Group Protocol for Multimedia Objects in Distributed Object-based Systems

Youhei Timura, Katsuya Tanaka, and Makoto Takizawa
Tokyo Denki University
Email {timura, katsu, taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by cooperation of a group of objects. A state of an object depends on in what order request and response messages are delivered to the object. In this paper, we newly define a novel precedent relation of messages based on a conflicting relation among requests. In addition, multimedia objects are manipulated and transmitted in the group. Multimedia objects transmitted here to satisfy quality of service (QoS) i.e. maximum delay time and message loss ratio required by applications. We discuss causality of messages with respected to QoS.

1 Introduction

In distributed systems, a *group* of multiple processes are cooperating to achieve some objectives. Many papers [4,5,10–12] discuss how to support a group of processes with the causally / totally ordered delivery of messages at a network level. In addition, a message is required to be delivered to all the destinations of the message, i.e. *atomic* delivery. The group protocol implies $O(n^2)$ computation and communication overheads for the number n of the processes in the group. The overheads can be reduced if only messages required to be ordered by the applications have to be causally and atomically delivered.

Processes manipulate data like files in the computers. An application is composed of these processes, i.e. process-based application. On the other hand, an application is now being object-based like CORBA [15], i.e. data and methods, which are processes manipulating the data, are encapsulated in an object. An application sends a *request message* with a method *op* to an object *o* in order to invoke *op*. The method *op* is performed on the object *o* and a *response message* with the result of *op* is sent back to the sender of the request. There are synchronous, asynchronous, and one-way invocations depending on how the sender waits for the responses [15]. Request and response messages carry objects as parameters and results, respectively. In addition, *op* may further invoke other methods, i.e. *nested invocation*. In the group communication, a message is sent to all the destinations in a group. In the parallel invocation, multiple methods are invoked at a same time and the invoker waits for the responses. In the *and* wait, all the responses are required to be received. In the *or* wait, at least one response is required to be received. Thus, messages may not be required to be delivered to all the destinations. The result obtained by performing a pair of conflicting methods depends on the computation order of the methods. Hence, if a pair of conflicting methods op_1 and op_2 are issued to multiple objects, the request messages op_1 and op_2 have to be delivered to the objects in the same order. Thus, we define how messages to be delivered based on types of invocations and conflicting relations in the object-based system.

In distributed applications, multimedia objects are exchanged among objects. The multimedia objects transmitted in the request and response messages are required to satisfy some quality of service (QoS). Maximum delay time (Δ) and message loss ratio (ε) are kinds of quality of service (QoS) required at the network level. If it takes a longer time to deliver a multimedia object than Δ, it is meaningless to deliver the object to a multimedia application. We discuss how to deliver multimedia messages in a group of objects so as to satisfy Δ and ε.

In section 2, we discuss the object-based system. In section 3, we discuss the object-based ordered relation of messages. In section 4, we discuss QoS-based causality of messages.

2 Model of Object-based System

2.1 Object-based system

Objects are encapsulations of data and methods for manipulating the data. A transaction invokes a method on an object by sending a request to the object. The method is performed on the object and the response is sent back to the transaction. Here, the method may invoke other methods, i.e. *nested invocation*.

The objects are distributed in computers interconnected with asynchronous networks [Figure 2]. A *computer* means a collection of objects, which does not necessarily mean a physical *computer*. A database server is an example of a computer where objects are tables and records. A computer sends

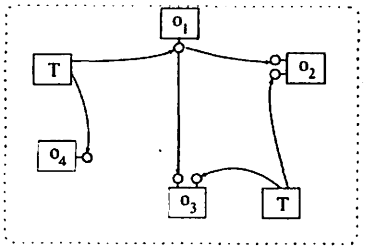

Figure 1: Group.

request and response messages issued by objects in the computer and receives messages issued to objects in the computer. In the traditional group protocols [5], the computers causally deliver messages independently of what kind of data is carried by messages. In this paper, we discuss what messages to be causally/totally delivered by taking into account types of messages exchanged among the objects and types of method invocations. For example, suppose a computer receives a pair of request *increment* and *decrement* messages m_1 and m_2 on a *counter* object. Since the state of the *counter* object obtained by performing *increment* and *decrement* is independent of the computation order, the computer can deliver m_1 and m_2 in any order even if m_1 and m_2 are causally ordered. A group G is composed of comuters supporting objects $o_1, \ldots, o_n$. A transaction invokes methods on an object only in the group G. A method on an object invokes only methods on objects in the group G [Figure 1]. Every method does not invokes any method which is not in the group G. The objects are cooperating with each other in the group G.

Multimedia objects like voice and video are transmitted among the objects. It is critical to discuss *quality of service* (QoS) supported by multimedia objects, e.g. maximum delay time, message loss ratio. Multimedia objects are required to be delivered so as to satisfy QoS. For example, it is meaningless to deliver multimedia objects if it takes a longer time to deliver them than a maximum delivery time Δ required by an application. In addition, a destination object may not require to receive all the messages decomposed from a multimedia object. Let ε be a maximum ratio of messages lost. Even if some messages are lost in the network, the destination object can take the multimedia object transmitted if the loss ratio is smaller than ε. In this paper, we discuss how to deliver multimedia objects in a specified delay time under constraint of the maximum delay time Δ and message loss ratio ε.

A group communication is composed of two sublayers, *object communication* and *transport* layers [Figure 2]. At the object layer, messages are ordered based on the object concept. At the transport layer, messages are delivered so as to satisfy Δ and ε constraints.

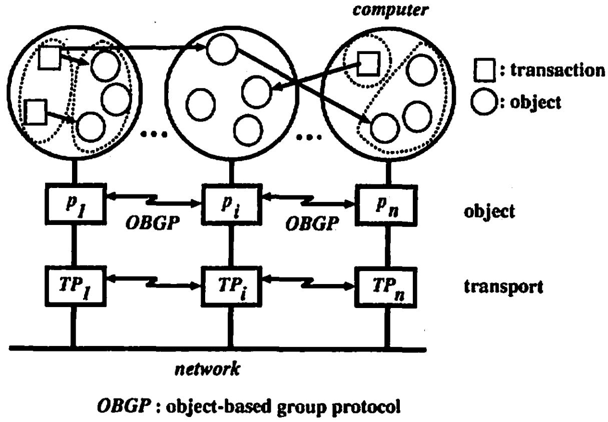

Figure 2: System model.

2.2 Invocation types

Methods are invoked in a nested manner in object-based systems. There are *synchronous*, *asynchronous*, and *one-way* invocations of a method *op* with respect to how an invoker, e.g. transaction T waits for the response of *op*. In the synchronous invocation, T waits for a response of *op*, i.e. a remote procedure call (RPC). In the asynchronous one, T is performed without blocking while eventually receiving the response of *op*. In the one-way invocation, T does not wait for the response of *op* after *op* is invoked. T and *op* are being independently performed.

There are *serial* and *parallel* invocations of multiple methods. In the serial invocation, at most one method is invoked at a time. On the other hand, multiple methods can be simultaneously invoked in the parallel invocation. Here, suppose op_1 and op_2 are synchronously invoked by a transaction T. T waits for the responses from op_1 and op_2. There are *and* and *or* ways to wait for the responses. In the *and* wait, T blocks until both of the responses are received. In the *or* wait, *op* starts to be performed only if at least one response is received in asynchronous and one-way invocations. In the *and* wait, the requests are required to be atomically delivered to op_1. On the other hand, at least one request is required to be delivered in the *or* wait.

2.3 Conflicting methods

Let op_1 and op_2 be a pair of methods supported by an object o. According to the traditional theories [4], op_1 *conflicts* with op_2 if the result obtained by performing op_1 and op_2 on the object o depends on the computation order of op_1 and op_2. Otherwise, op_1 is *compatible* with op_2. By using the locking mechanism [4], a pair of conflicting methods op_1 and op_2 are serially performed in traditional systems like database systems. For example, op_2 blocks while op_1 is being performed on the object o. If every object is locked according to the two-phase locking protocol [4], the computation of methods on every object is serializable.

Suppose that a computer p_t supports a *blackboard* object b with a *display* method. A computer p_s sends a request *display* d_s with a picture object m_s to the blackboard object b in p_t. Another computer p_u also sends a request *display* d_u with a picture object m_u to b. The pictures are displayed on the blackboard b in p_t. Suppose that areas where m_s and m_u are displayed on b are overlapped. Since m_s and m_u are large, it takes time to transmit *display* requests to objects and the response time is increased if m_s and m_n are serially delivered. Hence, after p_t starts delivering the request d from p_s, p_t starts delivering d_u from p_u. On the blackboard object b, the pictures are overwritten by the succeeding pictures. Here, a pair of the *display* methods are able to be concurrently performed on b but the state of b depends on which *display* method d_s or d_u is started to be performed earlier than the other. Thus, some pair of conflicting methods can be concurrently performed while it is critical to consider which method is started and ended earlier than the other.

A pair of methods op_1 and op_2 on an object o are related with respect to the following points:

1. op_1 and op_2 cannot be concurrently performed, i.e. *mutually exclusive*.
2. op_1 and op_2 can be concurrently performed.
   a. op_1 and op_2 can be started in any order.
   b. it is critical to consider which method op_1 or op_2 is started and ended before the other.

Now, we define new types of conflicting and compatible relations as follows:

[Definition] Let op_1 and op_2 be a pair of methods supported by an object o.

1. op_1 *conflicts* with op_2 iff the result obtained by performing op_1 and op_2 on the object o depends on the computation order of op_1 and op_2. Otherwise, op_1 is *compatible* with op_2.
2. op_1 *strongly conflicts* with op_2 iff op_1 conflicts with op_2 and op_1 is mutually exclusive with op_2.
3. op_1 *weakly conflicts* with op_2 iff op_1 conflicts with op_2 and op_1 is not mutually exclusive with op_2.
4. op_1 is *strongly compatible* with op_1 iff op_1 is compatible with op_2 and op_1 is not mutually exclusive with op_2.
5. op_1 is *weakly compatible* with op_2 iff op_1 is compatible with op_2 and op_1 is mutually exclusive with op_2. □

For example, the method *increment* is weakly compatible with the method *decrement* on the *counter* object because the methods cannot be concurrently performed. A pair of *show* methods are strongly compatible on the *counter* object c. A pair of *display* method weakly conflict on the *blackboard* object b. We assume every type of conflicting relation is symmetric but not transitive.

We define a significantly precedent relation among methods performed in p_t.

- op_1 *significantly precedes* op_2 ($op_1 \Rightarrow op_2$) iff op_1 conflicts with op_2 and op_1 is started before op_2.

op_1 and op_2 are significantly concurrent ($op_1 \parallel op_2$) if neither $op_1 \Rightarrow op_2$ nor $op_2 \Rightarrow op_1$.

3 Delivery of Messages in Objects

3.1 Ordered delivery

In the object-based system, *request* and *response* messages are exchanged among the computers. A message m_1 *causally precedes* another message m_2 if the sending event of m_1 *happens before* the sending event of m_2 [5, 8]. A message m_1 *totally precedes* another message m_2 iff every pair of common destinations of m_1 and m_2 deliver m_1 and m_2 in the same order. In addition, m_1 totally precedes m_2 if m_1 causally precedes m_2. Suppose a computer p_s sends a message m_1 to a pair of computers p_t and p_u, and p_t sends m_2 to p_u after receiving m_1. Since m_1 causally precedes m_2, p_u has to receive m_1 before m_2 according to the traditional causality theory. For example, suppose a computer p_s sends a request m_1 to other computers p_t and p_u. The method m_1 is performed on objects p_t and p_u. In the computer p_t, suppose a method m_3 sends a request m_2 to p_u. If m_1 and m_3 are compatible, the computer p_u can deliver m_1 and m_2 in any order. However, the computer p_u is required to deliver m_1 before m_2 if m_1 conflicts with m_3. Next, suppose p_s sends a message m_1 to p_t and p_u and p_v sends m_2 to p_t and p_k. In the totally precedent relation, m_1 and m_2 are delivered to p_t and p_u in a same order. If m_1 and m_2 are conflicting requests on objects in p_t and p_u, m_1 and m_2 are required to be delivered in the same order. Otherwise, m_1 and m_2 can be delivered in any order. Thus, applications do not require all the messages transmitted in the network be causally and totally delivered.

We define a *significantly precedent relation* "$\rightarrow$" among a pair of messages m_1 and m_2.

"$m_1 \rightarrow m_2$" is meaningful for object-based applications.

[Definition] A message m_1 *significantly precedes* another message m_2 ($m_1 \rightarrow m_2$) iff one of the following conditions holds:

1. a same method instance sends m_1 before m_2.
2. Let op_1 and op_2 be method instances which sends messages m_1 and m_2, respectively. op_1 significantly precedes op_2 ($op_1 \Rightarrow op_2$).
3. a same instance receives m_1 before m_2.
4. Let op_1 and op_2 be instances which receive m_1 and send m_2, respectively. $op_1 \Rightarrow op_2$.
5. $m_1 \rightarrow m_3 \rightarrow m_2$ for some message m_3. □

[Theorem 1] A message m_1 causally precedes another message m_2 if $m_1 \rightarrow m_2$. □

Suppose an instance op_1 of an object o_i invokes a method op_{11} on a replica o_j in a computer p_t and o_k in p_u, respectively. Suppose op_2 of o_l invokes op_{21} on o_j and o_k. If op_{11} strongly conflicts with op_{21}, the methods from op_1 and op_2 are required to be delivered to o_j and o_k in the same order. This is the serializability [4]. In addition to the significant precedency of messages, some messages are required to be totally preceded in the object-based system.

[Definition] A message m_1 *object-based precedes* (*OB-precedes*) another message m_2 ($m_1 \preceq m_2$) iff

1. if m_1 significantly precedes m_2 ($m_1 \rightarrow m_2$),
   - m_1 and m_2 are conflicting requests, or
   - m_1 or m_2 is not a request.
2. if $m_1 || m_2$, m_1 and m_2 are conflicting requests and $m_1 \preceq m_2$ in every other common destination of m_1 and m_2.□

A distributed system supports the *object-based ordered* (OBO) delivery of messages iff every message m_1 is delivered before m_2 in every common destination of m_1 and m_2 if $m_1 \preceq m_2$.

[Theorem 2] A message m_1 totally precedes another message m_2 if $m_1 \preceq m_2$. □

In the OBO delivery, only messages to be ordered in the object-based system are delivered in the OB-precedent order $\preceq$. On the other hand, every message transmitted in the network is delivered in the causally / totally precedent order. Hence, a message m can be delivered without waiting for every message causally preceding m. The delay time of each message can be reduced.

Figure 3 shows three computers p_1, p_2, and p_3 exchanging messages m_1, m_2 and m_3. According to the traditional causality theory, m_1 causally precedes m_2 and m_2 causally precedes m_3. The computer p_3 is required to deliver m_1, m_2, and m_3 in this order. A method instance op_1 in the computer p_1 issues a message m_1 to p_2 and p_3. Here, method instances op_3 and op_4 are invoked

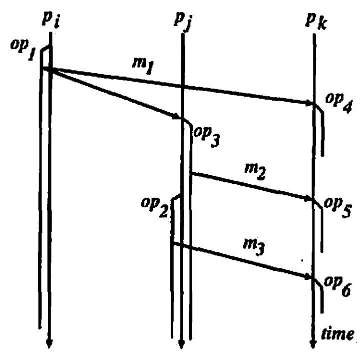

Figure 3: Message ordering.

in the computer p_2 and p_3, respectively. Then, op_3 invokes op_5 by sending a request m_2 to p_3. Another instance op_2 in p_2 invoke op_2 in p_3. Here, m_1 significantly precedes m_2 ($m_1 \rightarrow m_2$), i.e. $m_1 \preceq m_2$. If op_2 and op_3 are strongly compatible, m_3 is independent of m_1 and m_2. Hence, $m_1 \parallel m_3$. If op_2 and op_3 are compatible, $m_1 \parallel m_3$ Suppose that op_4 is invoked by m_1 and op_5 is invoked by m_2. If op_4 and op_5 conflict, m_1 is required to be delivered before m_2, i.e. $m_1 \rightarrow m_2$. Otherwise, $m_1 \parallel m_2$. This example shows that $m_1 \not\rightarrow m_2$ even if m_1 cusally precedes m_2.

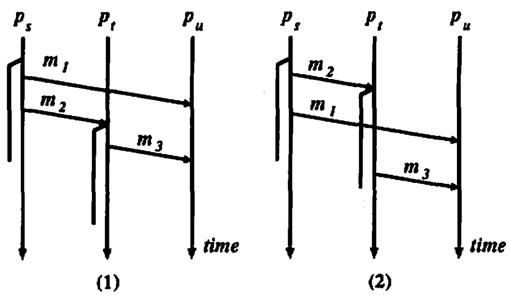

Figure 4: Message ordering.

In Figure 4, three computers p_s, p_t, and p_u are exchanging messages m_1, m_2, and m_3. According to the traditional causality theory, m_1 causally precedes m_3 because m_1 causally precedes m_2 in (1). However, m_1 and m_3 are causally concurrent while m_1 causally precedes m_2 in (2). Depending on the implementation, a message may be required to be serially sent to multiple destinations in order to multicast the message. Here, m_1 and m_2 should have been sent at a same time and m_1 and m_2 causally precede m_3. It depends on the sending order of m_1 and m_2 whether or not m_1 causally precedes m_3. This example shows m_1 does not causally precede m_2 even if m_1 causally precede m_2 not application level. Here, suppose that a same object o_s sends m_1 and m_2. According to the definition of the significant precedent relation, $m_1 \rightarrow m_3$ since m_1 and m_2 are sent by the same object and $m_2 \rightarrow m_3$. Thus, Theorem 1 holds if a message to be multicast is sent at a time. In the OBO delivery, messages to be causally ordered from the application point of view can be causally ordered even if messages cannot be sent at a time. Here, suppose that an object o in a computer p_t sends a pair of messages m_1 and m_2. If the object o does not receive any message after sending m_1 before m_2, m_1 and m_2 are referred to as *in a same transmission* in p_t. This same transmission relation is transitive. In the OBO delivery, If m_1 and m_2 are in a same transmission, m_1 and m_2 are considered to be transmitted at a time. If m_1 and m_2 are sent by methods which are compatible or on different objects in the computer p_s, m_1 does not causally precede m_3 in Figure 4(2).

3.2 Atomic delivery

Suppose multiple methods $op_1, \ldots, op_n$ are invoked on objects $o_1 \ldots, o_n$, respectively, by a method op on an object o in parallel. The request messages of $op_1, \ldots, op_n$ are required to be delivered to all the objects $o_1, \ldots, o_n$. In the *and* wait, the object o has to wait for all the responses. That is, the request message is required to be atomically delivered. For example, if o_i fails to receive op_i, op has to retransmit op_i to o_i. On the other hand, in the *or*-wait, op does not wait for all the responses. If op receives at least one response, op finishes the invocation of $op_1, \ldots, op_n$. Even if some object o_i faults to receive a request op_i, op does not retransmit op_i to o_i. Hence, the atomic delivery is not required to be supported in the or-wait parallel invocation.

[**Object-based delivery**]Let m be a request message op to multiple objects.

- If op is invoked in the *and* wait, m is required to be delivered to all the objects.
- If op is invoked in the *or* wait, m is required to be delivered to at least one destination. □

4 QoS-Causalities

In realtime multimedia applications, messages have to be delivered to the destinations by some deadline Δ specified for the messages. It is meaningless to deliver a message after the deadline Δ. Thus, a computer p_j has to receive a message m in Δ time units after p_i sends m [1, 3, 14]. Here, let $ts(m)$ be time when m is sent. Let $tr_i(m)$ be time when p_i receives m. Suppose that p_i sends a message m to p_j. m is referred to as *received in* Δ by p_j iff $ts(m) + \Delta \geq tr_j(m)$. The causality based on Δ [1] is defined as follows.

[**Δ-causality**] A message m_1 *Δ-causally precedes* another message m_2 iff m_1 causally precedes m_2 and $ts(m_1) + \Delta \geq ts(m_2)$. □

In the Δ-causality, the delay time between every pair of objects is assumed to be the same. However, delay time and message loss ratio are different for every pairs of computers. The maximum delay time Δ_{ij} and maximum loss ratio E_{ij} are specified for every pair of o_i and o_j by the application. Δ_{ij} can be obtained based on the statistics of delay time and message loss ratio between p_i and p_j. Here, let Δ^* be a set $\{ \Delta_{ij} \mid i, j = 1, \ldots, n \}$ of the delay requirements.

[**Δ^*-causality**] [12] Let m_1 and m_2 be messages sent by computers p_i and p_j, respectively. m_1 *Δ^*-causally precedes* m_2 ($m_1 \xrightarrow{\Delta^*} m_2$) iff m_1 causally precedes m_2 and $ts(m_1) + \Delta_{ij} \geq ts(m_2)$. □
That is, m_2 is sent in Δ_{ij} time units after m_1 is sent while $m_1 \rightarrow m_2$.

In Figure 5, a computer p_1 sends a message m_1 to p_2 and p_3, and p_2 sends m_2 to p_3 after receiving m_1. Suppose $m_1 \rightarrow m_2$. Since p_3 receives m_2 in Δ_{32}, p_3 delivers m_2. Then, p_3 receives m_1. Since p_3 receives m_1 in Δ_{31}, p_3 can deliver m_1. However, since m_1 is already delivered and $m_1 \xrightarrow{\Delta^*} m_2$, p_3 cannot deliver m_1. If m_1 is delivered, m_2 cannot be delivered because m_2 is obligated to be delivered after $ts(m_2) + \Delta_{32}$. There is inconsistency among Δ_{12} and Δ_{23}. This example shows that p_i may not deliver m even if m is received in Δ_{ij}. Thus, Δ^* may be inconsistent if each Δ_{ij} is independently decided. The Δ^*-causally precedent relation $\xrightarrow{\Delta^*}$ is *consistent* iff $ts(m_1) + \Delta_{ki} \leq ts(m_2) + \Delta_{kj}$ and m_1 causally precedes m_2 for every pair of messages m_1 and m_2 sent by objects o_i and o_j, respectively. The paper [11, 12] discusses how to decide consistent Δ^*.

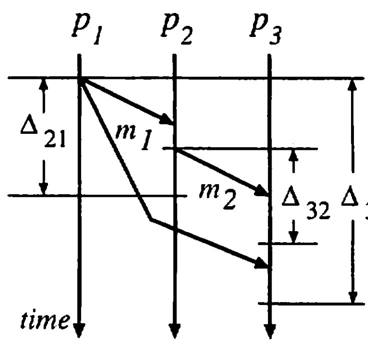

Figure 5: Δ^*-causality.

Due to congestions and network faults, some part of a message may be lost in the network. Let $Q_{ij}(m)$ show a loss ratio of a message m between a pair of computer p_i and p_j. The causality based on the loss ratio is defined as follows:

[**ϵ^*-causality**] A message m_1 *ϵ^*-causality precedes*

m_2 iff m_1 causally precedes m_2 and $Q_{ij}(m_1) \leq \epsilon_{ij}$, $Q_{jk}(m_2) \leq \epsilon_{jk}$, and $Q_{ik}(m_1) \leq \epsilon_{ik}$. □

5 Concluding Remarks

In this paper, we discussed how to support the object-based ordered (OBO) and $\Delta^*\varepsilon^*$ delivery of messages. While all messages transmitted in a network are causally or totally ordered in most group protocols, only messages to be causally ordered at the application level are ordered to reduce the delay time. Based on the conflicting relation among methods, we defined the object-based (OB) precedent relation among request and response messages.

References

[1] Adelstein, F. and Singhal, M., "Real-Time Causal Message Ordering in Multimedia Systems," *Proc. of IEEE ICDCS-15*, 1995, pp.36–43.

[2] Ahamad, M., Raynal, M., and Thia-Kime, G., "An Adaptive Protocol for Implementing Causally Consistent Distributed Services," *of IEEE ICDCS-18*, 1998, pp.86–93.

[3] Baldoni, R., Mostefaoui, A., and Raynal, M., "Efficient Causally Ordered Communications for Multimedia Real-Time Applications," *Proc. of IEEE HPDC-4*, 1995, pp.140–147.

[4] Bernstein, P. A., Hadzilacos, V., Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[5] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM Trans. on Computer Systems*, Vol.9, No.3, 1991, pp.272-314.

[6] Enokido, T., Higaki, H., and Takizawa, M., "Group Protocol for Distributed Replicated Objects," *Proc. of ICPP'98*, 1998, pp.570-577.

[7] Enokido, T., Higaki, H., and Takizawa, M., "Object-Based Ordered Delivery of Messages in Object-Based Systems," *Proc of ICPP'99*, 1999, pp.380–387.

[8] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *CACM*, Vol.21, No.7, 1978, pp.558–565.

[9] Mattern, F., "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms* (Cosnard, M. and , P. eds.), *North-Holland*, 1989, pp.215–226.

[10] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of IEEE ICDCS-14*, 1994, pp.48–55.

[11] Tachikawa, T., Higaki, H., and Takizawa, M., "Significantly Ordered Delivery of Messages in Group Communication," *Computer Communications Journal*, Vol. 20, No.9, 1997, pp. 724–731.

[12] Tachikawa, T., Higaki, H., and Takizawa, M., "Group Communication Protocol for Real-time Applications," *Proc. of IEEE ICDCS-18*, 1998, pp.40–47.

[13] Timura, Y., Tanaka, K., and Takizawa, M., "Group Protocol for Supporting Object-based Ordered Delivery," *Proc. of IEEE ICDCS-2000 Workshop*, 2000, pp.C-7–C-14.

[14] Yavatkar, R., "MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications," *Proc. of IEEE ICDCS-12*, 1992, pp.606–613.

[15] Object Management Group Inc., "The Common Object Request Broker : Architecture and Specification," Rev.2.1,1997.