: If H(k)=null, store the message name $x$ in H(k), and mark the event that receives message $x$ with "○". Otherwise, mark the event with "△".

(Case 1.2) In the case of encountering an event sending a message $x$ to process $p_k$: If I(h,k)=null, change the value of I(h,k) to "gen", mark the event that receives message $x$ with "×", create a new token for detection on the receive event and give a new identification number to the token. The identification number and the place of the token is registered at the end of T.

If I(h,k)= gen, do nothing and go to the next event.

**Step 2:** If list T is null, terminate the algorithm. If it is not null, remove the token from the head of list T and perform the following operation on

the token. The operation starts from the event at which the token exists (we assume that it is the event of process $p_d$), and terminates when it encounters the event marked "×" or reaches the final state. When the operation terminates, remove the token in both cases, and go back to the beginning of step 2.

(Case 2.1) In the case of encountering a reception event: Mark the event with "×".

(Case 2.2) In the case of encountering an event sending a message $x$ to process $p_k(k \neq h)$: If I(d,k)=null, change the value of I(d,k) to "gen". Also mark the event that receives message $x$ from $p_k(k \neq h)$ with "×", create a new token on the event, and give a new identification number to it. The identification number and the place of the token is registered at the end of T.

If I(h,k)= gen, do nothing and go to the next event.

(Case 2.3) In the case of encountering an event sending a message $x$ to process $p_h$: If the event receiving the message $x$ is marked "○", change the mark to "×", and change the value of H(d) to "none". If the event received the message $x$ is marked "△", change the mark to "×".

At the end of the algorithm, messages received by events marked "○" are included in an array H. The set QM(e,i) is constructed by messages in the array H.

The reason is explained informally as follows: Consider a message $x$ sent by event $e_s$ of process $p_k$ and received by event $e_r$ of process $p_h$. If $e \rightarrow e_r$ does not holds, $e_r \rightarrow e$ holds because e and $e_r$ are events of process $p_h$. When $e_r \rightarrow e$ holds, $x$ is consumed until the process $p_h$ reaches the state at which event e is defined. Thus, $x$ must not be an element of QM(e,i).

If $e \rightarrow e_s$ holds, $e_r$ is marked "×" in the algorithm and the message $x$ must not an element of QM(e,i). Next, if $e \not\rightarrow e_s$ and $e \rightarrow e_r$ hold, and $x$ is not the first receive message from process $p_k$, $e_r$ is marked "△" and $x$ is not an element of QM(e,i), either.

In a sense, the message received by the event marked "△" is the message which exist but does not exist at the head of the channel.

Thus, for an event $e_r$ marked "○", $e \not\rightarrow e_s$ and $e \rightarrow e_r$ holds, and $x$ is the first received message from process $p_k$. This is the definition of the set QM(e,i).

The above algorithm is fundamental.

Assume that a new function is implemented using interactions of new messages. Also assume a user uses an updated function when his process communicates with a non-updated process. Trivially, illegal action occurs. Thus, the allowable range of function updates seems to be very small.

However, consider if members of a group agree to add a new function within their group. They should understand that they cannot use updated functions when they communicate with processes outside the group, because communication between updated processes and non-updated processes results in illegal actions.

Thus, we can assume that a user on an updated process does not use updated functions when he communicates with a non-updated process. We call this assumption the "optimistic assumption". Under this assumption, the allowable range of function updates is sufficiently wide.

It is easy to adapt our algorithm to this optimistic assumption, but details of the adaptation are omitted in this paper.

Here we only mention the essential point of the detection algorithm under the optimistic assumption.

The user action is a trigger of an existing function or additional function. Assume that we can uniquely determine each user action to be a trigger of an existing function or additional function. This corresponds to a case where an additional function is implemented with different user actions and messages from an existing function. In this case, under the optimistic assumption, the algorithm never detects any illegal actions. On the other hand, when there is a user action that is sometimes a trigger of an existing function and sometimes a trigger of a new function, the algorithm reports an illegal action.

The example of illegal action described in section 2 is caused by the user action "Cancel:". "Cancel:" is sometimes a trigger of an existing function and sometimes a trigger of an additional function. Thus, the algorithm can detect illegal actions even under the optimistic assumption.

### 5.2 Complexity of the Algorithm

In this subsection, the complexity of this algorithm is evaluated.

The parameters are the number of processes $N$, the number of MSCs $m$, and the maximum length of FSM $u$.

To detect illegal actions, we must verify systems E and E+A before using the above algorithm, because we must examine whether assumptions (A1) and (A2)

hold for the specification. To verify systems E and E+A, we adopt the method proposed in [3]. The order of complexity is $O(um^3N^3)$.

Next, we evaluate the order of complexity of the algorithm proposed in the previous subsection. The method in [3] can include the operations of steps 1 and 2 without increasing its complexity.

The order of complexity of steps 3 and 4 is (the maximum number of states of an FSM) × (the number of processes) × (the maximum number of events defined at a state) = $O(um \times N \times N)$.

The order of complexity of step 5 (algorithm for calculating QM) is $O(uN)$, because the maximum number of repetitions for checking an event is 2. Therefore, the order is the maximum number of events specified in an MSC. These operations are repeated m times in step 6 and N times in step 7, so the total order of this part is $O(umN^2)$.

When we compare ET(e) with QM(e,i) in step 6, we assume that QM(e,i) is given in the form of an array H and that the message sent by process $p_h$ is stored in the h-th entry of H. The maximum number of elements in ET(e) is m (i.e., the number of MSCs), so the order of complexity is $O(m)$. As in step 5, these operations are repeated m times in step 6 and N times in step 7, so the total order of this part is $O(m^2N)$.

As the result, the total order of the algorithm proposed in this paper is $O(um^2N^2)$ (i.e., $O(umN^2) + O(umN^2) + O(m^2N)$).

Next, the order of complexity using only the conventional method [3] is evaluated. If we can assume that a particular K processes of the system have the possibility of having new functions added, the number of different E+(A) systems is $2^K$. So we must apply the method of [3] to each system. Then the order of complexity is $O(um^3N^32^K)$, where $2^K$ is a constant. But the method proposed in this paper guarantees that all processes of the system have this possibility. To guarantee the same situation using a method based on [3], the order of complexity would have to be $O(um^3N^32^N)$.

If we use the usual method, we must verify every pattern. In view of the order of complexity, this is impractical. On the other hand, the order of complexity of our proposed algorithm is polynomial, so it should be possible to apply this method to real systems.

## 6 Conclusions

This paper discussed a verification method for communication services on a network where functionally updated nodes and non-updated nodes coexist. The proposed method uses a polynomial verification algorithm which detects illegal actions caused by software version mismatches during communication. Thus, the problem discussed in the paper is very new and we believe it becomes an important problem in future communication networks.

The proposed algorithm is fundamental, and future work should include the following generalization of results: a resolution method for illegal actions and an algorithm extension for specifications including loops.

**Acknowledgments**

We would like to thank Dr. Haruhisa Ichikawa and Mr. Hiroaki Higaki for their valuable comments and suggestions.

## References

[1] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", Communications of ACM,21,7,pp.558-565 (1978).

[2] CCITT: "Draft Recommendation Z.120 Message Sequence Chart (MSC)" (1992).

[3] M. Itoh and H. Ichikawa, "Protocol Verification Algorithm Using Reduced Reachability Analysis" Trans. of IECE of Japan, vol. E66, No.2 pp. 88–93 (1983).

[4] D. Brand and P. Zafiropulo, "On Communicating Finite-State Machines", Journal of the ACM, Vol. 30, No. 2, April, pp. 323–342 (1983).

[5] F. J. Lin, P. M. Chu, and M. T. Liu, "Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies", ACM SIGCOMM (1987).

[6] Y. Kakuda, Y. Wakahara and M. Norigoe, "An Acyclic Expansion Algorithm for Fast Protocol Validation", IEEE Trans. on Software Engineering, Vol. 14, No. 8, pp. 1059–1070 (1988).

[7] H. Ichikawa, M. Itoh, J. Kato, A. Takura and M. Shibasaki, "SDE: Incremental Specification and Development of Communications Software", IEEE Trans. on Computers, Vol. 40, No. 4, pp. 553–561 (1991).

[8] J. Kramer and J. Magee, "Dynamic Configuration for Distributed Systems" IEEE Trans. on Software Engineering, Vol. 14, No. 4, pp. 424–436 (1985).

[9] H. Higaki, "Group Communications Algorithm for Dynamically Updating in Distributed Systems," Proc. of IPSJ DPS Workshop, pp. 81–90 (1993).