An Optimal Locking Scheme in Object-Oriented Database Systems

Woochun Jun
Dept. of Computer Education
Seoul National Univ. of Education
Seoul, Korea

Le Gruenwald
School of Computer Science
Univ. of Oklahoma
Norman, OK 73069
USA

Abstract

In this paper, a locking-based concurrency control scheme is presented for object-oriented databases (OODBs). It is designed for controlling accesses to class hierarchy, which is important concept in OODBs. Based on access frequency for each class, the proposed scheme incurs less locking overhead than existing works, explicit locking and implicit locking, for any OODB environments. In this paper, it is theoretically proven that the proposed scheme performs better than existing schemes.

1. Introduction

OODBs have been popular for many non-traditional database environments such as computer-aided design, artificial intelligence, etc. In a typical OODB, a class object consists of a group of instance objects and class definition objects. The class definition consists of a set of attributes and methods that access attributes of an instance or a set of instances. In OODBs, users can access objects by invoking transactions consisting of a set of method invocations on objects [2].

A concurrency control scheme is used to coordinate multiple accesses to the multi-user database so that it maintains the consistency of the database. A concurrency control scheme allows multi-access to a database but incurs an overhead whenever it is invoked. This overhead may affect on the performance of OODBs where many transactions are long-lived. Thus, reducing the overhead is critical to improve overall performance.

The inheritance is very important concept in OODBs. That is, a subclass inherits definitions defined on its superclass. Also, there is an *is-a* relationship between a subclass and its superclass so that

an instance of a superclass is a generalization of its subclasses [5]. This inheritance relationship between classes forms a class hierarchy. There are two types of accesses to a class hierarchy, MCA (multiple class access) and SCA (single class access), respectively [6]. MCA is an operation accessing possibly more than one class in the class hierarchy. MCAs are operations such as class definition modification operation and instance access to all or some instances of a given class and its subclasses. On the other hand, SCA is an operation accessing one class in the hierarchy. For example, SCAs are operations such as class definition read operation and instance access to a single class. Due to inheritance, for a locking based concurrency control scheme, when a MCA operation is requested on some class, say C, it may be necessary to get locks for all subclasses of C as well as C.

In the literature, there are two approaches dealing with class hierarchy, explicit locking and implicit locking, which will be discussed in Section 2. These approaches may work well only for particular applications in OODBs. That is, explicit locking incurs less locking overhead for transactions invoking mostly SCA operations. On the other hand, implicit locking incurs less locking overhead for transactions invoking mostly MCA operations. Recently a locking-based concurrency control scheme for class hierarchy in OODBs is presented [6]. The scheme is based on so called *special class* (SC) and can be used for any applications with less locking overhead than both explicit locking and implicit locking. In [6], with an assumption that the number of access is stable for each class, it is shown that the proposed scheme (called SC-based scheme) performs better than both explicit locking and implicit locking. Based on that work, in this paper, a new concurrency control scheme is proposed. Also, it is

proven that the proposed scheme incurs less locking overhead than explicit locking, implicit locking and the SC-based scheme .

This paper is organized as follows. In Section 2 we review previous works dealing with class hierarchy. In Section 3 a new concurrency control scheme is proposed. In Section 4, it is shown that the proposed scheme performs better than existing works. The paper concludes with future research issues in Section 5.

2. Related Work

2.1. Explicit Locking and Implicit Locking

In the literature, there are two major locking-based approaches dealing with a class hierarchy: explicit locking [2,9] and implicit locking [5,7,8]. In explicit locking, for a MCA operation on a class, say C, a lock is set not only on the class C, but also on each subclass of C in the class hierarchy. For an SCA operation, a lock is set for only the class to be accessed (called target class). Thus, for an MCA, transactions accessing a class near the leaf in a class hierarchy will require fewer locks than transactions accessing a class near the root in the class hierarchy. Also, another advantage is that it can treat single inheritance where a class can inherit the class definition from one superclass, and multiple inheritance where a class can inherit the class definition from more than one superclass in the same way. But, explicit locking incurs more locking overhead for transactions accessing a class near the root in a class hierarchy.

On the other hand, the implicit locking is based on intention locks [3]. The purpose of an intention lock on a class indicates that some lock is set on a subclass of the class. Thus, when a lock is set on a class C, it is required to set extra locking on a path from C to its root as well as on C. In implicit locking, when a MCA operation is accessed on a class, say C, locks are not required for every subclass of the class C. It is sufficient to set a lock only on the class C (in single inheritance) or locks on C and its subclasses having more than one superclass (in multiple inheritance) [5]. Thus, for a MCA access, it can reduce locking overhead than explicit locking. But, implicit locking requires more locking overhead when a target class is near the leaf in a class hierarchy due to intention lock overhead.

2.2. The SC-based Scheme

In [6], the SC-based scheme is proposed to incur less locking overhead than existing schemes, explicit locking and implicit locking. The scheme is based on SC where a SC is a class on which MCA operations are performed frequently. How to determine if a class is a SC or not will be discussed later.

The basic idea is summarized as follows. In that scheme, intention locks are set on only SCs. Thus, locking overhead is reduced than implicit locking requiring intention locks on every superclass of the target class. Also, in order to have less locking overhead than explicit locking, the following principle is adopted: for an SCA access, a lock is set on only the target class, like explicit locking. For a MCA access, unlike explicit locking, locks are set on every class from the target class to the first SC through the subclass chain of the target class. If there is no such SC, then locks are set on leaf classes. If the target class is an SC it self, then set a lock only on the target class.

The scheme is presented as follows. Assume that a lock is requested on class C. For simplicity, strict two-phase locking [1,4] is adopted.

Step 1) locking on SCs

- For each *SC* (if any) through the superclass chain of C, check conflicts and set an intention lock if it commutes. If it does not commute, block the lock requester.

Step 2) Locking on a target class

- If the lock request is an SCA, check conflicts with locks set by other transactions and set a lock on only the target class C if it commutes and set a lock on an instance if *a* method is invoked on the instance and commute. If it does not commute, block the requester.
- If the lock request is an MCA, then, from class C to the first *SC* (or leaf class if there is no SC) through the subclass chain of C, check conflicts and set lock on each class if commute. If the class C is an SC, then set a lock only on C.
- If class C has more than one subclass, perform the same step 2) for each subclass chain of C.

For SC-based scheme, the following SC assignment scheme is adopted in [6]. Assuming that the number of access to each class is stable and access frequency (of MCA and SCA) to each class is known in advance, the SC assignment scheme is constructed as follows.

//Start from each leaf class until all classes are checked //

step 1) If a class is a leaf, then the class is assigned as non-SC.

If a class, say C, has not been assigned yet and all subclasses of C have been already assigned, then do the followings

for class C and all of the subclasses,

calculate the number of locks (N_1) when the class is assigned as SC

calculate the number of locks (N_2) when the class is assigned as non-SC

step 2) Assign it as SC only if $N_1 < N_2$

For example, consider a simple single-inheritance class hierarchy as in Fig 1.a and assume access frequency information on each class as in Fig. 1.b. Note that, for MCA operations, the numbers represent only access initiated at a given class. Thus, the number of MCA accesses initiated at its superclasses is not counted. The SC assignment to each class is follows. First C_1 is assigned as non-SC since C_1 is a leaf class. At the class C_2, if C_2 is assigned as non-SC, the number of locks needed for class C_1 and C_2 are 200 (for C_1) and 700 (for C_2), respectively, resulting 900 locks. On the other hand, if C_2 is assigned as SC, then locks needed for classes C_1 and C_2 are 800 locks, where 400 locks are for C_1 (200 locks for MCA and 200 locks for SCA) and 400 locks are for C_2 (100 locks for MCA and 300 locks for SCA). Thus, C_2 becomes SC. Similarly, two other classes C_3 and C_4 become non-SCs. Fig. 1.c shows the result of the SC assignment scheme.

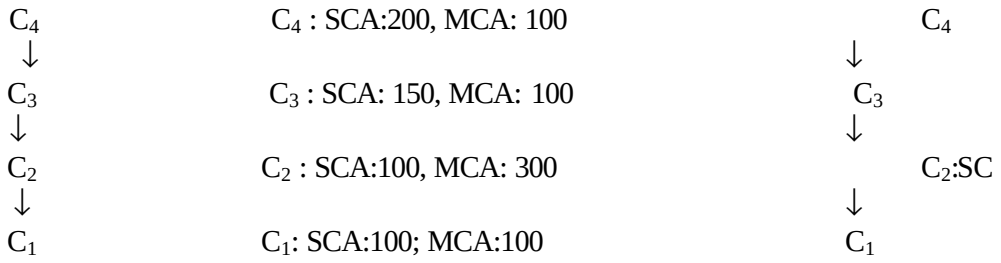

Fig. 1.a. A class hierarchy Fig. 1.b. Access frequency for each class Fig. 1.c. Result of SC assignment

Based on the above assignment scheme, consider the following lock requests by two transactions T_1 and T_2 on a class hierarchy in Fig. 2.a

1) T_1: class definition modification operation on class C6

2) T_2: class definition read on class C4

Let L_i be a lock L set by transaction T_i. Assume that class C1, C4, C7 and C10 are SCs. As in Fig 2.b, 2.c, and 2.d, 6, 7 and 10 locks are required for T_1 and T_2 by the SC-based scheme, explicit locking, and implicit locking, respectively.

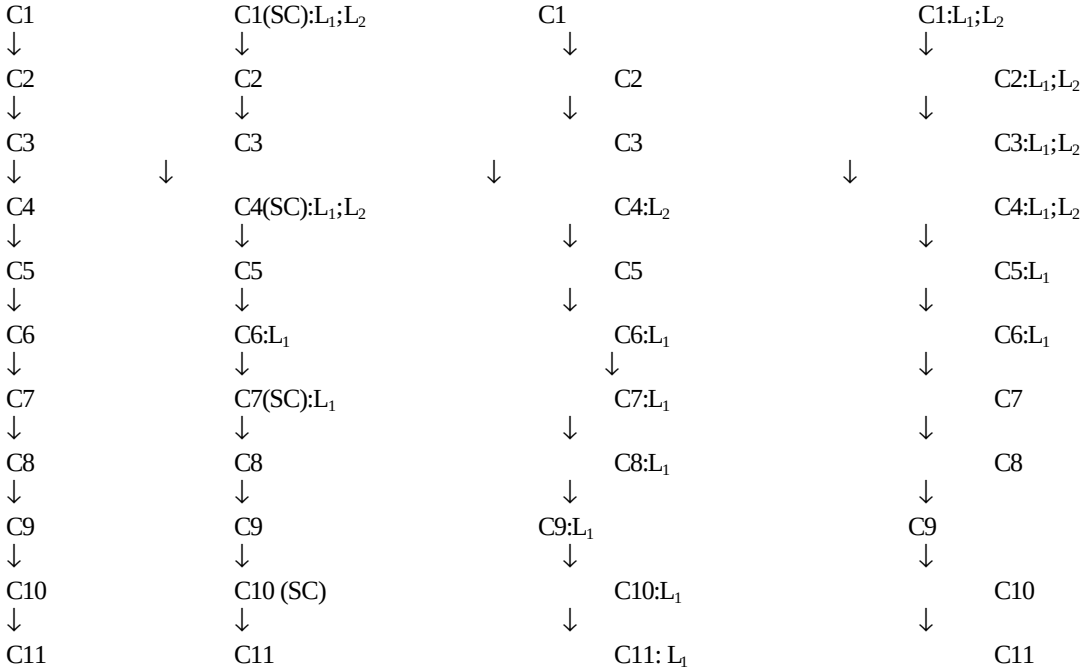

Fig 2.a class hierarchy Fig. 2.b. Locks by SC-based scheme Fig. 2.c. Locks by Explicit locking Fig. 2.d. Locks by Implicit locking

3. Proposed class hierarchy locking scheme

3.1. Background

The proposed scheme is based on the SC-based scheme. The basic idea is that some redundant locks can be reduced without affecting the correctness of the scheme. Assume that a class C is accessed so that it needs to be locked. For SC-based scheme, an intention lock is set on every SC through the superclass chain of C On the other hand, the proposed scheme does not have to set intention locks on every SC through the superclass chain. That is, only the first SC near root and the last SC near the class C need to be locked as long as SCs excluding the first SC and the last SC have only one subclass.

For example, consider the class hierarchy in Fig. 3.a. Also, assume the following access by transaction T_3. Fig 3.b and Fig. 3.c show locks by the SC-based scheme and the proposed scheme, respectively.

T_1: class definition update operation on class C11.

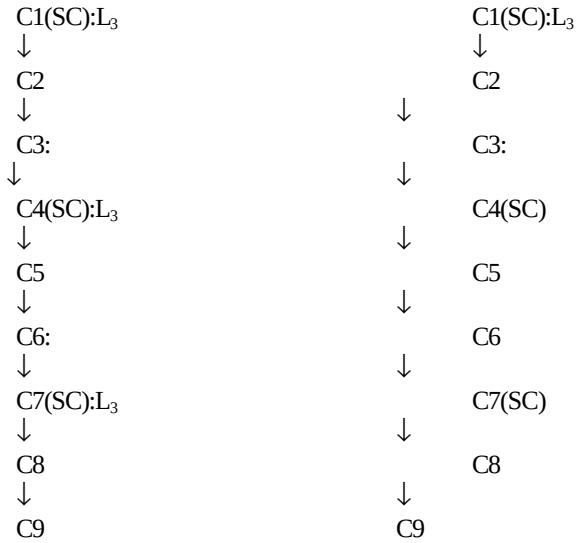

Fig. 3.b. Locks by SC-based scheme Fig. 3.c. Locks by the proposed scheme

3.2. A New Class hierarchy Locking Scheme

Based on idea explained as in Section 3.1, the proposed scheme is as follows. Assume that a lock is requested on class C. Also, it is assumed that the strict two-phase locking is adopted.

Step 1) locking on SCs

- (case I) at least one of SC excluding the first SC and last SC through the superclass chain of C has more than one subclass.

For each *SC* (if any) through the superclass chain of C, check conflicts and set an intention lock if it commutes. If it does not commute, block the lock requester.

- (case II) Otherwise

For the first SC and the last SC through the superclass chain of C, check conflicts and set an intention lock if it commutes. If it does not commute, block the lock requester.

Step 2) Locking on a target class

- If the lock request is an SCA, check conflicts with locks set by other transactions and set a lock on only the target class C if it commutes and set an *a* lock on the instance to be accessed if *a* method is invoked on the instance and commute. If it does not commute, block the requester.
- If the lock request is an MCA, then, from class C to the first *SC* (or leaf class if there is no SC) through the subclass chain of C, check conflicts and set a lock on each class if commute. If the class C is an SC, then set a lock only on C.

The reason to set a lock on each class (besides the first SC) from the class C to the first SC (not including the SC) is as follows: if a lock is set only on the first SC, then some conflict may not be detected. For example, if a requester accesses a subclass of a lock holder's class locked by MCA, then such a conflict may not be detected.

- If class C has more than one subclass, perform the same step 2) for each subclass chain of C.

For example, consider the class hierarchy as in Fig. 1.a. Also, assume that locks are requests by T_1 and T_2 as follows.

1) T_1: class definition update operation on class C6

2) T_2: class definition update operation on class C7

As in Fig 4.a, 4.b, 4.c and 4.d, 6, 7 11 and 13 locks are required for T_1 and T_2 by the proposed scheme, SC-based scheme, explicit locking, and implicit locking, respectively.

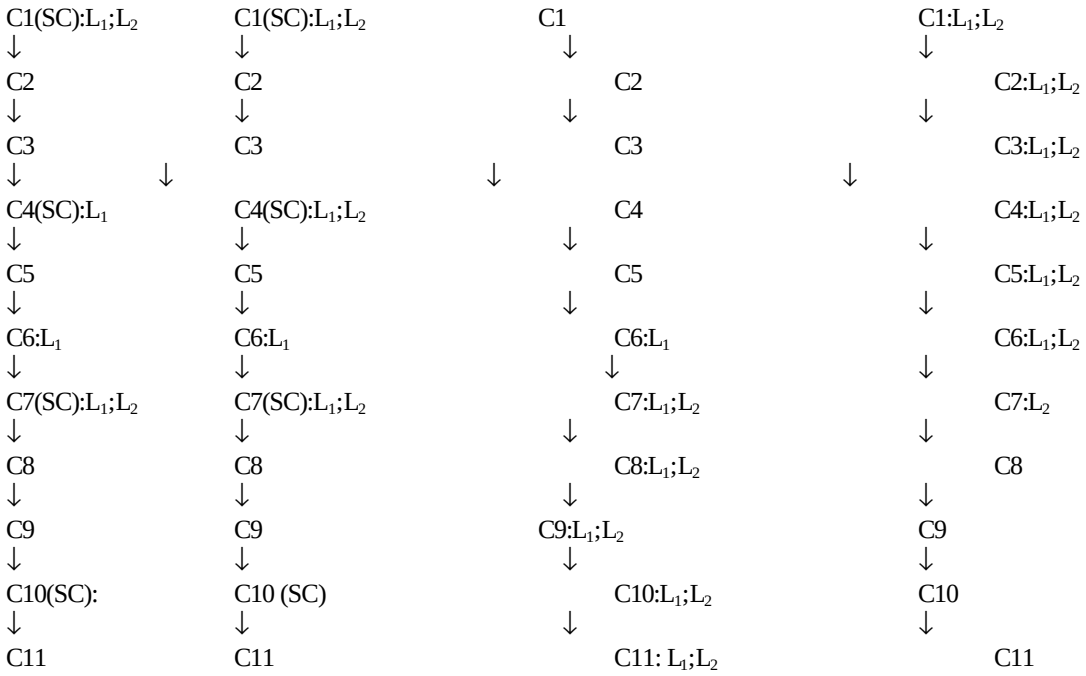

Fig 4.a. Locks by Proposed scheme | Fig. 4.b. Locks by SC-based scheme | Fig. 4.c. Locks by Explicit locking | Fig. 4.d. Locks by Implicit locking

4. Performance Evaluation of the Proposed Scheme

In this Section, we will show that the proposed scheme performs better than existing works, explicit locking, implicit locking and the SC-based scheme. It is shown that the SC-based scheme performs better than both explicit locking and implicit locking in [6]. Thus, it is sufficient to show that the proposed scheme performs better than only the SC-based scheme. Based on the discussion from Section 3.2, the proposed scheme incurs less than or equal number of locks than the SC-based scheme for any kinds of accesses to OODBs. Thus, in this Section, it is sufficient to prove that the proposed scheme is correct, that is, it satisfies serializability [1]. We prove this by showing that, for any lock requester, its conflict with a lock holder (if any) is always detected. With this proof, since our class hierarchy locking scheme is based on two-phase locking, it is guaranteed that the proposed scheme satisfies serializability.

Depending on the lock requester's type, lock holders can be divided as follows. If a lock requester is an SCA, then its lock holders (whose lock modes need to be checked for conflict with lock requester) consist of transactions holding locks on the target class and all SCs in the superclass chain of the target class. If a lock requester is an MCA, then its lock holders include those defined above plus transactions holding locks on each class from the target class to the first SC in the subclass chain of the target class.

There are four cases depending on the types of lock requesters and holders.

case 1) the lock holder is an SCA
the lock requester is an SCA

If a lock holder (H) and a lock requester (R) access different classes, there is no conflict. If a lock holder and a lock requester access the same class, the possible conflicts can be detected on the target class. This is due to the reason that there is no conflict on all SCs through the superclass chain of the target class since intention locks on SCs can be compatible with R.

case 2) the lock holder is an SCA
the lock requester is an MCA

Let C_R and C_H be two classes on which the R requests a lock and the H holds a lock, respectively. If C_H is a superclass of C_R, there is no conflict since the R does not access the C_H. If the C_H is C_R itself or its subclass, then there are two subcases. If there exists a SC which is a superclass of both C_R and C_H, then conflict is detected on the SC. (case 2.1). That is, in Fig. 5.a, the possible conflict is detected on SC1 since both R and H must have locks on SC1. Otherwise, the conflict is detected as follows. As in Fig 5.b, in case 2.2, if there is a SC between C_R and C_H, the conflict is detected on SC1 since C_R and C_H must have locks on SC1 based on the proposed scheme. On the other hand, if there is no such SC between C_R and C_H as in Fig. 5.c, the conflict is detected on C_H since R must have a lock on C_H.

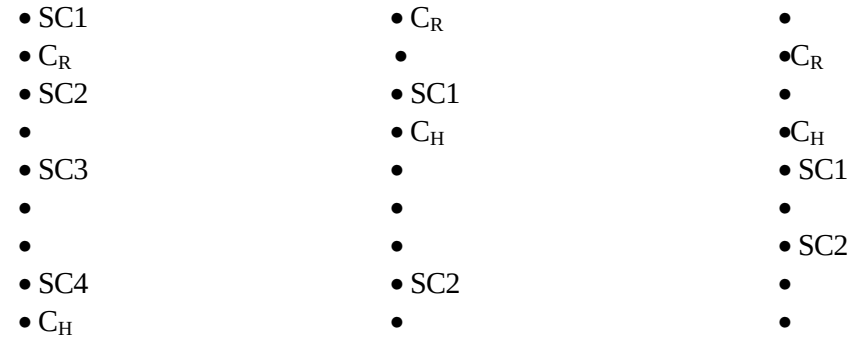

Fig 5. a. case 2.1

Fig. 5.b. case 2.2

Fig. 5.c. case 2.3

case 3) the lock holder is an MCA
the lock requester is an SCA

If the C_H is a subclass of the C_R, there is no conflict. If C_H is C_R itself or superclass of C_R, then there are two cases in which conflicts will be detected. If there exists a SC, which is a superclass of both C_R and C_H as in Fig. 6.a, then conflict is detected on the SC. (case 3.1). This is due to that H and R must a lock on the SC according to our scheme. Otherwise, there are two subcases. At first, if there exists a SC between C_H and C_R, the possible conflict is detected on the first SC through the subclass chain of C_H. For example, in Fig. 6.b. the conflict can be detected on SC1. If there is no SC between C_H and C_R as in Fig. 6.c, the conflict is detected on C_R since C_H must set a lock on the class C_R.

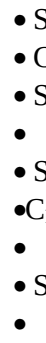

Fig 6. a. case 3.1

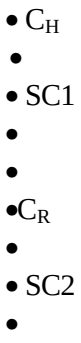

Fig. 6.b. case 3.2

Fig. 6.c. case 3.3

case 4) the lock holder is an MCA
the lock requester is an MCA

If the C_H is C_R itself or superclass of C_R, the conflict is detected as in case 3. Otherwise, the conflict is detected as in case 2.

From cases 1), 2), 3) and 4), we can conclude that, for any lock requester, it is guaranteed that its conflict with a lock holder (if any) is always detected. Also, since the proposed scheme is based on two-phase locking, serializability is guaranteed [1]. In turn, this means that the proposed scheme performs better than existing schemes, explicit scheme, implicit scheme and the SC-based scheme.

5. Further work

In this paper, a locking-based concurrency control scheme is presented for object-oriented databases (OODBs). It is designed for controlling accesses to class hierarchy, which is important concept in OODBs. Based on access frequency for each class, the scheme incurs less locking overhead than existing works, explicit scheme and implicit scheme and the SC-based scheme, for any OODB environments. In this paper, it is theoretically proven that the proposed scheme performs better than existing schemes.

Currently we are developing a concurrency control scheme for controlling access to composite object hierarchy, which is also major aspect in OODBs. Our goal is to combine our class hierarchy scheme with composite object scheme. Also, we will conduct the performance evaluation study in order to compare our work with existing schemes using either simulation or analytical model.

References

[1] Bernstein P, Hadzilacos V and Goodman N, *Concurrency Control and Recovery in Database Systems*, (Addison-Wesley, 1987).

[2] Cart M and Ferrie J, Integrating Concurrency Control into an Object-Oriented Database System, 2nd Int. Conf. on Extending Data Base Technology, Venice, Italy, (Mar. 1990), 363 - 377.

[3] Date, C., *An Introduction to Database Systems*, Vol. II, (Addison-Wesley, 1985).

[4] Eswaran K, Gray J, Lorie R and Traiger I, The notion of consistency and predicate locks in a database system, Communication of ACM, Vol. 19, No. 11, (Nov., 1976), 624 - 633.

[5] Garza J and Kim W, Transaction Management in an Object-Oriented Database System, ACM SIGMOD Int. Conf. on Management of Data, Chicago, Illinois, (Jun., 1988), 37 - 45.

[6] Jun W and Gruenwald L, An Effective Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems, Journal of Information And Software Technology, Vol. 40. No. 1, (Apr. 1998) 45-53.

[7] Lee L and Liou R, A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems, IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 1, (Feb. 1996), 144 - 156.

[8] Malta C and Martinez J, Controlling Concurrent Accesses in an Object-Oriented Environment, 2nd Int. Symp. on Database Systems for Advanced Applications, Tokyo, Japan, (Apr., 1992) 192 - 200.

[9] Malta C and Martinez J, Automating Fine Concurrency Control in Object-Oriented Databases, 9th IEEE Conf. on Data Engineering, Vienna, Austria, (Apr., 1993), 253- 260.