Consistency Mechanism for Cooperative Work in Distributed 3D Virtual Spaces

Eisuke Itoh
itou@cc.kyushu-u.ac.jp
Computing and Communications Center,

Tomohiko Kawano
kawano@csce.kyushu-u.ac.jp
Dept.of Computer Science and Communication Engineering,
Kyushu University.

Yoshihiro Okada
okada@i.kyushu-u.ac.jp
Dept. of Informatics,

Hakozaki 6-10-1, Higashi-ku, Fukuoka, 812-8581, JAPAN.

Abstract— **Currently the authors are studying on CSCW in distributed 3D virtual spaces. A problem is how to keep the same state among several computers for consistency. A logical clock idea seems available to solve this problem because it can perform user operation events in order according to their happen times. The authors introduced the logical clock idea into IntelligentBox, which is a constructive software development system for interactive 3D graphics applications. Furthermore, the authors evaluated its performances. This paper describes the consistency mechanism and its application examples, and also presents its performance evaluations.**

Keywords— **IntelligentBox, CSCW(Computer Supported Cooperative Work), 3D Application, Distributed Environment, Consistency mechanism.**

I. Introduction

We have been studying on CSCW(Computer Supported Cooperative Work). Various CSCW systems have been studied and developed, for example, TV conference systems, remote lecture systems, etc. Recent advances in computer and Internet technologies make it possible to develop distributed virtual reality applications. Currently, we are interested in CSCW in distributed 3D virtual environments. In this paper, we presents a consistency mechanism for cooperative work in distributed 3D virtual environments and its performance evaluations.

Okada and Tanaka proposed a IntelligentBox system[1][2][3][4]. It is a user-friendly rapid prototyping software development system for interactive 3D graphics applications. It represents objects as reactive 3D visual objects called "boxes". A *box* can be manually combined with other *boxes* by the dynamic construction mechanism. With this mechanism, users can develop application systems through direct manipulations on a computer screen without writing any text-based programs.

The IntelligentBox system also provides collaborative environments. This is realized by a functionality of a particular *box* called RoomBox. A RoomBox manages user-operation events. Multiple RoomBoxes on different host share user-operation events with each other so that the multiple RoomBoxes virtually provide a shared 3D space.

The well-known problem of distributed applications is how to keep the same state among several computers for consistency. To keep consistency needs to realize mutual exclusion mechanisms and ordering mechanisms of distributed events.

A simple and traditional but practical solution is to use the server-client model. The centralized server process (the daemon process, in other words) manages all states of each clients, and each clients follow the server. However, this server-client model needs a high performance computer as a server, and a network traffic and a maintenance of the server becomes serious problem.

We selected the distributed model, i.e., each host communicates to each other by a point-to-point connection. The logical clock idea[5] is available for the distributed model because it can execute user-operation events in order according to their happen times. Each host communicates to each other and coordinates each state using the logical clock. We introduced a logical clock idea into the IntelligentBox system and furthermore evaluated its performances. This paper describes the detail of the consistency mechanism based on the logical clock, and presents its performance evaluations.

The remainder of this paper is organized as follows. Section II describes CSCW systems and distributed 3D virtual environment applications. In section III, we describe our consistency mechanism using the logical clock. Section IV shows an implementation of the mechanism in the IntelligentBox system, and also shows examinations. Section V discuss the performance. Finally, we conclude this paper in Section VI.

II. Cooperative Work in Distributed 3D Virtual Spaces

A. CSCW Tools

There has been a lot of researches and systems on Computer Supported Cooperative Work, for example, TV conference systems[6][7], group editors[8][9], task coordination systems[10]. TV conference systems are divided into two types, i.e., point-to-point communication TV conference systems and multi-attendants TV conference systems. The latters employ a server-client model. A main requirement of TV conference systems is a real-time communication facility for video and audio data on the Internet. The related systems includes DOLPHIN[11], CLEARBOARD[12], Conversation Board[13], DisEdit[14],

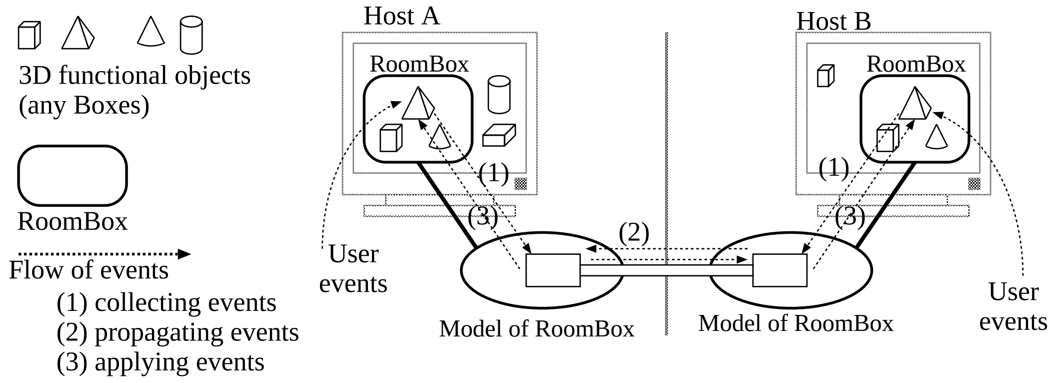

Fig. 1. The Concept of “RoomBox”

Argo[15], and so on. DOLPHIN is a remote meeting system that provides collaborative pen-based workspaces for window-based meeting. CLEARBOARD is an interpersonal design system based on video cameras. Conversation Board is a shared window-based conversation system using RENDEZVOUS[16].

Above application systems provide multiple users with 2D, not 3D, collaboration environments. Our research interest focuses on CSCW systems in distributed 3D virtual environments. Then we use the IntelligentBox system as our research system. Next subsection briefly introduce the IntelligentBox system.

B. The IntelligentBox system

IntelligentBox is a constructive visual software development system for interactive 3D graphics applications. IntelligentBox provides 3D reactive objects called *boxes*. Each *box* has a 3D visible shape and a unique function. Indeed each *box* consists of a model and a display object. A model holds state variables of a *box*. They are called “slots.” A display object defines how the *box* appears on a computer screen and defines how the *box* reacts to user operations. IntelligentBox also provides a dynamic data linkage mechanism called a slot-connection for combining functions of two *boxes*. By this mechanism, users can develop 3D graphics applications through direct manipulations on a computer screen without writing any text-based programs. This is a main feature of the IntelligentBox system.

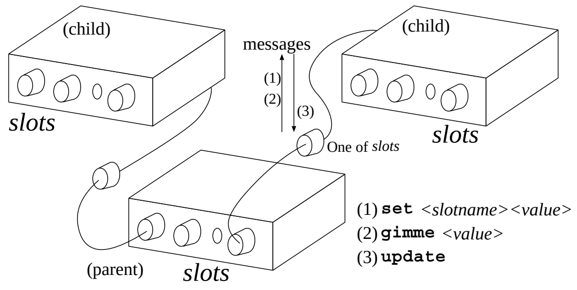

Fig. 2. Standard messages between *boxes*

Fig.2 shows a data linkage concept by slot-connections. Actually a slot-connection is realized by a message sending protocol consisting of three standard messages, i.e., a `set` message, a `gimme` message, and an `update` message. By these three messages, states of two *boxes* are exchanged and their two functions are combined.

As previously mentioned, the IntelligentBox also provides collaborative environments by a functionality of a particular *box* called a RoomBox. Fig.1 shows a concept of sharing user-operation events using two RoomBoxes existing separate hosts.

As shown in Fig.1, the RoomBox has a slot named ’event’ which holds a current user-operation event operated on its descendant *boxes*. Some specific user-operation events generated in a RoomBox are always stored in this slot until the next event is generated. As shown in Fig1, IntelligentBox provides a distributed model sharing mechanism. It enables multiple, distributed RoomBoxes to virtually share a common model with each other through messages passed via a network. By the distributed model sharing mechanism, multiple RoomBoxes can share user-operation events with each other. Here, descendant *boxes* of a RoomBox are treated as collaboratively operable 3D objects.

The structure shown in Fig.1 can be built only by making a copy of a RoomBox on Host A and transferring it to Host B. Therefore, using RoomBoxes, users can develop collaborative 3D virtual environments easily and rapidly.

Fig.3 and 4 show screen images of a Tank Battle game developed by IntelligentBox. Two players using a different host play the game simultaneously. This tank battle game is developed easily and rapidly by using IntelligentBox.

IntelligentBox provides good collaboration environment, but it has a problem for consistency. If multiple users operate a *box* at the same time (in a short period) and the operations conflict with others, then the states of each host becomes inconsistent.

III. Consistency Mechanism

The well-known problem of cooperative systems in distributed environments is how to keep the same state among several hosts for consistency. To keep consistency of distributed systems, mutual exclusion mechanisms and ordering mechanisms of distributed events have been studied and

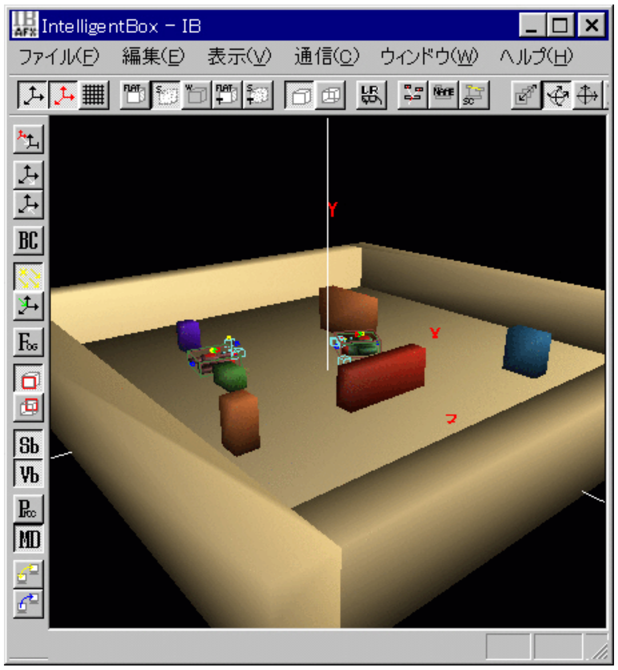

Fig. 3. A Tank Battle Game (1)

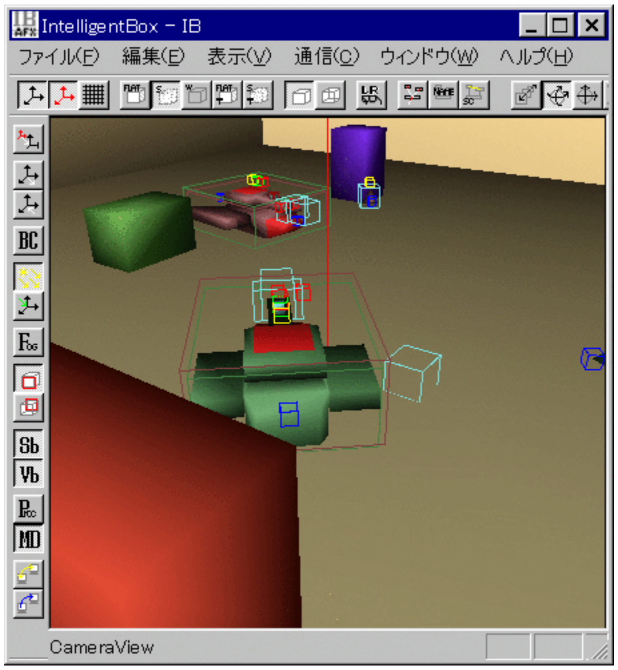

Fig. 4. A Tank Battle Game(2)

developed.

Latency exists in the communication between two hosts, and it may cause inconsistency. For example, let consider a distributed real-time interactive application like a car race game, multiple users play it simultaneously. In the case that a user's operation event is generated at the almost same time as another user's operation event, if the two events are executed in the different order on each host, the states of two hosts will be different.

To keep consistency of shared resources in a distributed environment, there are some solutions, i.e., a centralized model, a token ring model and a distributed model[18].

Centralized model

A simple but practical approach to keep consistency is to use the server-client model. This model assumes that a centralized server process (the daemon process, in other words) works as the manager or (fearless) leader. It keeps the important information and makes all decisions, and all clients obey the sever.

Here is how a client would operate a shared resource R.

1. Sends a request to the server
2. Waits for a reply from the server
3. Uses a resource R
4. Sends a release notice to the server

This means that there are three redundant messages per one resource request.

In the case that there is not any event buffer, if the server process receives multiple requests at the same time, the server selects only one client process, sends "OK" message to it, and sends "rejection" messages to the others.

In this model, tolerance of a distributed system depends on the performance of the server. If the load of the server is over its limit, the total performance will decrease. Furthermore it is obvious that if the server fails, all client processes will not work any more. These are defeats of the server-client model.

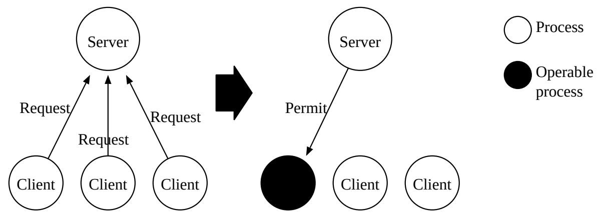

Fig. 5. The client-server model

Token ring model

This model assumes that there is a communication topology among the processes, which is a ring. Although a process P_i communicates directly to its two neighboring processes P_{i-1} and P_{i+1}, a token reaches indirectly to any process through sequential operations of reading a token from a left process and writing the token to a right process as shown in Fig.6. Only one process holds a token at a certain time. This process has ownership. This means that only this process operate a shared resource.

A mutual exclusion mechanism is realized by the following way.

1. Connects all processes in a logical ring. The ring is called as a token ring.
2. The token hops and circulates all processes along with the ring.
3. Only the process which holds the token is able to operate shared resources.

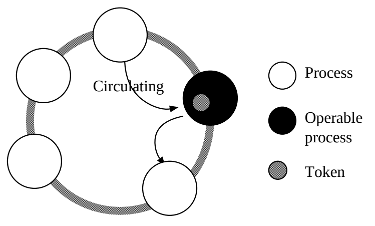

Fig. 6. Toke ring model

This is a good way of enforcing mutual exclusion in a fair way, however, there are problems. It is necessary to construct the token ring before the beginning of collaboration. If any one process may fail, the ring must be reconstructed. If a token may be lost, any mechanism must be prepared to decide a process that creates a new token.

Distributed model

Let consider the number of process is n, they possibly work at different hosts. When each process needs the shared resource, it sends a request to the other (n-1) processes; when it receives (n-1) replies, it comes to use the resource. When the process finishes using the resource, it sends a release message to all the processes that are waiting for the resource.

This model assumes a reliable mechanism for delivering messages. Each use of the critical shared resource, requires $2 * (n - 1)$ messages.

Here is the Ricart-Agrawala algorithm to realize mutual exclusion[17].

1. Before operate to a shared resource, each process sends "request" message to all other processes. The request message contains the shared resource name, process number and current time.
2. When a process receives the request message, it replies as follows:

- If not in use and does not want, reply OK.
- If in use, queues request (no reply).
- If wants to use, compare timestamp in request to its own. If lower sends OK, otherwise queue.
- Requesting process waits until everyone sends it an OK message.
- When exiting dequeues all processes and sends OK.

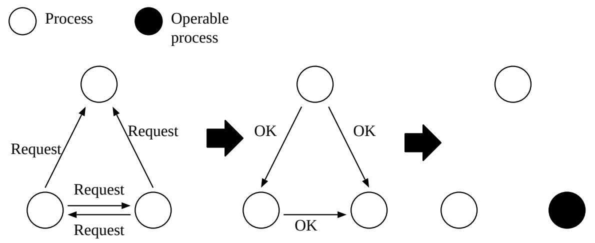

Fig. 7. The decentralized model

The performance of this model is not so high but it is robust. This model can also handle dynamic join and leaving of a group. The problem of this approach is how to decide which request message is the fastest among others, in other words, all events in the distributed hosts must be totally ordered.

It can't rely on the physical clock time on each host. They don't show an accurate time and it is impossible to synchronize all physical clocks. Lamport proposed the logical clock idea for this problem in [5]. We used the logical clock idea to solve the problem.

The logical clock

Lamport proposed logical clock in [5] to order the events occurring in a distributed system. Certainly if an event A happens before an even B, then A cannot have been caused by B. In this situation we cannot say that A caused B, but we cannot exclude that it might have influenced it. We want to characterize this "happened-before" relation on events. We will consider only two kinds of events, the sending of a message and the receiving of a message.

1. If events e_1 and e_2 occur in the same system and e1 occurs before e_2 (there is no problem to determine this in a single system) then e_1 happened-before e_2, written $e_1 \rightarrow e_2$.
2. If event e_1 is the sending of a message and e_2 is the receiving of that message, then e_2 happened-before e_2.
3. If $e_1 \rightarrow e_2$ $e_2 \rightarrow e_3$ then $e_1 \rightarrow e_3$.

The $\rightarrow$ relation is a partial order. Given events e_1 and e_2 it is not true that either they are the same or one happened-before the other. Events that are not ordered by happened-before are said to be concurrent.

This happened-before relation is unsatisfactory since, given two events, it is not immediate to determine if one happened before the other or if they are concurrent.

We could like a clock C that applied to an event returns a number so that the following holds:

If $e_1 \rightarrow e_2$ then $C(e_1) < C(e_2)$.

He defined one such C, a logical clock, as follows:

1. On each process P_i starts with a clock L_i set at 0.
2. If e is the sending of a message, then increment the local clock L_i, set $C(e)$ to L_i and timestamp the message with L_i.
3. If e is the receiving at P_i of a message with timestamp t, then set the local clock and $C(e)$ to $max\{t, Li + 1\}$.

Fig.8 is an example showing logical clocks and mutual exclusion of distributed model. A, B, C are processes and the number n in "[ ]" shows local logical clock time. In this case, A and C want to operate a shared resource. Their request messages and OK replies are arrived as shown in lower part of Fig.8. Each process acts according to the Ricart-Agrawala algorithm. Finally, process C gets the first priority for operation to the shared resource.

IV. Implementation and Examination

A. Implementation

As mentioned in section II-B, the IntelligentBox system didn't have enough consistency mechanism. The IntelligentBox system provides a virtual space shared by multiple hosts using RoomBoxes. Multiple users can operate

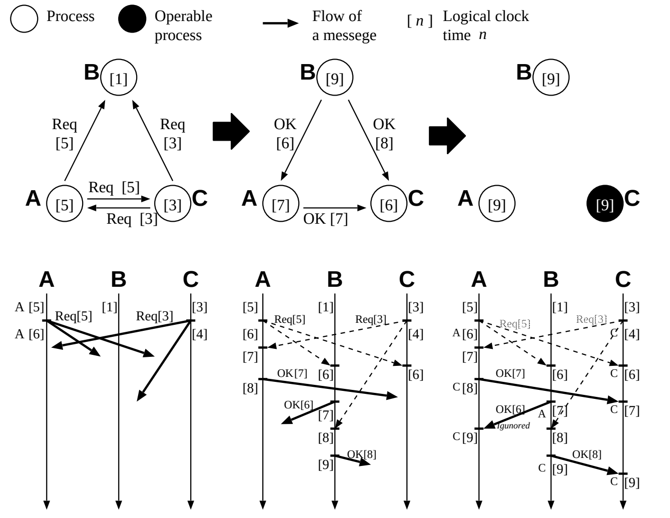

Fig. 8. An example showing logical clocks and mutual exclusion of the distributed model

Boxes in the 3D virtual space. But there may be inconsistent states, if more than one user operates a *box* simultaneously. For consistency, we have introduced an object lock mechanism to realize mutual exclusion.

We chose the distributed model for this implementation, because the distributed model is robust and it doesn't need any server. Although the communication cost of the distributed model is higher than both the sever-client model and the token ring model, the number of hosts (users) we assume will not so much, at most about 10 hosts. Then the communication cost will not be so much.

Fig.9 shows the state transition of each process. To lock a *box* in a 3D virtual space, each process behaves according to this state transition.

B. Examinations

We had three examinations with respect to three types of network topologies. Fig.10, 11 and 12 show the network topologies. Here is the characteristics of the topologies.

- Exam.1 : 3 hosts on the same subnet.
- Exam.2 : 3 hosts on 2 subnets.
- Exam.2 : 4 hosts on 2 subnets.

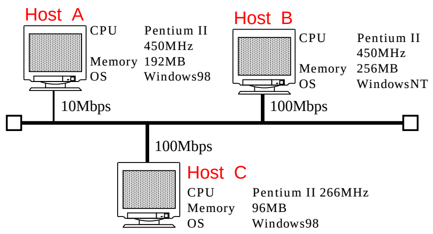

Fig. 10. Network topology of exam.1

We used a simple application shown in Fig.13 for examinations of realtime distributed collaborations. The *box* shown in Fig.13 consists of three primitive *boxes*, two cubic *boxes* and one cylindrical *box*.

Operations for realtime distributed collaboration in these examinations are very simple, mouse click and drag. When

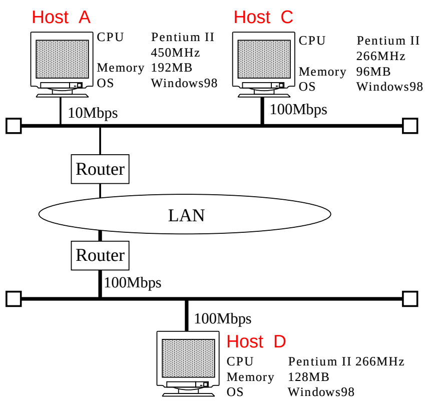

Fig. 11. Network topology of exam.2

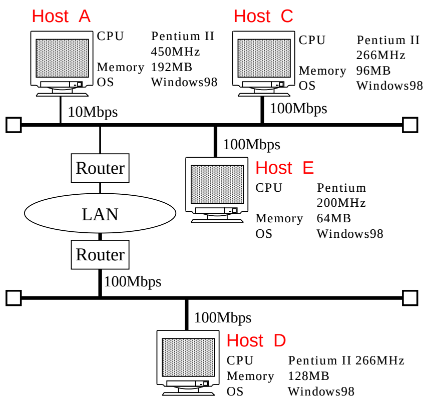

Fig. 12. Network topology of exam.3

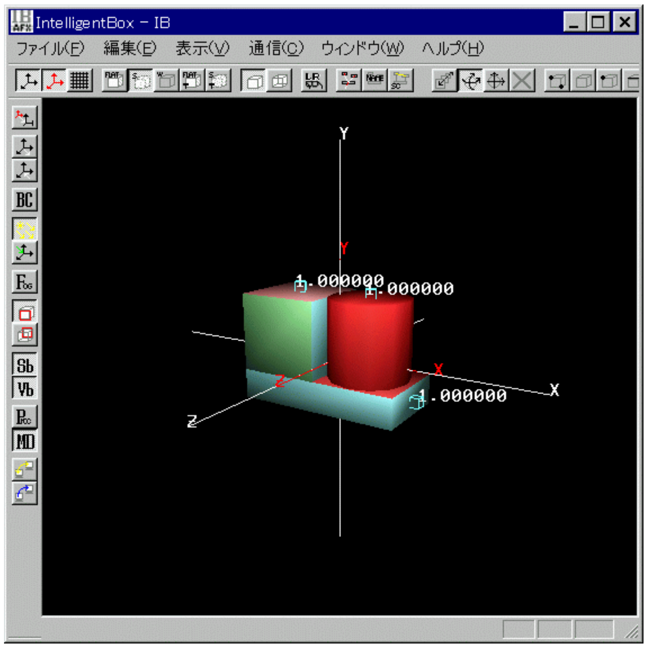

Fig. 13. A Screen image of the application used examinations

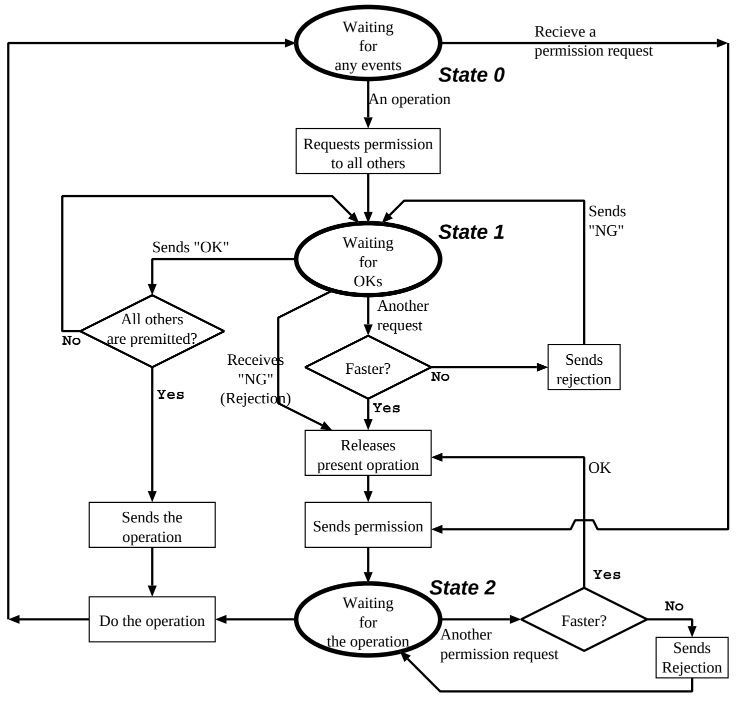

Fig. 9. The state transition

a user clicks on a *box*, if the click is the fastest in other clicks, he/she gets ownership of the Box. After that, the user can operate (move, rotate, etc..) the *box* until he/she releases it.

C. Results

We examined the implementation in three cases. 3 or 4 users used an application (shown in Fig.13) in different network environments. Operation records were stored in log files.

Table I, II, III, IV were analysis results of log files. Table I describes latency times from sending "Requests" until receiving "OKs". Table II, III, IV describe response times of operations.

TABLE I
LATENCY TIME FROM "REQUEST" UNTIL "OK"

	Exam.1 (msec)	Exam.2 (msec)	Exam.3 (msec)
The mean	237	445	503
Minimum	99	165	235
Maximum	360	907	967

TABLE II
RESPONSE TIME FOR OPERATION TO *boxes* (IN EXAM.1)

	B		C	
A	Mean	162	Mean	214
	Min	99	Min	97
	Max	242	Max	360
B			Mean	296
			Min	267
			Max	330

(msec)

TABLE III
RESPONSE TIME FOR OPERATION TO *boxes* (IN EXAM.2)

	C		D	
A	Mean	204	Mean	388
	Min	32	Min	264
	Max	357	Max	907
C			Mean	306
			Min	144
			Max	829

(msec)

V. Discussion

A. Latency time for obtaining ownership

From Table I, while the mean latency time of the examination 2 is close to the examination 3, the mean latency time of the examination 1 is almost half of the examination 2 or 3. Therefore, it can be said that the elapsed time for message transfer between two different subnets occupied the major part of the latency time for obtaining ownership.

B. Granularity of events

The second row of Table II shows that the response time of the host C concerning the host A is smaller than concerning the host B, and the first column of Table III shows that the response time of the host A concerning the host

TABLE IV

Response Time for operation to *boxes* (in exam.3)

	C		D		E	
A	Mean	269	Mean	273	Mean	384
	Min	231	Min	229	Min	110
	Max	299	Max	319	Max	967
C			Mean	280	Mean	420
			Min	238	Min	305
			Max	314	Max	535
D					Mean	401
					Min	273
					Max	869

B is smaller than concerning the host C. Then, it is found out that there is the order, the host A < host B < host C, for the response time. This reason is obvious because the above order is the same as the order of three computers CPU speeds. From Table III, the mean time of the three maximum response times is 311 msec = (242+360+330)/3. Therefore, at least, three operation events seem executable in a second in the case shown in Fig.10. As well, from Table IV, the mean time of the three maximum response times is 697 msec = (357+907+829)/3. Therefore, at least, one operation event seems executable in a second in the case shown in Fig.11. In this way, granularity of events is an important factor. This depends on application types, i.e., a real-time application or not. Furthermore, when developing a practical application, you have to consider the number of computers and a network type, WAN or LAN.

C. *Improvement of System Performance by Timeout mechanism*

From Fig.12, while the host A, C, E are all existing on the same subnet, the host D is only existing on the another subnet. Then, it can be thought that the host D would become a bottleneck. However, Table IV shows that the host E rather than the host D was a bottleneck. This reason is also obvious because the host E is a lowest performance computer and its graphics hardware is not sufficient for rendering 3D images.

If the host E is removed from the system, the system performance would become better. Therefore, one way to improve system performance is to remove hosts whose response time is very large compared with the average response time, by a time-out mechanism. We are trying to introduce this mechanism.

VI. Conclusion

In this paper, we showed a consistency mechanism for distributed CSCW systems. We used the IntellligentBox system which is a user-friendly development software for interactive 3D graphics applications. It provides collaborative environments, however its consistency management mechanism was not sufficient.

We introduced mutual exclusion and object locking mechanism to the IntelligentBox system. We used the logical clock idea for consistency of distributed systems. Although communication cost for mutual exclusion of the distributed model is higher than the sever-client model or the token ring model, but it is robust, and its cost is not so much when the number of hosts is small.

We also implemented and examined an application in three cases. We measured latency time to get ownership of a *box* and response time to operate it. From the results of examinations, we found out that the lowest performance computer becomes a bottleneck, so it is possible to improve system performance by using timeout to remove such lower performance computers.

In the future, we are supposed to develop more practical applications, e.g., networked education systems, realtime network games, and so on. We will report these findings.

Acknowledgments

The authors would like to thank all members of Computing and Communications Center, Kyushu Universtiy for their suggestions.

References

[1] Y. Okada, K. Shinpo, Y. Tanaka and D. Thalmann, *Virtual Input Devices based on Motion Capture and Collision Detection*, Proc. of Computer Animation '99, pp.201–209, May 1999.

[2] Y. Okada and Y. Tanaka, *Collaborative environment of Intelligent Box for distributed 3D graphical applications*, The Visual Computer, No.14, pp.140–152, 1998.

[3] Y. Okada and Y. Tanaka, *Collaborative Environment of Intelligent Box for Distributed 3D Graphical Applications*, Proc of Computer Animation '97, pp.22–30, Jun. 1997.

[4] Y. Okada and Y. Tanaka *IntelligentBox : A Constructive Visual Software Development System for Interactive 3D Graphic Applications*, Proc of Computer Animation '95, pp.114–125, Apl. 1995.

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

[6] S. A. Bly, S. R. Harrison and S. Irwin, *Media Spaces: Video, Audio, and Computing*, Comm. of the ACM, Vol.36, No.1, pp.28–47, Jan. 1993.

[7] R. W. Root, *Design of a Multi-Media Vehicle for Social Browsing*, Conf. on CSCW (CSCW 88), pp.25–38, 1988.

[8] G. Foster and M. Stefik, *Cognoter, Theory and Practice of a Colaborative Tool*, Conf. on CSCW (CSCW86), pp.7–15, 1986.

[9] C. A. Ellis, S. J. Gibbs and G. L. Rein, *Groupware: Some Issues and Experiences*, Comm. of the ACM, Vol. 34, No. 1, pp.38–58, Jan. 1991.

[10] T. Winograd, *A Language/Action Perspective on the Design of Cooperative Work*, Human-Computer Interaction, Vol. 3, No. 1, pp.3–30, 1987, http://hci.stanford.edu/%7Ewinograd/papers/language-action.html.

[11] N. Striz, GeiBler, J. Haake and J. Hol, *DOLPHIN : Intelligent Meeting Support across LiveBoards, Local and Remote Desktop Environments*, Proc. of ACM 1994 Conf. on Computer-Supported Cooperative Work(CSCW'94), pp.345–358, 1994.

[12] H. Ishii, M. Kobayashi and J. Grudin, *Integration of Interpersonal Space and Shared Workspace: ClearBoard Design and Experiments*, ACM Trans. on Information Systems, Vol.11, No.4, p.349–375, 1993.

[13] T. Brinck and L. M. Gomez, *A Collaborative Medium for the Support of Converstional Props*, ACM CSCW'92 Proceedings, pp.171–178, 1992.

[14] Michael J. Knister and Atull Prakash, *DisEdit : A Distributed Toolkit for Supporting Multimedia Group Editors*, ACM CSCW'90 Proceedings, p.343, 1990.

[15] H. Gajewsak, J. Kistler, et.al., *Argo: A System for Distributed Collaboration*, ACM Multimedia 94, pp.433–440, 1994.

[16] R. D. Hill, *A 2-D graphics system for multi-user interactive graphics based on objects and constraints*, in E. H. Blake & P. Wisskirchen(Eds.) Advances in Object-Oriented graphics I. Springer-Verlag, 1991.

[17] G. Ricart and A. K. Agrawala, *An optimal algorithm for mutual exclusion in computer networks*, Comms. ACM, vol.24, no.1,pp.9–17, 1981.

[18] Y. Ishii, *CSCW and Groupware*, Ohmsha Ltd., 1994. (in Japanese)