A CONCURRENCY CONTROL MODEL FOR MULTILEVEL SECURE OBJECT-ORIENTED DATABASES

William Perrizo
Computer Science Department
North Dakota State University
Fargo, ND 58105

Hossein Hakimzadeh
Math. & Computer Science Dept.
Indiana University at South Bend
South Bend, IN 46634

Ramzi Haraty
Computer Science Department
Moorhead State University
Moorhead, MN 56563

Brajendra Panda
Computer Science Department
College of West Virginia
Beckley, WV 25802

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 multilevel 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 describe a secure algorithm for transaction management in Multilevel Secure Object-Oriented Database Systems.

Key Words: Multilevel Security, Object-Oriented Database Systems, Concurrency Control, Multiversion Objects.

1. INTRODUCTION

In order to enhance performance in a database environment, transactions 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 the system. A few common concurrency control protocols [Bern87] are: time stamp ordering (TO), two phase locking (2PL), and serialization graph testing (SGT). However, each of these protocols uses a scheduler which is a bottleneck in the system. Besides, TO suffers from livelock, 2PL suffers from deadlock, and the SGT requires unbounded amount of memory space to store its data structures [Bern87]. A different type of concurrency control technique, known as Request Ordered Linked List (ROLL) object [Perr91] [Haki92a] was developed at North Dakota State University to eliminate the problems described above. Based on these methods, some concurrency control protocols have been developed [Jajo92] [Pand94] [Hara92] for multilevel secure relational database systems. A multilevel secure database system (MLS/DBS) protects data from both direct and indirect unauthorized accesses. In this paper, a concurrency control algorithm for multilevel secure object oriented database systems has been developed that uses the ROLL protocol to ensure both correctness and security in the system.

2. OBJECT ORIENTED MODEL AND OBJECT ORIENTED DATABASE SYSTEMS

An object is meant to represent a concept in the real world. Each object belongs to (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.

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 [Haki92b].

3. THE SECURITY MODEL FOR OODBs

As mentioned earlier, the concurrency control algorithm described in this paper is meant for a multilevel secure object oriented database system. In a system that uses multilevel security, users, or the processes that execute on behalf of users are referred to as *subjects*, and data items are referred to as *objects*. Each subject is assigned a *clearance*, and each object is given a *classification level*. Both classification and clearance consists of two components: a security class and a set of non-hierarchical compartments. The security class is chosen from a totally ordered set such as: unclassified, confidential, secret, and top secret, unclassified being the lowest and top secret being the highest class. An example of a set of compartments are: conventional, chemical, and nuclear. The reader should note that a subset of the set of compartments (including the null set) is included in a classification or clearance. For the rest of the paper, we write security class instead of a classification or clearance. A security class, S_1, is dominated by another class, S_2, if S_2 is hierarchically higher than S_1 and contains all of S_1's compartments. A famous model, the Bell-LaPadula model [Bell76], commonly used in a multilevel secure environment, defines two security policies to enforce access rights to objects by subjects. They are:

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

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

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.

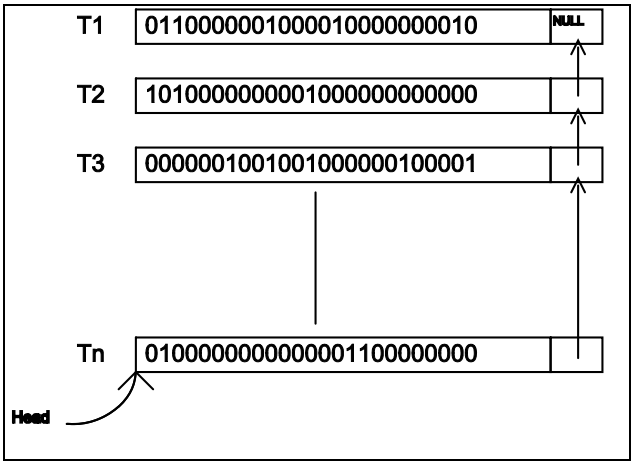
These polices guarantee that proper access to objects will not be violated directly. However, they are insufficient to guarantee indirect violations through covert channels. *Covert channels* are

channels that are not intended to route information through, but nevertheless they do [TCSEC85]. There are two important types of covert channels known as covert *storage channels* and covert *timing channels.* A covert *storage channel* 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. A covert *timing channel* can covertly send information from high to low subjects by modulating an observable delay in accessing a common resource. A secure system must also guard against covert channels.

4. ORIGINAL ROLL CONCURRENCY CONTROL

ROLL concurrency control object is described in [Perr91] and implemented in [Haki92a]. ROLL is a pessimistic, non-blocking, deadlock free, 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 (RVs) with 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 data items are available. At this point, each transaction will acquire all available items that it has requested and will only wait 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.

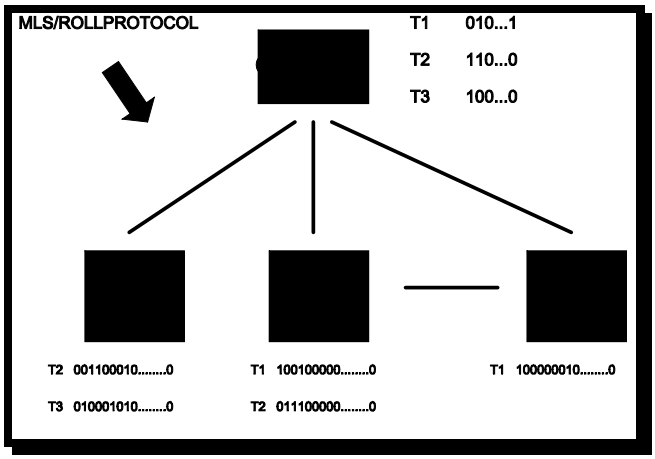
Finally, depending upon the concurrency control and recovery policy adopted by the system, the RELEASE operation is called to unlock the data items. As one can see, the scheduler is replaced with a series of concurrently executing transaction managers. The POST operation is the only operation which requires atomicity. In the next section, we describe the concurrency control

algorithm, called MLS-OOROLL, for a multilevel secure object-oriented database system.

5. MULTIVERSION MLS-OOROLL PROTOCOL

The proposed algorithm is based on two layers. The first layer is the TCB and the second layer is comprised of local Request Ordered Linked List (LROLL) driver objects at each container. The TCB, by using a ROLL object called the Global ROLL, authenticates users' accesses to containers. The global ROLL along with the local ROLLs provide correctness assurance using global serializability as the correctness criterion.

The multiversioning of data is utilized, since, it provides concurrent write access to a *low* user and read access to a *high* user for the same object. Otherwise, having a low user wait for a high user would establish a covert channel, and having a high user wait for a low user is undesirable, since, quick decision making at a higher level is extremely important. In the following section, we will describe the method by which the ROLL algorithm is modified to accommodate multilevel security constraints. We categorize a transaction as *local* if its access is limited to objects at the same level of the transaction (i.e., intra-security-level), or as *global* if the transaction requires access across containers (inter-security-level). Local transactions can be *read-local* and/or *write-local.* Global transactions can be any combination of *read-local, write-local* and *read-down*.



5.1 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. For example, an RV = 0101 means that the transaction is requiring access to containers 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.

5.2 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 are observed.

2) At the LROLL level, the CHECK process is modified to eliminate the possibility of signaling channels. 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 if the object is current. If the object 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.

5.3 RELEASE Protocol

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

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 ensures recoverability.

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

5.4 THE ALGORITHM

The MLS-OOROLL concurrency control algorithm is enumerated below. However, proof of correctness of this algorithm is omitted in this paper due to space constraints. Inquiries regarding the proof should be directed to the authors.

1. When a local transaction, T_i, enters the system for execution, its RV will be POSTed in the LROLL that exists at the same level of T_i.

2. When a global transaction, T_j, enters the system for execution, a Global RV (GRV) is created in which a 1 bit is set for each local container where access is needed. The GRV is posted in the GROLL.

3. The GROLL is CHECKed periodically to see if any of the containers requested by T_j are available.
   - 3.1. If yes, then transaction manager of T_j, denoted by TM_j, is informed to POST its RV in LROLL(s).
   - 3.2. If no, GROLL is reCHECKed again.

4. When a transaction, T_j, POSTs its RV in an LROLL, the corresponding bit in the GROLL is RELEASEd. This process continues until all the bits in the GRV of T_j are RELEASEd.

5. When a transaction, T_j, wants to read an object x, TM_j CHECKs the LROLL at the security level of object x.
   5.1. If a conflict exists, then TM_j waits to reCHECK later.
   5.2. If there is no conflict, then T_j reads object x.

6. When a transaction T_j wants to write an object x, TM_j CHECKs the LROLL at the security level of x. This time, the CHECK operation works as follows:
   6.1. If a conflict arises due to a read or write operation by a transaction at the same level of T_j, TM_j waits to reCHECK later.
   6.2. If a conflict arises due to a read request by a transaction at a higher level, TM_j is not informed about the conflict and is allowed to write the object. However, a new version of the object is created. This eliminates the possibility of establishing a covert channel.
   6.3. If no conflict exists, TM_j writes the object and no new version is created.

The reader should note that a transaction is not allowed to notice different versions of data objects at own level; otherwise, a covert channel would be established.

6. CONCLUSION

In this paper, we have provided two new security policies specifically developed for OODBs. These policies are required since the security constraints of the Bell-LaPadula model would not be enough to prevent from covert channels. Then, we have provided an alternative transaction processing algorithm which ensures both correctness and security. This was achieved using the ROLL concurrency control technique combined with a modified version control strategy. Unlike other concurrency control protocols, our algorithm is free from deadlocks, free from livelocks, and also eliminates the use of a scheduler that becomes a bottleneck in the system. We believe this algorithm represents a significant improvement over other algorithms developed for maintaining concurrency and security in a multilevel secure object-oriented database management system.

REFERENCES

[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.

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

[Haki92a] Hakimzadeh, H., "ROLL Concurrency Control", Computer Science Department, Technical report (NDSU-TR-1992-03). North Dakota State University, Fargo, ND.

[Haki92b] Hakimzadeh, H., "Object Orientation Primer", Department of Computer Science, Technical Report (NDSU-CSOR-TR-1992-20). North Dakota State University, Fargo, ND.

[Hara92] Haraty, R., "Transaction Management in Multilevel Secure Database Systems", Ph.D. Dissertation, North Dakota State University, Fargo, ND.

[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.

[Pand94] Panda, B., Perrizo, W., and Haraty, R., "Secure Transaction Management and Query Processing in Multilevel Secure Database Systems", Proceedings of the ACM-SIGAPP 1994 Symposium on Applied Computing, March, 1994, Phoenix, AZ.

[Perr91] Perrizo, W., "A Concurrency Control Object", Proc. of the IEEE Conf. on Data Engineering, April, 1991, Kobe, Japan.

[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.