A CONCURRENCY CONTROL MODEL FOR MULTILEVEL SECURE OBJECT-ORIENTED DATABASES

William Perrizo[1], Hossein Hakimzadeh[2], Ramzi Haraty[3], B. Panda[4]

[1]Computer Science Department
North Dakota State University
Fargo, ND 58105
perrizo@plains.nodak.edu - (701) 237-8562

[2]Math. & Computer Science Dept.
Indiana University at South Bend
South Bend, IN 46634
hossein@natasha.iusb.indiana.edu - (219) 237-4517

[3]Computer Science Department
Moorhead State University
Moorhead, MN 56563
haraty@mhd1.moorhead.msus.edu

ABSTRACT

Much attention is being directed toward the development of secure database systems. Such systems are critical for both military as well as sensitive commercial applications. The majority of research in security and multi level secure database management systems (MLS/DBMS) are focused on relational systems. However with the emergence of new and complex applications of the 1990's, research in object oriented security is gaining more prominence [Thur90], [Keef89]. In this paper, we address the issues of transaction management for multilevel Secure Object-Oriented Database systems. We begin by introducing two new security policies specifically designed for OODB's. Later we identify the existence of covert *channels in the traditional conflict-serializable concurrency control schedulers and provide an alternative transaction processing algorithm which ensures both correctness and security.* We conclude with the proof of correctness.

Key Words: Object-Oriented Databases, Computer Security, Concurrency Control, Version Control, Commutativity, MLS/OODB.

INTRODUCTION

In order to enhance performance in a multilevel secure database environment, database applications must be allowed to interleave their execution. However concurrent execution gone unsupervised can lead to erroneous results and inconsistent database state. A database management system prevents such inconsistencies by enforcing a concurrency control strategy, which enhances performance and maintains correctness. In other words, the concurrency control strategy will only produce *serializable* schedules. Such schedulers will produce executions which are equivalent to some serial order. A number of methods such as time stamp ordering (TO) [Reed83] [Bern80], locking (2PL) [Eswa76] [Gray78b] [Bern87], serialization graph testing (SGT) [Bada79] [Schl78], tree locking (TL) [Silb80] [Baye77] [Bern87], optimistic certifiers [Kung81] [Robi82] and ROLL [Perr91] [Haki92b] have been proposed. Many of the existing relational and object-oriented database systems use these methods with varied levels of performance. In addition to the above methods a number of alternative concurrency control strategies have been proposed which seek to better accommodate object oriented applications. The focus of this paper is to point out the security breaches in existing concurrency control algorithms and develop a high performance concurrency control algorithm which is both, secure and correct.

OBJECT ORIENTED MODEL

An object is meant to represent a concept in the real world. Each object belongs (is an instance of) a single class. A class is viewed as having two parts, a *structure* and a *behavior*. The structure is the instance variables and the methods of the class define its behavior. Classes are encapsulated entities, and the public methods for each class provide the user interface to that class, hiding the implementation details. Classes can be either "base" or "derived". Derived classes inherit from one or more base classes. The set of classes in OODB are organized into a class hierarchy and the schema of each class includes the schema of all of its superclasses.

OBJECT ORIENTED DATABASE SYSTEMS

An object-oriented database is a system which provides all the functionalities of a traditional database such as persistence, integrity, transaction management, concurrency control, recovery, query processing and security, as well as object-oriented features such as data abstraction, encapsulation, inheritance, object identity, intelligence, versioning and better performance for complex applications.[Haki92c]

Data abstraction provides the necessary facilities for incorporating more complex data types such as images, voice segments, vectors, etc.. *Object Oriented Data Model* provides a better, more powerful, and often more efficient data model. Object oriented data modeling is closer to real world modeling and therefore, it is more intuitive. Information is modeled in the form of classes and objects which capture the *structure* and *behavior* of real world entities. OODB systems maintain unique *Object Identifiers* (OID) for every object. Therefore, eliminating the need for arbitrarily primary keys. This feature solves many integrity issues and speeds up the database access. The access speed is improved due to the reduction of expensive joins on attributes. *Encapsulation* couples the data and its associated operations in a atomic unit. This practice has the benefit of hiding the details of the data from the user. Further more, the implementation of the operations (methods) may be modified without invalidating the applications which use them. OODB systems are more *intelligent* than traditional databases. This is mainly due to encapsulation which gives the database the ability to *reason* about its domain, integrity, validation and consistency. This awareness in part of the OODB systems *triggers* appropriate methods to deal with any possible problems. *Inheritance* encourages data and code reusability and incremental development. Organizing generalized classes at the top of the hierarchy and deriving specialized classes from them allows us to incrementally augment/extend the database functionality. *Versioning* provides the ability to maintain multiple versions of each object allowing design teams to speculate with what if scenarios. Better *performance*, is often achieved by applications which need to display complex objects. Such applications can perform two to three orders of magnitude better in an object oriented environment[Edel91]. This is due to the fact that, it is much easier and faster to follow pointers than to join multiple tables.

BACKGROUND ON OBJECT ORIENTED CONCURRENCY CONTROL

In the past few years, numerous extensions to existing traditional concurrency control algorithms have been proposed in the context of object oriented database systems. The following is a short review of each method.

Object and Class Level Locking [Barg91] [Hugh91] or otherwise known as multi-granularity locking was first described in the framework of tree locking. Variations of this method is now applied to OODB's. **Non-Serializable Approaches to Concurrency Control** [Hugh91] relax serializability, however they certain constraints on the operation and behavior of the transactions. Examples of such methods are *linearizability*, *commit-serializability*, *nested-transactions* and Sagas. **Linearizability** requires transactions to be reduced to a *single operation* on a *single object* [Hugh91]. This implies that complex transactions must be decomposed into a number of smaller transactions. Once the transaction is decomposed, the smaller transactions can execute concurrently. **Commit Serializability (CS)** [Hugh91] requires transactions to commit in a serializable order, however, these transactions need not directly correspond to the initial transactions. The proposal is to split the transaction in to (read-set and write-set) and form new transactions. **Nested Transactions** [Reed78] [Hugh91] [Gray93] allows a transaction to spawn concurrent child transactions, however the interaction between the sub-transactions are serialized by the parent. The parent transaction governs all of its child sub-transactions and no child transaction can commit until its parent has committed. **Sagas** [Barg91] [Garc87] was introduced to solve long transactions and is very similar to nested transactions (transactions are broken down to sub-transactions). Sagas is not a serializable algorithm and has the possibility of cascading aborts. The notion of roll-back, is replaced by compensating transactions (functions) which are supplied by the user. The compensating transaction will semantically undo the original transaction. **Version Control & CC** [Agra89c] [Bjor89] [Barg91] [Scio91] are algorithms which combine version control with one or more traditional concurrency control algorithms such as BTO, Optimistic, or 2PL in order to enhance concurrency. Traditionally, multiple versions were used to allow read only transactions, fast and consistent access to the database. Read-only transaction that conflict with other transaction would simply read the previous version therefore, allowing more concurrency. Version control algorithms are also popular with current object-oriented database applications. Transactions in such applications are often long-lived and therefore, the potential for conflicts become greater. Versioning provides an alternative to restarting. **Commutativity & CC** [Weih88] [Naka92] are semantically driven algorithms which ensure serializability of transactions by using conflict relations based on the commutativity of operations. Two operations are required to conflict only if they do not commute therefore, concurrent transactions can access and update the same object as long as their operations commute.

SECURITY RELATED ISSUES

There are two standard types of security in database systems: *discretionary* and *mandatory* security. Discretionary security restricts access to data items at the discretion of the owner. Most commercial database management systems employ some form of discretionary security by controlling access privileges and modes of users to data [Grif76]. Discretionary security is not adequate in a multilevel secure environment however, because it does not prevent Trojan horse attacks and provides a low level of assurance. Mandatory security restricts access to data items to cleared database users. It is widely employed in military applications and provides a high level of assurance.

Numerous commercial and military applications require a MLS/OODB. In an MLS/OODB, database users are assigned classifications levels, and data items are assigned sensitivity levels. It is the responsibility of the MLS/OODB to ensure that users can access only those data items for which they have been granted a clearance.

The goal of this paper is to address the issue of transaction management in a MLS/OODB. We will identify a set of security constraints and later describe a transaction processing model which interacts with these security constraints in order to achieve a high performance MLS/OODB.

THE SECURITY MODEL FOR OODBs

We use the standard military security approach which consists of two components. A set of security classes and a set of non-hierarchical compartments. The security classes are totally ordered from the lowest to the highest as follows: unclassified $<<$ confidential $<<$ secret $<<$ top secret. Within each security class there can be zero or more compartments (for example, conventional, chemical, and nuclear).

We say that a security class, S1, is dominated by another class, S2, if S2 is hierarchically higher than S1 and contains all of its compartments.

We refer to users, or the processes that execute on behalf of users, as subjects. Users are trusted, but processes are not. Objects, on the other hand, correspond to data items. The Bell-LaPadula model defines two security policies commonly accepted in a system that enforces multilevel security [Bell76]:

A. The Simple Security Policy: A subject is allowed read access to an object if the subjects classification level is identical to or higher than that of the object's sensitivity level.

B. The *-Policy: A subject is allowed write access to an object if the subject's classification level is identical to or lower than that of the object's sensitivity level.

These policies, although important, are not complete for an object oriented setting. We propose two new security constraints which can be summarized in the following policies:

C. The Class Security Policy: The sensitivity level of a class must be identical to or lower than the sensitivity level of its subclasses and identical to the subjects classification.

D. The Instances Security Policy: The sensitivity level of all instances (objects) of a class must be identical to or higher than that of its class.

These polices guarantee that proper access to objects will not be violated directly. However, they are insufficient to guarantee indirect violations through covert channels. *Covert channels* are channels that are not intended to route information through, but nevertheless they do [TCSEC85]. There are three main types of covert channels: covert *storage channels* and covert *timing channels* and signaling channels.

Covert *storage channels* can disclose implicit information from high to low subjects through manipulation of a physical object. This manipulation can be in the form of creation or destruction of a given persistent object.

Covert *timing channels* can covertly send information from high to low subjects by modulating an observable delay in the accessing of a common resource. A system that is free from covert channels is called *Covert Channel Secure (CCS)*. This is the strongest form of security [Keef90].

A *Signaling Channel* is a means of information flow inherent in the basic algorithm or protocol, and hence appears in every implementation. Note that a covert channel is a property of a specific implementation, and not the general algorithm or protocol [Jajo92].

OBJECT ORIENTED DATABASE TRANSACTION MODEL

Our database model consist of the four components, subjects, objects, classes, methods and instance variables. Similar to relational modal, subjects (users) are given a security level and object (data items) are given a sensitivity level. In addition to subjects and objects, our model includes the notion of a class and its methods.

- Subject (s-id, Authorization)
- Object (o-id, Authorization)
- Class (c-id, Authorization)
- Method (Same as the authorization of the objects or the subject which ever is higher)
- Instance Variables(same as object's authorization)

Our transaction model consists of three components (M, C, O) where M stands for Methods, C stands for Classes and O stands for Objects. A transaction is described as the invocation of a set of one or more methods. Each method acting on a given object of a given class.

Method(Class, Object)

For the purpose of formalizing concurrency control, methods are ultimately classified as *read,* or *write* operations. Traditional, concurrency control algorithms based on "conflict serializability" disallow conflicting operations on the same data item by different transactions[Kort90]. The compatibility matrix for read and write operations is as follows:

	read	write
read	N	Y
write	Y	Y

Note that the only operations that do not conflict are two reads. It is clearly evident that conflict serializability introduces a covert channel in a MLS/OODB environment (due to waiting by local transactions for resources held by read-down transactions). In order to eliminate covert channels we propose a new commutativity matrix which distinguishes between the security classification (i.e. unclassified, classified, secret, top-secret) of operations. In other words, a read operation is defined as either read-local or read-down. If the classification of the subject and the sensitivity of the object match, the operation is considered as a read-local however, if the classification of the subject is higher than the sensitivity of the object, then the operation is considered as a read-down. The matrix below represent our commutativity protocol.

Commutativity Matrix for Resolving Covert Channels

	read-local	**write-local**	**read-down**
read-local	**n**	**y**	**n**
write-local	**y**	**y**	**n if(r-d < w-l)**
read-down	**n**	**n if(r-d < w)**	**n**

n = no conflict
y = conflict

Note that the order of operation in the above matrix makes a difference such that (read-down followed by a write-local) and (write-local followed by a read-down) is not symmetric. This notion is called forward vs. backward commutativity and is described in [Weih88].

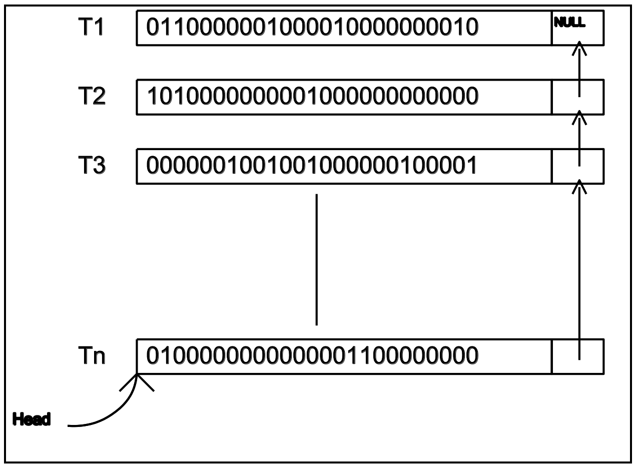
ELIMINATING COVERT CHANNELS

In order to eliminate the covert channels we must come up with an algorithm which eliminates the flow of information (of any kind) from higher to lower containers. This means that no operations of a lower classification can be directly effected by an operation from a higher classification. For example a w-local operation must never wait for a lock held by a r-down operation. This is a subtle yet extremely important issue since if a local transaction ever has to wait for a r-down transaction (an operation from higher container) we have established a direct and obvious covert channel. This allows the subject invoking a (w-local) to make an inference based on the lock held by the (r-down) operation (depending upon the amount of time it has to wait). Granted that the bandwidth and the accuracy of such covert channels may be low however, it is still not tolerated by the database security community.

In the following sections we will describe how ROLL concurrency control [Perr91] can be complemented by a simple yet effective versioning scheme to accommodate the security requirements of a MLS/OODBMS.

ORIGINAL ROLL CONCURRENCY CONTROL

ROLL concurrency control object is described in [Perr91] and implemented in [Haki92b]. Roll is a pessimistic, non-blocking, deadlock and restart free concurrency control algorithm developed at North Dakota State University. The basic idea is to create an encapsulated concurrency control object which allows transaction managers to police and monitor their own concurrency. The ROLL object is simply a linked list of Request Vectors (one RV per transaction). Each bit in the vector corresponds to a lockable granule in the database. A 1 bit represents a lock request, and a 0 bit indicates, no lock is requested. The ROLL object also provides 3 public methods. The methods are POST, CHECK, and RELEASE.



```
Class ROLL {
        unsigned int    T_id;
        ROLL            *Up_link;
        Vector_type     *Roll_vector;
        Install(….);
        Clean(….);
        Free(….);
   Public:
        ROLL(T_id, vector_size, …..);
        Post(….);
        Check(…..);
        Release(…..);
}
```

Informally, the algorithm works as follows. Transaction T_1 composes a RV and POSTs it into the ROLL object. Immediately after POSTing, T_1 can invoke its CHECK operations. (Note: there is no scheduler) The CHECK operation simply checks the request vectors ahead of T_1 and returns an access vector (AV) which tells T_1 specifically which one of the data items are available. At this point each transaction will acquire all available items that it has requested and only waits for those that are in conflict (No unnecessary waiting). Due to the sequential nature of the POSTing process, waits are never circular therefore, there are no deadlocks. Based on linear or exponential backoff, T_1 continues to perform periodic CHECK operations until all of its lock requests are granted.

Finally depending upon the concurrency control and recovery policy adopted by the system the RELEASE operation is called to unlock the data items. As one can see, the scheduler is replaced with a series of concurrently executing transaction managers. The POST operation is the only operation which requires atomicity. The other operations such as CHECK and RELEASE can be executed concurrently. Detailed information regarding advantages and disadvantages of ROLL concurrency control can be found in [Perr91].

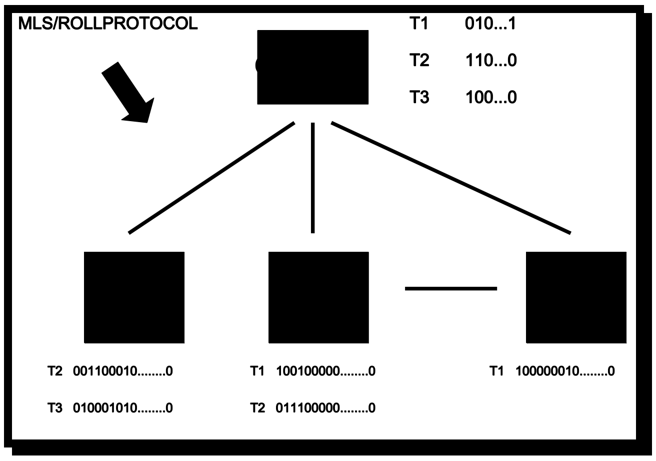
The proposed multilevel model is based on two layers. The first layer is the TCB and the second layer is comprised of local Request Order Linked List (LROLL) driver objects at each container [Perr91]. The TCB authenticates users and determines whether returned data is to be viewed by the querying users or not. The local ROLL objects provide correctness assurance using global serializability as the correctness criterion. There is one ROLL object in the TCB, the Global ROLL and a separate ROLL object at each container, the Local ROLLs.

COMBINING MULTI-VERSIONING AND ROLL CONCURRENCY CONTROL

The basic objective in combining multi-versioning and ROLL concurrency control is to eliminate any covert or signaling channels which may exist in conflict serializable algorithms and provide a highly concurrent and secure algorithm. In multi-version ROLL local read or write operations will never have to wait for any operations initiated from a higher level container.

In the following section we will describe the method by which the ROLL algorithm is modified to accommodate multi-level security constraints under the kernelized architecture.

- Transactions are categorized as *local* (intra-security-level) or *global* (inter-security-level).
- Intra-level transactions can be *read-local* and/or *write-local.*
- Inter-level transactions can be any combination of *read-local, write-local* and *read-down*.



POST Protocol

1) Inter-security-level transactions must first POST to a trusted GROLL structure. Followed by one or more POSTs to various LROLL structures. At the GROLL level, transactions specify the containers they are interested in, by creating a request vector and placing a 1 bit for the appropriate container. (i.e. RV = 0101, the transaction is requiring access to container 2 and 4)
2) Intra-security-level transactions must only POST to their local LROLL structure.
3) Once the POSTing is completed the serialization partial order is established.

CHECK Protocol

Once the inter-security-level transaction POSTs at the GROLL, it must perform a CHECK operation. The CHECK at the GROLL will return an access vector (AV) indicating which containers (if any) are available for further POSTing.

1) At the GROLL level, the conventional POST, CHECK and RELEASE described earlier is

observed.

2) At the LROLL level, the CHECK process is modified to eliminate the possibility of any covert timing or signaling channels.

3) The CHECK process is modified such that local operations will never have to wait for a global (i.e. read-down) operation.

This is implemented as follows:

When a CHECK by a local transaction is initiated the CHECK method will return an Access Vector (AV) indicating which if any of the items are available for use by the transaction. The CHECK process is designed to distinguish between local and non-local Request Vectors (RVs) and if there exists a conflicting global transaction, the versioning mechanism is invoked. The versioning mechanism will check the currency (if the data item is current or old) of the data item, if the value is current, the local transaction is allowed to write the data item and the system will manage the version control.

If the result of the check operation reveals the existence of a conflict between two local transactions, the conventional CHECK operation is performed.

RELEASE Protocol

In a multi-version environment, the RELEASE operation is modified to accommodate both local and global transaction.

1) The RELEASE at the GROLL may be done in a container at a time basis. In other words after the transaction POSTs its appropriate LROLL in the desired containers, it may release the bit for that container, allowing others to proceed.

2) The RELEASE at the LROLL by a local operation (r-local, w-local) is done as an strict, and atomic operation. This behavior will ensures recoverability.

3) The RELEASE by a read-down operation may be invoked discarding the old version of the object.

PROOF OF CORRECTNESS

THEOREM 1: The ROLL concurrency control object produces only serializable executions:

PROOF: A transaction may not access an object until all preceding, conflicting transactions in the ROLL have accessed and released it. Every method performed by a transactions which conflicts with a younger transaction methods (i.e. has POSTed after) must follow the methods of that transaction. Thus every execution partial order is compatible with the POSTing order which is serial.

Notation:

- A version order, for some data item x in history H, is a total order of versions of x in H and is denoted by $<_{vo}$.
- $r_i[x_j]$ represents that transaction T_i has read a data item written by transaction T_j.
- $w_k[x_k]$ represents that the version k of x is written by a (committed) transaction T_k.
- C(H) denotes the history over the set of committed transactions in H.

Definition:

Given a Multi-Version (MV) history, H, and a version order, $<_{vo}$ (the union of the version orders for all data items), the Multi-Version serialization graph for H, MVSG(H), is a directed graph whose nodes are the committed transactions in H and there are two types of edges:

a) $T_i \rightarrow T_j$ $(i \neq j)$ whenever for some x, T_j reads x from T_i, i.e., $r_j[x_i]$ is in C(H).
b) For each $r_k[x_j]$ and $w_i[x_i]$ in C(H) where i,j,k are distinct, if $x_i <_{vo} x_j$ then the edge $T_i \rightarrow T_j$ is in MVSG(H), otherwise the edge $T_k \rightarrow T_i$ is in MVSG(H).

Theorem:

A MV history H is one-copy serializable (1SR) iff there exists a version order, $<_{vo}$, such that MVSG(H) is acyclic [Bern87].

The following theorem establishes the correctness of our algorithm.

Theorem:

Let H be any Multi-Version history produced by our algorithm. Then the MVSG(H) is acyclic and hence H is 1SR.

Proof:

To prove that MVSG(H) is acyclic, we first show that every edge in MVSG(H) follows the post order of transactions. That is, if $T_i \rightarrow T_j$ is in MVSG(H), then $PO(T_i) < PO(T_j)$ where $PO(T_i)$ and $PO(T_j)$ denote the post order of transactions T_i and T_j respectively.

The following properties of our algorithm would help in establishing the proof.

1) All global transactions must POST in the GROLL and LROLLs in order to access any data items.
2) The POST order is serial.
3) No transaction can access any data item until a conflicting transaction having less POST order releases all the bits in its request vector in LROLL.
4) Transactions RELEASE the bits in request vector during commit time only.

Now, let us consider an edge $T_i \rightarrow T_j$ in MVSG(H) for the operation $r_j[x_i]$ in H for some data item x. By properties 3 and 4 above, T_i commits before T_j reads x. Thus $PO(T_i) < PO(T_j)$.

Next, let us consider a version order edge in MVSG(H) due to two different versions, x_i and x_k of x. If the operation $r_j[x_i]$ is in H, then the edges

a) $T_k \rightarrow T_i \rightarrow T_j$ is in MVSG(H) if $x_k <_{vo} x_i$

b) $T_i \rightarrow T_j \rightarrow T_k$ is in MVSG(H) if $x_i <_{vo} x_k$

We need to show that in case a, $PO(T_k) < PO(T_i) < PO(T_j)$, and in case b, $PO(T_i) < PO(T_j) < PO(T_k)$.

In case a, since T_i can write x, only after T_k RELEASEs all its bits, obviously, $PO(T_k) < PO(Ti) < PO(T_j)$. In case b, let us assume that $PO(T_i) < PO(T_k) < PO(T_j)$. But since T_i, T_j, and T_k are conflicting transactions, the commit order of T_i, T_j, and T_k is the same as their post order. But then as per our algorithm, T_j would read the highest available version, which in this case would be x_k, not x_i, of x.

This contradicts our assumption that $r_j[x_i]$ is in H. Thus, we have proven that every edge in MVSG(H) follows the POST order of transactions that represent the nodes.

Since a transaction POSTs only once in an LROLL, and the POST partial orders across containers is the same for the same set of transactions (this is maintained by the GROLL), there would never be a cycle in the post order. Therefore, there would never be a cycle in the MVSG(H). This completes the proof.

CONCLUSION

We have provided two new security policies, specifically developed for OODB's. These policies ensure the security constraints of the Bell-LaPadula Model. We have also identified the existence of covert channels in the traditional concurrency control algorithms, and provided an alternative transaction processing algorithm which ensures both correctness and security. This was achieved using the ROLL concurrency control combined with a modified version control strategy.

REFERENCES

[Atki92] Atkins, M. S., Coady M. Y., "Adaptable Concurrency Control for Atomic Data Types", ACM Transactions on Computer Systems, Vol. 10, No. 3, Aug. 1992. (pp 190-225)

[Badr92] Badrinath, B. R., Ramamritham, K., "Semantics-Based Concurrency Control: Beyond Commutativity", ACM Trans. on Database Systems, Vol. 17, No. 1, March 1992.

[Barg91] Barghouti, N. S., Kaiser, G. E., "Concurrency Control in Advanced Database Applications", ACM Computing Surveys, Vol. 23, No. 3, Sept. 1991.

[Bell76] Bell D.E., Lapadula, L.J., "Secure Computer System: Unified Exposition and Multics Interpretation" Tech. Report MTR-2997, Mitre Corp., Bedford, Mass., March 1976, Available as NTIS AD A023588.

[Bern80] Bernstein, P. A., Goodman, N., "Timestamp-based Algorithm for Concurrency Control in Distributed Database Systems", Proceedings of the International Conference on Very Large Databases, 1980. pp 285-300.

[Bern87] Bernstein, P. A., Hadzilacos, V., Goodman, N., "Concurrency Control and Recovery in Database Systems", Addison Wesley, 1987.

[Bjor89] Bjornerstedt a., Hulten C., "Version Control in an Object-Oriented Architecture, in Object-Oriented concepts, Databases and Applications", Addison-Wesley, Reading, Mass., 1989, pp 451-485

[Eswa76] Eswaran, K. P., Gray, J.N. Lorie, R.A., Traiger, I. L., "The Notions of Consistency and Predicate Locks in a Database System", Communications of the ACM, Nov. 1976. pp 624-633

[Garc87] Garcia-Molina, H, Salem, K., "SAGAS", Proc. of the ACM SIGMOD, Annual Conf. (May), ACM Press, pp 249-259, 1987.

[Grif76] Griffiths P. P., Wade B.W., "An Authorization Mechanism for Relational Database Systems", ACM transactions on Databases Systems, Vol. 1, No. 3, 1976. pp 242-255.

[Gray78b] Gray J. N., "Notes on Database Operating Systems: An Advanced Course", Editors: R. Bayer, R. M. Graham and G. Seegmuller, Springer-Verlag, New

York, 1978.

[Gray93] Gray J., Reuter A., "Transaction Processing: Concepts and Techniques", Morgan Kaugmann Publishers, Inc. 1993.

[Haki92b] Hakimzadeh, H., "ROLL Concurrency Control", Computer Science Department, Technical report number: NDSU-TR-1992-03. North Dakota State University, Fargo, ND.

[Haki92c] Hakimzadeh, H., "Object Orientation Primer", Department of Computer Science, Technical Report. (NDSU-CSOR-TR-1992-20). North Dakota State University, Fargo.

[Haki92d] Hakimzadeh, H., "Object Centered Concurrency Control for Object Orientation Databases", Ph.D. in progress, Department of Computer Science, North Dakota State University, Fargo, ND.

[Haki93] Hakimzadeh, H., Perrizo, W., "Fine Granularity Locking for Object-Oriented Databases", ISCA conference proceedings, March 10-12, Washington D.C. (March 93)

[Hugh91] Hughes, J. G., "Object Oriented Databases", Prentice Hall International Series In Computer Science., 1991.

[Jajo92] Jajodia, S., Atluri, V., "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases", Proceedings of the 1992 IEEE symposium on Research in Security and Privacy, page 216-224, Oakland, CA, May 4-6, 1992.

[Keef89] Keefe, T. F., Tsai, W. T., Thuraisingham, M. B., "SODA: A Secure Object-Oriented Database System", Computers and Security, Vol. 8, No. 6, Oct. 89, Pg 517-533.

[Kers84] Kersten, M., Tebra, H., "Application of an Optimistic Concurrency Control Method", Software Practice and Experience 14, Feb. 1984.

[Kort90] Korth, H.F., Levy, E., Silberchatz, A., "A Formal Approach to Recovery by Compensating Transactions.", Proceedings of the VLDB-90, 1990, pp. 95-106.

[McLe90] McLean J., "The Specification and Modeling of Computer Security", IEEE Computer, Jan. 1990.

[Naka92] Nakajima, T., "Commutativity-Based Concurrency Control and Recovery for

Multiversion Objects", Pre-Proceedings of the International Workshop on Distributed Object Management. Edited by M. T. Ozsu, U. Dayal, P. Valduriez, August 18-21, 1992, Edmonton, Canada. 1992. pp. 101-119.

[Perr91a] Perrizo, W., "A Concurrency Control Object", Proc. of the IEEE Conf. on Data Engineering, April, 1991, Kobe, Japan.

[Perr91b] Perrizo, W., Rajkumar. J., Ram Prabhu, "HYDRO: A Heterogeneous Distributed Database System", Proc. of the ACM SIGMOD, 1991.

[Reed78] Reed, R., "Naming and Synchronization in a Decentralized Computer System", Ph.D. Dissertation, MIT Laboratory of Computer Science, MIT Tech. Report. 1978.

[Schl78] Schlageter, G., "Process Synchronization in Database Systems. ACM Trans. on Database Systems, pp 248-271, Sept. 1978.

[Scio91] Sciore, E., "Using Annotations to Support Multiple Kinds of Versioning in Object-Oriented Database Systems", ACM Trans. on Database Systems,Vol. 16, No. 3, Sept. 1991. pp 417-438

[Skar89] Skarra, A. H., Zdonik S. B., "Concurrency Control and Object-Oriented Databases", Edited by Kim, W. Lochovsky F. H., "Object-Oriented Concepts, Databases and Applications", ACM Press, Addison-Wesley Publishing Company. pp 395-419.

[TCSEC85] DOD, Trusted Computer Systems Evaluation Criteria. National Computer Security Center. 1985.

[Thur90] Thuraisingham, M.B., "Security in Object-Oriented Database Systems", Journal of Object-Oriented Programming, Vol 2., No. 6, Mar/Apr 1990.

[Weih88] Weihl, W. E., "Commutativity-Based Concurrency Control for Abstract Data Types", IEEE Trans. on Computers, Dec. 1988.

[Weih89] Weihl, W. E., "The Impact of Recovery on Concurrency Control", Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principals of Database Systems, Philadelphia, Pennsylvania, March 1989.