경영정보학연구
제5권 1호
1995년 6월

컴퓨터 지원, 협업 환경 하에서의 동시성 제어[1)]

서 용 무[2)]

Concurrency Control in a Computer-Supported Cooperative Environment

Complexity of some problems often transcends the problem solving capability of human individuals. As such, we cannot but take a collaborative approach to those complex problems. Collaboration while using computing systems can take place through shared objects. However, current commercial DBMSs do not provide a satisfactory control for concurrent access to objects shared by collaborators. A new concurrency control scheme is proposed which will help a group of people work in a more collaborative and natural way. The idea of softlock is refined into three different kinds of softlocks and the roles of collaborators are taken into account in the proposal of the new scheme. Although these softlocks are illustrated together with read/write hardlocks in this paper, the use of softlocks can be extended to be exploited with other kinds of hardlocks, for example, of granularity locking.

---

1) The research fund was granted by the Daeyang Foundation 1994
2) 세종대학교 경영대학 정보처리학과

Ⅰ. Introduction

As problems become more and more difficult and intractable beyond individual's problem solving capability, more attention has been paid to team works or group projects these days. As such, the participants in those team works or group projects are concerned with how to attain high collaboration among themselves, while they are trying to accomplish their common goal. Such an environment in which several collaborators use computing facilities to do their common task is called a computer-supported cooperative environment (CSCE). Systems run in CSCE is called CSS][3], collaboration support systems, to emphasize that collaboration is the first concern of the users while using those systems. A few examples of CSCE are engineering design by teams, CAD/ACM, software development by a group, and co-authoring.

In CSCE, collaboration can take place through shared objects. For example, when a couple of co-authors work together to write a book about a topic, collaboration can occur by sharing even intermediate (that is, unfinished) results, such that one co-author's intermediate writing can be reviewed by another co-author and he or she may be informed of modification, if necessary. This implies that database system's support is required for collaborative activities in CSCE on intermediate objects as well as on finished ones.

However, current database systems fall short of supporting the collaborative activities in CSCE. There are a few reasons for saying this. First, a requirement for supporting collaboration in CSCE is that intermediate results of a collaborator be shared by other collaborators. Though traditional database systems primarily deal with the problem of sharing data, they sometimes do not allow sharing data by locking in an exclusive mode. Further, they do not allow a user to access intermediate results of other users. To achieve a higher level of collaboration, a lock being held on intermediate data by a collaborator should be made to be broken as necessary by other collaborators. Otherwise, they may have to wait for a long time, which has a deadly bad impact on collabo-

3) A similar term, groupware is defined in [Ellis et. al., 1991] as computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment. In this paper, a new term CSS is used to suggest the importance of collaboration.

ration. Here, we agree that traditional multi-user database systems have insulated users from each other, rather than helping them collaborate to achieve a common goal [Ellis et. al., 1991].

Second, another requirement for supporting collaborative activities in CSCE is that collaboration should be supported naturally. Diverse organizational and/or social aspects (e.g., roles and responsibilities of collaborators, incentives etc.) need to be taken into account when developing a system to run in CSCE. As many users work together, the roles of collaborators in an organization come into play. For example, a manager of a group project may have to be allowed to access an object related to the project even when a member of the project is currently using that data (i.e., holding a lock in a traditional DBMSs). Traditional database systems do not allow such an access to be made by the manager, thereby allowing only a limited collaboration.

For these reasons, traditional database systems can be said to be inadequate to supporting collaborative activities in CSCE. The objective of this paper is to suggest a new way of supporting collaboration in CSCE. Section 2 describes a scenario that will help understand the situation which requires collaboration among a group of people working together toward a common goal. Section 3 reviews several ways of supporting concurrency control in DBMSs, published in the literature. In section 4, we suggest a role-based soft lock protocol as a new way of supporting collaboration in CSCE. Section 5 summarizes the paper.

Ⅱ. An Example Scenario of collaboration in CSCE

We want to start with a scenario in CSCE, in which a group of people need to collaborate to work together successfully in a group project of motion analysis. Suppose the project is to find trajectories of moving objects from a sequence of time-varying frames. Researchers both from image processing field and from computer vision field are supposed to take part in this project. For example, suppose the participants consist of a principal investigator (PI), a senior researcher (SR1) and two junior researchers (JR11 and JR12) from the image processing field, and a senior researcher (SR2) and two junior researchers (JR21 and JR22) from the computer vision field.

It is usual that a big problem is decomposed into small ones, each of which in

turn is further decomposed into smaller ones of manageable units, iteratively. And a solution for the big problem is obtained by integrating solutions to the small problems. PI, as a manager of this group project, decided to follow the same divide-and-conquer strategy for this project. So, he decomposed the whole project into two small tasks as follows:

1) finding objects in each frame (T1)

2) matching objects found in a frame to those found in another frame, for each pair of succeeding frames (T2)

PI assigned the first task (e.g., named as T1) to SR1 and the second (e.g., names as T2) to SR2. SR1 decomposed T1 into three subtasks (e.g., named as T11, T12, and T13) and T13 into two smaller ones (e.g., named as T131 and T132). SR2 assigned T2 both to JR21 and JR22, asking them to use two different matching algorithms. So, the final decomposition of the original task is:

1) finding objects in each frame (T1 by SR1)
   a) taking a sequence of frames of moving objects (T11 by JR11)
   b) digitizing them into computer systems (T12 by JR12)
   c) finding objects from each frame (T13 by SR1)
      i) extracting basic features from each frame (T131 by JR11)
      ii) finding objects based on the features extracted (T132 by JR12)

2) matching objects found in a frame to those found in another frame, for each pair of succeeding frames (T2 by SR2)
   a) matching objects using algorithm A (T21 by JR21)
   b) matching objects using algorithm B (T22 by JR22)

This situation can be depicted as in the figure 1, which shows the whole task T is decomposed into T1 and T2, and Ti is decomposed into T11, T12 and T13, and so on.

Because of the dependency among the subtasks, collaboration is required among the participants through shared objects. For example, since the output from task T1 will be used in T2, SR2 should keep a constant interaction in some way with SR1 with respect to the output. If there needs to be a change in the output of T1 for some reason, SR1 should inform SR2 of the change. Similarly, JR11 and JR12 should collaborate each other, because JR12 uses the features extracted by JR11

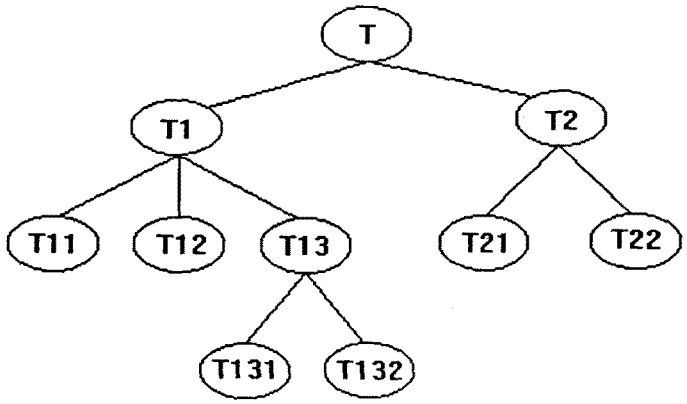

Figure 1 : a task decomposition

to find objects in each frame. So should JR21 and JR22, because they share objects recognized from each frame to find the matching objects in succeeding frames.

Now, let us consider several situations which have room for a further collaboration in this group project (Here, we have listed only some situations that might take place while carrying out the task T1.):

1) JR11 does not allow JR12 to make an access to his output until he finishes his job completely (e.g., T131: extracting features from all frames).
   — It would be better for JR11 to allow JR12 to make an access to his partial output (e.g., output from some frames), so that JR12 starts his job as soon as the partial output from T131 becomes available.
2) SR1 is not allowed to make an access to intermediate output of JR11 or JR12.
   — As a person who is responsible for the task T13, SR1 needs to verify the outputs both from task T131 and from task T132. As such, it would be better for SR1 to be able to access their outputs at any time.
3) SR1 is not allowed to make a change to the output of JR11 or JR12, either.
   — What if JR11 (or JR12) is not available for a long time when his output needs to be changed ? In that case, it would be better for SR1 to have the authority to make the change.

Objects to be shared among the collaborators include requirement specifications and various notes on each task, as well as outputs from each task. Those notes are put into the system by any member of the group project. For example, PI can make a note about the whole project, such as due date for each subtask T1 and T2. Also, JR11 may put a note, saying that a new set of features extracted using a different parameter has been created into files with certain names, so that JR12 can use it for his job.

When several people are involved in a group project like above, it is desirable to set up a set of regulating rules that will help the participants to coordinate their activities. Considering the situations mentioned above, the regulating rules can be established solely by PI, by PI with the help of SR1 and SR2, or by group members in a meeting. (Some of them may be inherent to the organization and some of them may be peculiar to the project[4], they need to set up a set of regulating rules, which may be decided on the dependency among the different chapters of the book . }}.) Anyway, those rules will specify what is the role (i.e., in terms of authority and responsibility) of each participant. Here, we enumerate some of the plausible regulating rules for the project:

rule 1: At any time, PI can tell any participant in the project to show the current results of the task that he or she is working on.

rule 2: At any time, SR1 can tell JR11 and JR12 to show the current results of their work. (At any time, SR2 can tell JR21 and JR22 to show the current results of their work.)

rule 3: At any time, JR11 and JR12 can ask each other to show the current result of their work. (At any time, JR21 and JR22 ask to each other to show the current results of their work.)

rule 4: SR1 is responsible for declaring the output of task T1 public, so that any member of the project can access it. (SR2 is responsible for declaring the output of task T2 public, so that any member of the project can access it.)

rule 5: For private objects which belong to T1 and are not yet declared as public, any one of SR1, JR11 and JR12

4) Different regulating rules should be defined depending on the task to achieve. For example, when co-authors work together to write a book

can ask the person who is responsible for them to make a change, if necessary. (For private objects which belong to T2 and are not yet declared as public, any one of SR2, JR21 and JR22 ask the person who is responsible for them to make a change, as necessary.)

rule 6: Anyone who finds it necessary to make a change to a public object, should inform of this to the senior researcher who declared it public. Then the senior researcher should tell the junior researcher who worked on it to make a change, and he should inform the other senior researcher of the change. rule 7: The role of a group member can be played by someone who is in a higher position in the role hierarchy of figure 1. Also, anyone can give his or her rights to another involved in the project (i.e., someone who is in a lower position in the role hierarchy).

Note that some of the regulating rules are associated with concurrency control and with access authorization in database systems, but they are not supported by concurrency control schemes of the current database systems, which will be reviewed in the next section. Also, note that these regulating rules can be made as a shared object.

Ⅲ. Related Work: Concurrent Access to a Shared Object

To support multi-users who may want to access shared objects almost at the same time is called concurrency control in DBMSs. Traditional concurrency control aims to maintain consistency based on the concept of seriabilizability and atomicity. Concurrency control in current commercial DBMSs mainly depends on locking mechanism. In this section, some of the concurrency control protocols developed in the database field are reviewed from the viewpoint of supporting collaboration: 1) the two-phase locking protocol [Eswaran et. al., 1976], 2) the granularity locking protocol [Gray, 1978], 3) protocols for coordinating concurrent development [Harrison, 1990], 4) a locking mechanism with soft lock modes [Hornick and Zdonik, 1987], and 5) concurrency control within groupware [Ellis, 1991: Yeh et. al., 1989: Ellis and Ege, 1987].

Two-Phase Locking Protocol In this protocol, all lock requests are done in the

growing phase, and then they are released in the shrinking phase. This protocol ensures the serializability, however, it is not flexible enough to be used in CSCE, because an X lock on a shared object prevents other transactions from accessing it. To enhance the level of collaboration through shared objects, the idea of lock conversion might be used as introduced in [Garza and Kim 1988: Korth et. al., 1988: Korth and Silberschatz, 1991].

Granularity Locking Protocol To reduce the number of locks required on a hierarchically structured object, a richer set of lock modes is developed and the concept of implicit locking is utilized. The granularity locking protocol can be applied with a slight modification to a class hierarchy of an object-oriented database, in which a class has only one superclass. Locking protocols proposed and implemented for an object-oriented database system, called ORION, are illustrated in [Kim, 1991]. The class hierarchy in which a class may have more than one superclass, and composite object were taken into account in the protocols.

Similar approach has been taken in [Korth et. al., 1988] to handle long-duration transactions in a CAD environment, which is further characterized by interactive modifications.

Design objects and versions are added to the set of granules of database, and predicate-wise two-phase locking and update-mode locks for lock conversion are suggested to increase the parallelism among the CAD transactions.

Both two-phase locking and granularity locking protocols ensure the serializability. However, they both are not free from such problems as deadlock and long waiting time, incurring when a conflicting lock is held on an object by a long-duration transaction. As such, obviously, neither do they satisfy the requirements aforementioned in section 1.

Protocol for Coordinating Concurrent Development A new model for concurrent development of software systems, documents or engineering designs is presented in [Harrison et. al., 1990] based on the concept of coordination consistency. The coordination consistency is defined in terms of change-serializability, atomicity, and completeness. Change-serializability differs from the usual concept of serializability in that only changes are taken into account, thereby permitting much greater concurrency than serializability. This also ensures that changes occurred in parallel would not be overwritten by a merge. Complete-

ness ensures that nothing is lost in the modification histories.

A set of locking protocols for concurrent development using the hierarchy of stores is presented which endures the coordination consistency by making all merges safe. Safe merge means that every file in one file set has been derived from other file in another file set to be merged. A strict locking protocol, which requires that a file be locked in all stores into which it might be merged later, prevents concurrent modification of a file by multiple modification activities. Another protocol, called a lenient locking protocol, warns a developer who is about to change the file which is locked but allows him to do that, in order not to hold up his task. In this case, a special consideration must be taken into, because merge may not be safe.

Here, we can see that although by using the concept of change-serializability it allows greater concurrency, it does not allow the file which is being modified to be available to other modification activity. If the modification lasts long time, the file would not be available for a long time.

Locking Protocols based on Soft Locks

In [Horniek and Zdonik, 1987], a set of lock modes and a set of communication modes are proposed to allow users to define new protocols easily. Proposed lock modes are: null, NR(nonrestrictive)-read, R(restrictive)-read, NR-write, and R-write from the weakest in that order. While an NR-read lock is held on an object, other transactions can request a write lock (either NR-write or R-write) on the object. An R-read lock is the typical read lock in that it prevents other transactions from writing, and therefore, as long as an R-read lock is held on an object, a request for either NR-write or R-write lock is prohibited. Also, an R-write lock is the typical write lock in that it does not allow either read or write lock.

Proposed communication modes are: U-notify (need-to-update notification), R-notify (need-to-read notification), W-notify (need-to-write notification), RW-notify, and N-notify (no notification). For each lock mode, different kinds of notification are valid, depending on the operation. Out of 25 combinations of lock modes and communication modes, only 14 is meaningful, and users can use subset of them to satisfy their application requirements. For example, for an interactive cooperative application, one may use RW-notify in combination with NR-write or R-write lock. Holders of such a lock, when notified of other client's lock request, can either free

the lock or commit the change to the locked object, thus allowing other clients to lock the object.

In this mechanism, a lock on an object can be broken and its owner is informed of the potential change on the object by another user. However, this mechanism does not satisfy one of the requirements to support collaboration in CSCE. That is, a lock can be broken, independently of the roles of the lock holder and the requester.

Concurrency control within real-time groupware An object base management system, Gordion, depends on the soft locking mechanism coupled with notification to control concurrent access to shared objects. Another approach to concurrent access is to use versioning mechanism coupled with notification. That is, either whenever a lock is broken or whenever a new version of the shared object is created, notification is forwarded to the user of the shared objects. Then discussion through a phone call is utilized as a medium to resolve the conflict access to the shared objects.

In summary, the two traditional locking mechanisms for concurrency control (i.e., two-phase locking and granularity locking) is not appropriate to be used in CSCE because of the waiting time incurring in a system adopting hard locking mechanism. Therefore, to support the collaborative activities in CSCE, we need to adopt the soft locking mechanism. However, the soft locking mechanism introduced in the above is not a natural way of breaking locks held on shared objects. That is, breaking locks occurs irrespective of the roles of the lock owner and the requester. To our best knowledge, there has been no attempt to reflect the roles into the concurrency control mechanism. In the next section, we specialize the soft lock into a variety of soft locks and propose a role-based soft lock mechanism in which roles of the lock owner and the requester are taken into account, together with specialized soft locks.

Ⅳ. A Role-Based Soft Locking Mechanism in CSCE

A variety of Soft Lock Soft lock is a lock that can be broken after a lock is set on a shared object. As is mentioned before, we need to further specialize it into a variety of soft locks: a soft lock requiring negotiation, a soft lock requiring a notification, and a soft lock which can be broken without notification or negotiation but based on the roles of lock holder and lock requester. For a soft lock requiring negotiation, a shadow copy of the locked object

is made for new transactions, which should contact the lock holder prior to its commitment. This allows multiple transactions to have concurrent access to a shared object without holding up other transactions. For a soft lock requiring notification, a lock can be set so that it can be broken without negotiation with the current lock holder but the fact that the lock has been broken should be notified to the lock holder. For a soft lock requiring comparison of roles of the lock holder and the requester, a lock can be broken, if the role of the lock requester subsumes the role of the current lock holder. Therefore, the role hierarchy[5] of users should be maintained in the system.

A Role-Based Soft Lock Protocol Now, we propose a role-based soft lock protocol, in which whether or not a lock can be broken is determined based on the kind of locks and roles of the lock holder and the lock requester. We assume that the roles of users can change dynamically depending on which project they participate in (As such, a role hierarchy is maintained for each project); a role hierarchy of users is maintained in a database; database systems provide various kinds of soft locks as introduced in the previous paragraph, including read(shared) and write (exclusive) hard locks. This role-based soft lock protocol requires to define a lock compatibility matrix and a role hierarchy among users.

A Lock Compatibility Matrix First, let us define a lock compatibility matrix. Assume that we have implemented several types of locks in our database system: a hard read lock and a hard write lock (denoted as Wr and Wh, respectively), a soft read lock requiring notification and a soft write lock requiring notification (denoted as Rs-ntfy and Ws-ntfy, respectively), a soft read lock requiring negotiation and a soft write lock requiring negotiation (denoted as Rs-nego and Ws-nego, respectively), and a soft read lock based on roles and a soft write lock based on roles (denoted as Rs-role and Ws-role, respectively). The compatibility matrix in Table 1 shows the semantics of this protocol.

It can be answered from the table whether the current lock is compatible with the requested lock or whether the current lock is breakable: 1) entries with

5) Each member in a group has one or more roles to play. These roles can be defined in a role hierarchy depending on the importance of roles in a group. (In [Rabitti, et. al., 1991], a role lattice is defined as a DAG among the users of different roles to reduce the number of authorization subjects.)

value 'Ok' indicate that the current lock and the requested lock are compatible and thus the lock request can be accepted; 2) entries with value 'No' indicate that the current lock and the requested lock are compatible and thus the requested lock cannot be accepted; 3) entries with value 'Brk' imply that the current lock may be broken (without notification or negotiation). It depends on the roles of the lock holder and the lock requester; 4) entries with value 'Ntfy' imply that the current lock can be broken after notification to the lock holder is made; 5) entries with value 'Brk/Nego' imply that the breakability of the current lock is determined, first by the relationship between the two roles of the lock holder and the lock requester, and then (i.e., if the first attempt fails) by negotiation with the lock holder. Entries with 'Brk' is further explained below.

An algorithm for Determining Lock Breakability Next, let us consider a role hierarchy, which represents access priority among the different roles of collaborators. That is, a collaborator playing a role in a higher level of the role hierarchy has the priority over those playing the roles in a lower level when accessing a shared object. This hierarchy reflects some of the regulating rules, especially those concerning the access priority.

In organizations, a person can be involved in multiple projects, and/or involved in more than one task within a project. Thus, if we draw a role hierarchy of

Table 1 : Compatibility Matrix

Requested / Currently held		HardLock		SoftLock					
		Rh	Wh	Rs-ntfy	Ws-ntfy	Rs-nego	Ws-nego	Rs-role	Ws-role
Hard Lock	Rh	Ok	No	Ok	No	Ok	No	Ok	No
	Wh	No	No	No	No	No	No	No	No
Soft Lock	Rs-ntfy	Ok	Ntfy	Ok	Ntfy	Ok	Ntfy	Ok	Ntfy
	Ws-ntfy	Ntfy	Ntfy	Ntfy	Ntfy	Ntfy	Ntfy	Ntfy	Ntfy
	Rs-nego	Ok	Brk/ Nego	Ok	Brk/ Nego	Ok	Brk/ Nego	Ok	Brk/ Nego
	Ws-nego	Brk/ Nego	Brk/ Nego	Brk/ Nego	Brk/ Nego	Brk/ Nego	Brk/ Nego	Brk/ Nego	Brk/ Nego
	Rs-role	Ok	Brk	Ok	Brk	Ok	Brk	Ok	Brk
	Ws-role	Brk	Brk	Brk	Brk	Brk	Brk	Brk	Brk

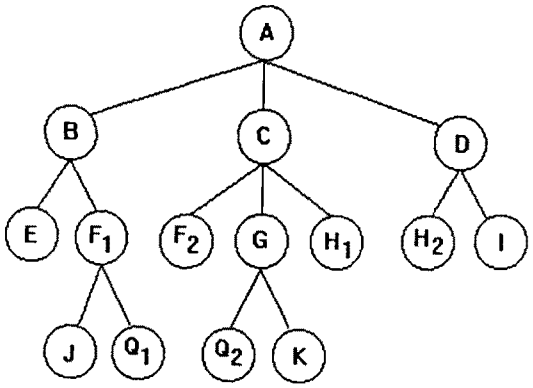

Figure 2 : an example of a role hierarchy

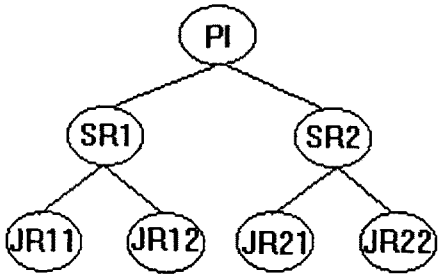

Figure 3 : a role hierarchy

all projects, it may become a directed acyclic graph (DAG) which has several roots, one for each project. However, we maintain a separate role hierarchy for each project, because access priority is meaningful when defined among the collaborators who share the same objects of a project. For the same reason, the roles of a person who take part in more than one task, are represented distinctively in each role hierarchy. As a result, a role hierarchy becomes not a DAG but a tree. An example of a role hierarchy is shown in Figure 2, in which multiple roles of some persons are distinguished using subscripts.

Since the fact that a collaborator has granted his or her rights to another cannot be incorporated into the role hierarchy, a separate list of pairs (R1, R2) needs to be kept along with the role hierarchy. A pair

(R1, R2) indicates that a person of role R1 has granted his or her authority to access shared objects to a person of role R2. Note that such a grant can take place among the collaborators who share objects.

Now, let us describe an algorithm for determining lock breakability, given a role hierarchy tree (RHT), a list of pairs (GPL), indicating grants of roles, and a current lock holder (Rh) and a lock requester (Rr). The breakability of a lock is determined as follows:

1) Adjust Rh and Rr according to the grant pairs, if any, in GPL: Scan the list of pairs to check whether there is any pair, in which Rh appears as the second element in the pair. If such a pair (R1, Rh) is found, replace Rh with R1. Repeat this for all such pairs. Change Rr, in the same way.
2) Determine the breakability by comparing Rh and Rr: If Rh and Rr do not belong to the same branch, the current lock cannot be broken.

Otherwise, search the tree depth-first. If Rr appears before Rh in the same branch, the current lock can be broken. Otherwise, the current lock cannot be broken.

Suppose we have to determine the lock breakability of a lock whose current holder is F_2 and the lock requester is Q_2, in Figure 2. If we have (C,F_2) and (G,Q_2) in the list of pairs then Step 1) changes the role of a person whose role is F_2 to C and that of a person whose role is Q_2 to G. In step 2), we search for either C or G in the same branch. Since C is encountered prior to G, it implies that the current lock cannot be broken. If we have only (G, Q2) in the list of pairs, we have to search for either F2 or G in the same branch. But they are not in the same branch, and thus it is implied that the current lock cannot be broken. What if the list contains only (G,Q2) and (C,G)? Then the role of a person C has been granted to a person whose role is Q2. Thus, the lock can be broken. Now, let us take an example of group project of motion analysis mentioned earlier. Suppose that the role hierarchy among the collaborators for a set of objects O1, O2, O3 and O4 is defined like the one in Figure 3, and suppose that SR1 is holding a Wh lock on object O1 and a Ws-ntfy lock on object O2, and SR2 a Ws-role lock on object O3 and a Ws-nego lock on object O4, respectively. The above compatibility matrix can be explained by the following cases:

1) If anyone requests an Rs-role lock

on object O1, then his or her request cannot be accepted, because currently a Wh lock is being held on the object which cannot be broken.

2) If anyone requests any lock on object O2, then the request is accepted and notification is made to the current lock holder SR1. It is independent of the roles of the current lock holder and the lock requester.
3) If anyone requests any lock on object O3, then the current lock Ws-role is broken without any notification to SR2. In this case, the current lock can or cannot be broken, depending on the role of the requester. For example, if PI is the requester it is broken, while if the requester is JR21 or JR22 it is not broken.
4) If the PI requests an Rs-role lock on object O4, then the current lock is broken without negotiation. But what if SR1, JR11 or JR12 requests the Rs-role lock on the same object O4? Then, the requester has to negotiate with the current lock holder, SR2, because no access priority is defined between the current holder and the requester in the role hierarchy.

V. Summary

Complex and large-scale problems require collaborative effort of a group of people who have interest in the problems. When a group of people are working together for a common goal, a person's behavior should not be a barrier to another person's behavior, whether or not they use computing systems. For example, a collaborator's absence should not prevent another collaborator from doing his or her work.

Literature survey shows that traditional database management systems have not supported the collaboration of users working for a common goal, since they do not allow a lock to be broken naturally as in real life. This implies a long duration transaction initiated by a collaborator may bring a long delay to another collaborator who wants to access the shared object, if it is locked by the long transaction. As a solution to this long delay problem, we have proposed a role-based softlock concurrency control scheme. In this scheme, a lock may be broken to prevent long delay from occurring, mainly depending upon the roles of the current lock holder and the lock requester on a shared object.

The role-based softlock concurrency

control scheme introduced in this paper seems to be extended further in two aspects: search for other organizational aspects to be incorporated into the scheme, and combination of softlocks with other hardlcoks, for example, of granularity locking.

참 고 문 헌

Ellis, C. and Ege, A., "Design and Implementation of Gordion, an object Base Management System", *Proceedings of International Conference on Data Engineering*, 1987

Ellis, C., "A Model and Algorithm for Concurrent Access within Groupware", a working paper, 1991

Ellis, C., Gibbs, S., and Rein, G. "Groupware: Some Issues and Experiences," *CACM*, Vol. 34, No. 1, 1991

Eswaran, G.D., Gray, J.N., Lorie, R.A., and Traiger, I.L., "The Notions of Consistency and Predicate Locks in a Database Systems," CACM, Vol. 19, No. 11, 1976

Garza, F.J., and W. Kim, "Transaction Management in an Object-Oriented Database System," in Proc. *ACM SIGMOD International Conference on Management of Data*, Chicago, Ill., 1988

Gray,J.N. "Notes on Database Operating Systems", IBM Research Report: RJ2188, *IBM Research, San Jose*, Calif. 1978

Harrison, W., Ossher, H., and Sweeney, P., "Coordinating Concurrent Development," in Proc. *CSCW*, 1990

Hornick, M., and Zdonik, S., "A Shared, Segmented Memory System for an Object-Oriented Database," *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, 1987

Kim, W., *Introduction to Object-Oriented Database*, The MIT press, 1991

Korth, H.F., Kim, W., and Bancilhon, F., "On Long-Duration CAD Transactions,", *Information Sciences Journal*, 46, 1988

Korth, H.F., and Silberschatz,A., *Database System Concepts*, McGraw-Hill, 1991

Rabitti, F., Bertino, E., Kim, W., and Woelk, D., "A model of Authorization for Next-Generation

Database Systems", *ACM Transactions of Database Systems*, Vol. 16, No. 1, March, 1991

Yeh, S., Ellis, C., Ege, A., and Korth, H.F., "Performance Analysis of Two Concurrency Control Schemes for Design Environment", *Information Sciences Journal*, 49(1), 1989.

◇ 저자소개 ◇

저자 서용무 는 서울대 수학교육과를 졸업하고, 한국과학원에서 전산학 석사, University of Texas (at Austin)에서 경영정보학 박사학위를 취득하고, 현재 세종대학교 경영대학 정보처리학과에서 조교수로 재직 중이다. 한국과학원 졸업 후, 3년간 한국과학기술 연구소에서 근무했으며, 박사학위 취득 후 잠시 삼성데이타시스템에서 컨설팅 팀의 일원으로 근무했다. 주요 관심 분야는 객체지향 패러다임, Organizational Computing, Use of IT to support Collaboration, 등이다.