# Multilevel Secure Rules and its Impact on the Design of Active Database Systems

Indrakshi Ray

Colorado State University, Fort Collins, CO 80523
iray@cs.colostate.edu

**Abstract.** The event-condition-action paradigm (also known as *triggers* or *rules*) gives a database "active" capabilities – the ability to react automatically to changes in the database or in the environment. One potential use of this technology is in the area of multilevel secure (MLS) data processing, such as, military, where the subjects and objects are classified into different security levels and mandatory access control rules govern who has access to what. Unfortunately, not much work has been done in the area of multilevel secure active database system. In this paper we define the structure of MLS rules and identify what effects these rules have on the execution semantics of an active database system. Such knowledge is essential before developing a multilevel secure active database system.

## 1 Introduction

Traditional database management system are *passive*: the database system executes commands when requested by the user or application program. However, there are many applications where this passive behavior is inadequate. Consider for example, a financial application: whenever the price of stock for a company falls below a given threshold, the user must sell his corresponding stocks. One solution is to add monitoring mechanisms in the application programs modifying the stock prices that will alert the user to such changes. Incorporating monitoring mechanisms in all the relevant application programs is non trivial. The alternate option is to poll periodically and check the stock prices. Polling too frequently incurs a performance penalty; polling too infrequently may result in not getting the desirable functionalities. A better solution is to use active databases.

Active databases move the reactive behavior from the application into the database. This reactive capability is provided by *triggers* also known as *event-condition-action rules* or simply *rules*. In other words, triggers give active databases the capability to monitor and react to specific circumstances that are relevant to an application. An active database system must provide trigger processing capabilities in addition to providing all the functionalities of a passive database system.

One potential use of this technology is in the area of secure data processing, such as, the military which uses an underlying multilevel secure (MLS) database. A multilevel secure database system is characterized by having a partially ordered set of security levels (the ordering relation is referred to as the dominance relation); all the database objects and the operations (transactions) on the database objects have security levels

associated with them. Mandatory access control policies determine which transactions can access which objects. The idea is that information can flow from the dominated level to the dominating level but not in the other direction.

To the best of our knowledge, the only work in providing active capabilities to an MLS database is done by Smith et al. [33]. Smith's work is based on the MLS relational model that supports polyinstantiation. Since this relational model is not widely prevalent, the MLS active database system proposed by Smith is not used either. The absence of a suitable MLS active database limits the potential use of active technology to applications that use an MLS database as the underlying database. This paper aims at filling this gap.

Providing reactive capabilities in a MLS database system has two components. First, we must provide a knowledge model for triggers. The knowledge model will specify what kinds of rules can be supported, what can be said about these rules, and how can the rules be classified into security levels. Next, we must provide an execution model for the triggers. The execution model will specify the runtime strategies for rule execution. The execution model must ensure that no illegal information flow occurs because of the execution of triggers.

This paper has two contributions. Our first contribution is that we focus on the knowledge model of an MLS active database system. Specifically, we show how to define an MLS rule, and how to assign security levels to such rules. Our second contribution illustrates the different execution models in existing active database systems, and the impact our MLS rules have on these execution models. Specifically, we identify those choices in execution model that can be supported without violating the MLS constraints. This knowledge is essential before developing an MLS active database system.

The rest of the paper is organized as follows. Section 2 briefly describes the underlying MLS model on which our work is based. Section 3 proposes the knowledge model of an MLS active database system. Section 4 identifies the impact our knowledge model has on the execution model. Section 5 describes the possible architectures for an MLS active database. Section 6 describes the relevant related work in this area. Section 7 concludes the paper with some pointers to future directions.

## 2 Our Model of an MLS Database

A database system is composed of database objects. At any given time, the *database state* is determined by the values of the objects in the database. A change in the value of a database object changes the state. Users are responsible for submitting transactions and application programs that are to be executed on the database. A *transaction* is an operation on a database state. An *application program* may or may not perform an operation on the database state. Execution of a transaction or application program may cause the database to change state.

An MLS database is associated with a security structure that is a partial order, (**L**, $<$). **L** is a set of security levels, and $<$ is the dominance relation between levels. If $L_1 < L_2$, then $L_2$ is said to strictly dominate $L_1$ and $L_1$ is said to be strictly dominated by $L_2$. If $L_1 = L_2$, then the two levels are said to be equal. $L_1 < L_2$ or $L_1 = L_2$ is denoted by $L_1 \leq L_2$. If $L_1 \leq L_2$,then $L_2$ is said to dominate $L_1$ and $L_1$ is said to be dominated by

$L_2$. Two levels $L_1$ and $L_2$ are said to be incomparable if neither $L_1 \leq L_2$ nor $L_2 \leq L_1$. We assume the existence of a level $U$, that corresponds to the level unclassified or public knowledge. The level $U$ is the greatest lower bound of all the levels in **L**. Any data object classified at level $U$ is accessible to all the users of the database.

Each database object $x \in \mathbf{D}$ is associated with exactly one security level which we denote as $L(x)$ where $L(x) \in \mathbf{L}$. (The function $L$ maps entities to security levels.) We assume that the security level of an object remains fixed for the entire lifetime of the object.

The users of the system are cleared to the different security levels. We denote the security clearance of user $U_i$ by $L(U_i)$. Consider a military setting consisting of four security levels: *top- secret* ($TS$), *secret* ($S$), *confidential* ($C$) and *unclassified* ($U$) (where $U < C < S < TS$). The user Jane Doe has the security clearance of *top-secret*. That is, $L(JaneDoe) = TS$. Each user has one or more principals associated with him. The number of principals associated with the user depends on his security clearance; it equals the number of levels dominated by the user's security clearance. In our example Jane Doe has four principals: *JaneDoe.TS*, *JaneDoe.S*, *JaneDoe.C* and *JaneDoe.U*. At each session, the user logs in as one of the principals. All processes that the user initiates in that session inherit security level of the corresponding principal.

Each transaction $T_i$ is associated with exactly one security level. The level of the transaction remains fixed for the entire duration of the transaction. The security level of the transaction is the level of the principal who has submitted the transaction. For example, if Jane Doe logs in as *JaneDoe.S*, all transactions initiated by Jane Doe will have the level *secret* ($S$).

We require a transaction $T_i$ to obey the simple security property and the restricted $\star$-property [3] that are given below.

1. A transaction $T_i$ with $L(T_i) = C$ may read a database object $x$ only if $L(x) \leq C$.
2. A transaction $T_i$ with $L(T_i) = C$ may write a database object $x$ only if $L(x) = C$.

Property 1 is the simple security property. This property places a restriction on the objects that a transaction can read. A transaction can read an object only if the level of the transaction dominates the level of the object. For example, a *secret* level transaction can read *secret* and *unclassified* documents but not *top-secret* documents. Property 2 describes the restricted $\star$-property. This property places a restriction on the objects that a transaction can write. By virtue of this property, a transaction can write to database objects that are at its own level. To prohibit a transaction from passing information from the dominating level to the dominated level, a transaction at a dominating level is not allowed to write to the objects at the dominated level. For example, if a *top-secret* level transaction is allowed to write an *unclassified* document, then the transaction may pass along *top-secret* level information to the *unclassified* document. For integrity reasons, a transaction is not allowed to write to objects at the dominating level. For example, an *unclassified* transaction can corrupt a *top-secret* level document by writing incorrect information.

We give the formal definition of an MLS transaction below.

**Definition 1. [MLS Transaction]** *An MLS transaction $T_i$ is a set of read and write operations on database objects which are preceded by the command* begin *and followed by*

*the command* abort *or* commit. *The transaction $T_i$ is associated with security level $L(T_i)$ where $L(T_i) \in \mathbf{L}$; it accesses database objects in accordance with the simple security and the restricted $\star$-property.*

We use the term application program in a more general manner.

**Definition 2. [MLS Application Program]** *An MLS application program $A_i$ is a set of operations submitted by the user – the operations may or may not access the database objects. An application program $A_i$ that accesses the database objects is associated with security level $L(A_i)$, where $L(A_i) \in \mathbf{L}$; it accesses database objects according to the rules specified by the simple security and the restricted $\star$-property.*

## 3 Rules in an MLS Active Database

In addition to transactions and application programs, an active database system also has *rules*. A rule is specified by three components: *event*, *condition* and *action*. An event causes a rule to be triggered. Active database systems have mechanisms that monitors the database to check whether an event has occurred. If an event associated with a rule occurs, the rule's condition is evaluated. If the rule's condition evaluates to true, then the rule's action is scheduled for execution. The details of rule execution are considerably more complex than this simple description. We elaborate on the details of rule execution in Section 4.

A rule is a database object on which we allow the following operations.

1. Create – this operation allows a new rule to be created.
2. Delete – this operation allows an existing rule to be deleted.
3. Update – this operation allows an existing rule to be modified.
4. Enable – this operation allows an existing rule to be enabled. Only enabled rules can be triggered.
5. Disable – this operation allows an existing rule to be disabled. A disabled rule cannot be triggered.
6. View – this operation allows an existing rule to be viewed.

Like other database objects in an MLS database, rules are also associated with security levels. Each rule $R_j$ is created by some principal, say $P$, and it inherits the security level of the principal that created it, that is, $L(R_j) = L(P)$. Creation, deletion, modification, disabling, enabling of the rule corresponds to writing of the rule object. Hence, by the restricted $\star$-property these operations can be performed only by transactions or applications whose security level is the same as that of the rule. Viewing the rule corresponds to a read operation of the rule object. Thus, the view operation can be performed by transactions or applications whose security levels dominate the level of the rule.

Next, we give the formal definition of an MLS rule.

**Definition 3. [MLS Rule]** *An MLS rule $R_j$ is defined as a triple $< e_j, c_j, a_j >$ where $e_j$ is the event that causes the rule to be triggered, $c_j$ is the condition that is checked when the rule is triggered and $a_j$ is the action that is executed when the rule is triggered. The MLS rule $R_j$ is associated with exactly one security level which we denote by $L(R_j)$ where $L(R_j) \in \mathbf{L}$. The operations allowed on rules obey the mandatory restrictions specified by the simple security and the restricted $\star$-property.*

Before describing how the security level of a rule is related to the levels of its components (that is, events, conditions and actions), we must describe the components of a rule in more details.

### 3.1 Events in an MLS Active Database Systems

Event specifies what causes the rule to be triggered. Possible events that can be supported in an active database system are

1. Data modification/retrieval events – the event is raised by an operation (insert, update, delete, access) on some database object.
2. Transaction event – the event is raised by some transaction command (e.g. begin, abort, commit etc.).
3. Application-defined event – the application program may signal the occurrence of an event.
4. Temporal events – events are raised at some point in time. Temporal events may be absolute (e.g., 25th December, 2002) or relative (e.g. 15 minutes after x occurs).
5. External events – the event is occurring outside the database (e.g. the sensor recording temperature goes above 100 degrees Celsius).

Events can further be classified into primitive and composite events.

- Primitive event – the event cannot be divided into subparts.
- Composite event – the event is raised by some combination of primitive events.

For example, inserting a tuple in $Employee$ relation is a primitive event. Two hours after a tuple has been inserted in $Employee$ relation is a composite event.

A composite event is constructed using two or more primitive events connected by an event operator. Any composite event $e$ can be denoted as follows.

$$e = e_1\ op_1\ e_2\ op_2 \ldots\ e_n$$

where $e_1, e_2, \ldots, e_n$ are the primitive events making up the composite event $E$, and $op_1, op_2, \ldots op_{n-1}$ are the event operators. Event operators can be logical event operators ($\vee$, $\wedge$ , etc.), sequence operators (;), or temporal composition operators ($after$, $between$, etc.).

**Assigning Security Levels to Events** First, we discuss how to assign security levels to primitive events.

**Security Level associated with Data Modification/Retrieval Event:** The event $e$ has the same security level as the operation $O$ that caused it, that is, $L(e) = L(O)$. If this operation $O$ is performed by some transaction $T$, then the level of $O$ is the same as the level of $T$.

**Security level associated with the Transaction Event:** The event $E$ has the same security level of the transaction $T$ that caused it, that is, $L(e) = L(T)$.

**Security Level associated with Application-Defined Event:** The event $e$ has the same security level as the level at which the application $A$ that generated it is executing, that is, $L(e) = L(A)$.

**Security Level associated with Temporal Event:** An absolute temporal event $e$ is observable by any body and so its security level is public, that is, $L(e) = U$. A relative temporal event is a composite event. The manner in which the level of composite event is calculated is given below.

**Security Level associated with External Event:** The level of the event $e$ is the greatest lower bound of the security clearances of the users $U_1$, $U_2$, ..., $U_n$ who can observe this external event $e$, that is, $L(e) = glb(L(U_1), L(U_2), \ldots, L(U_n))$ (where $L(U_i)$ denotes the security clearance of user $U_i$).

An event like the outside air temperature is 110 degrees Fahrenheit, is observable by all users and so its level is public. Whereas, an event like the sensor reading from a military satellite that can be observed only by *top-secret* personnel, will have a security level of *top-secret*.

Now we discuss how to assign security levels to composite events.

**Security Levels associated with Composite Event:** Consider the composite event $E$ given by, $e = e_1\ op_1\ e_2\ op_2 \ldots\ e_n$, where $e$ is composed of primitive events $e_1$, $e_2$, ..., $e_n$. The security level of the composite event $e$ is the least upper bound of the levels of the primitive events $e_1$, $e_2$, ..., $e_n$ composing it, that is, $L(e) = lub(L(e_1), L(e_2), \ldots, L(e_n))$.

### 3.2 Conditions in an MLS Active Database System

In an active database, when a rule has been triggered *condition* specifies the additional conditions that must be checked before the action can be executed. If the condition part of the rule evaluates to true, then the action is executed. Possible conditions in a rule are

1. Database predicates – the condition might be a predicate on the database state (average salary of employees greater than 50000).
2. Database queries – the condition might be a query on the database state. If the query returns some results, the condition is said to be satisfied. If the query fails to return any result, the condition is not satisfied.
3. Application procedures – the condition may be a specified as a call to an application procedure (example, *max_exceeded()*) which may or may not access the database.

**Assigning Security Levels to Condition** Checking a condition involves reading database objects associated with the condition. We define the level of a condition $c$, denoted by $L(c)$, as follows: It is the least upper bound of all the data that is accessed by the condition. That is, $L(c) = lub(L(D_1), L(D_2), L(D_3), \ldots, L(D_n))$ where $D_1, D_2, \ldots, D_n$ are the data objects accessed by condition $c$.

### 3.3 Actions in an MLS Active Database System

When the rule is triggered and its condition evaluates to true, the action of the rule must be executed. Possible actions in an MLS active database include

1. Data modification/retrieval operation – the action of the rule causes a data operation (insert, update, delete, access).
2. Transaction operation – the action of the rule causes a transaction operation (e.g. abort).
3. Application-defined operation – the action causes some procedure in an application to be executed.
4. External operation – the action causes some external operations (e.g. informing the user).

Some active database languages allow a rule to specify multiple actions. Usually these actions are ordered which allows them to be executed sequentially.

**Assigning Security Levels to Actions** This is how we assign security levels for the actions.

**Security Level associated with Data Modification/Retrieval Action:** The action has the same level as the operation it causes, that is, $L(a) = L(O)$.

**Security Level associated with Transaction Operation:** The action $a$ has the same level as the transaction $T$, that is, $L(a) = L(T)$.

**Security Level associated with Application-defined Operation:** The action $a$ has the same level as the application process $A$, that is $L(a) = L(A)$.

**Security Level associated with External Operation:** In this case, the action is viewed by external observers. The level of the action must be lower than or equal to the security clearances of all the users viewing the action. The level of action $a$ is the greatest lower bound of the security clearances of the users $U_1$, $U_2$, ..., $U_n$ who can observe this operation, that is, $L(a) = glb(L(U_1), L(U_2), \ldots, L(U_n))$ where $L(U_1)$, $L(U_2)$, ..., $L(U_n)$ are the security clearances associated with users $U_1$, $U_2$, ..., $U_n$ respectively.

**Security Level of Action composed of Multiple Constituents:** Consider a rule $R = < e, c, a >$ where the action $a$ is composed of multiple actions, $a_1, a_2, \ldots, a_k$. To keep our model simple, we require that the level of all the actions must be the same. That is, $L(a_1) = L(a_2) = \ldots = L(a_k)$.

### 3.4 Relationship of Security Levels associated with a Rule

The following illustrates the relationship of the level of the rule $R_j$ with the levels of the constituent event $e_j$, the condition $c_j$ and the action $a_j$.

1. $L(e_j) \leq L(R_j)$
2. $L(c_j) \leq L(R_j)$
3. $L(a_j) = L(R_j)$

Item 1 states that a rule may be triggered by an event whose level is dominated by the level of the rule. Item 2 states that a rule may require checking conditions at the dominated level before it can be fired. Item 3 states that a rule can take an action only at its own level.

In a secure environment it might be necessary for dominating levels to monitor suspicious events taking place at some dominated level and take some precautionary action; hence the need for $L(e_j) \leq L(R_j)$. Moreover, $L(e_j) \not> L(R_j)$ ensures that a dominating event does not trigger a dominated rule and create a covert channel. The same reasoning applies for condition $c_j$; thus, the rule $R_j$ might check conditions involving dominated level data (that is, $L(c_j) \leq L(R_j)$), but not data at the dominated levels ($L(c_j) \not> L(R_j)$). The level of the action is the same as the level of the rule, that is, $L(a_j) = L(R_j)$. Since $L(a_j) \not< L(R_j)$, a rule at the dominating level cannot result in an action at the dominated level and create a covert channel[1]. Also, since $L(a_j) \not> L(R_j)$, a rule at the dominated level while executing its action cannot corrupt data at the dominating level.

## 4 Execution Model

The execution model of an active database specifies how the active database behaves at run-time. The execution model will depend on the underlying DBMS. At this point, we do not wish to commit to any particular DBMS. Hence, we do not propose a detailed execution model for an MLS active database system. Instead, we identify the issues that need to be addressed before developing an execution model.

Irrespective of the execution model used, the following activities (as outlined by Paton et al. [29]) are involved during rule execution. When or how these activities are carried out constitute the details of the execution model.

**Event Detection Phase** – refers to the detection of an event occurrence caused by an event source.

**Triggering Phase** – triggers the rules corresponding to the events produced. The association of a rule with its event occurrence is termed rule instantiation.

**Evaluation Phase** – evaluates the condition of the triggered rules. The conflict set is formed which is made up of all the rule instantiations whose conditions evaluate to true.

**Scheduling Phase** – chooses which rule will be processed from the conflict set.

**Execution Phase** – carries out the actions of the chosen rule instantiation.

Next, we investigate each component of the execution model and identify what effect, if any, our MLS rules have on these components.

### 4.1 Rule Processing Granularity

The granularity of rule processing indicates at which instances rules can be processed. Widom and Ceri [38] have identified four kinds of granularity:

1. Always – rules may be processed at any point in time,

[1] Path of illegal information flow based on monitoring the usage of system resources.

2. Smallest database operations – for example, insertion, deletion, update or fetch of a single tuple.
3. Data manipulation statements – for example, at the end of every SQL statement where a statement inserts, deletes, updates, or fetches numerous tuples.
4. Transaction boundaries.

**Kinds of Granularity in an MLS Active Database** In our model, an event at the dominated level can trigger a rule at the dominating level. Since the event may have been generated by a transaction or an application program, we do not want the dominated transaction to be suspended for the execution of a dominating rule and introduce the possibility of a timing channel. A timing channel arises between a dominating level and a dominated level when the dominating level can vary the amount of time required to complete a task to signal information to the dominated level. Thus, the first three kinds of rule processing granularity enumerated above cannot be supported in an MLS Active Database. The only viable rule processing granularity is the fourth item, that is, transaction processing boundaries.

### 4.2 Conflict Resolution and Rule Priorities

In sequential processing, at any given time, only one rule is chosen for execution. However, in an active database system many rules may be triggered at the same time. This may happen because of several reasons: (i) several rules specify the same triggering event, (ii) the rule processing granularity is coarse – many events are triggered before the rules are processed, (iii) rules that are triggered but not chosen for execution remain triggered.

In such a scenario, conflict resolution specifies how the rule to be executed can be chosen. Some conflict resolution strategies are

1. a rule may be chosen arbitrarily,
2. a rule may be chosen based on static properties – time of rule creation or the data on which rules are defined,
3. a rule may be chosen based on dynamic properties – most recently fired rule,
4. a rule may be chosen based on priorities that are specified during rule definition. Priorities are specified by ordering the set of rules, by declaring relative priorities between each pair of rules, or by assigning numeric priorities.

**Conflict Resolution and Rule Priorities in an MLS Active Data base** We can specify any of the conflict resolution policies enumerated above for rules having the same security level. However, if there are rules belonging to different security levels, the conflict resolution policy must always favor the dominated rule. This is because delaying a rule at the dominated level because of the execution of a rule at the dominating level may give rise to a timing channel.

In a multilevel secure active database system we can also specify priorities, but the requirement is that no dominating rule must have a higher priority than a dominated rule. Thus, if priorities are specified by ordering the set of rules, then all rules at dominated levels must be ordered before any rule at the dominating level.

If numeric priorities are to be specified, one approach is to make the priority specification have two parts: one for the security level and the other for the number. For rules having different security levels, the dominated rules will get preference over the dominating rules. For rules having the same security level, the number will decide which rule is chosen for execution.

Note that if dominated rules always get more preference than dominating rules, then the dominating rules might suffer from starvation. One solution is to allocate fixed time slots for each level. Suppose there are two levels: *Low* and *High*, where $Low < High$. We allocate the time slot $< T_1, T_2 >$ for rules at level *Low*, the slot $< T_2, T_3 >$ for rules at level *High*, $< T_3, T_4 >$ for rules at level *Low* etc. The problem with this approach is that if there are no *High* rules for execution at $< T_2, T_3 >$, then processor time gets wasted. To minimize wasting computational resource, a better approach would be to study the processor requirements of the rules at different levels and then divide the time slots accordingly.

### 4.3 Sequential Versus Concurrent Execution

In sequential rule processing only one rule is executed at a time. If multiple rules are triggered, the conflict resolution strategy decides which rule should be executed. An alternate approach to sequential rule execution is concurrent execution. Concurrent rule execution provides a better performance than sequential execution. There can be two kinds of concurrent rule execution:

1. inter-rule concurrency – a rule is executed as one atomic transaction.
2. intra-rule concurrency – rules are divided into parts and each of these part is executed as an atomic transactions. In other words, these parts are executed concurrently.

We can, of course, have systems that allow both the above options.

In inter-rule concurrency, each rule $R_j$ generates a transaction $T_j$ at the same level. In intra-rule concurrency, each rule $R_j$ generates two transactions at its own level: $T_{cj}$ (for evaluating the rule's condition) and $T_{aj}$ (for executing the rule's action). Note that, we need to ensure that $T_{aj}$ commits only after $T_{cj}$ commits and returns a true value.

**Concurrent Execution in an MLS Active Database** Both inter-rule and intra-rule concurrency can be supported in an MLS active database system. Each rule in such a scenario generates one or more transactions at its own level. Since rules at different levels generate transactions at different levels, these transactions must be executed concurrently. The concurrency control algorithms must ensure that no illegal information flow occurs due to the execution of rules. Multilevel secure concurrency control algorithms [2] can be used in such a scenario.

### 4.4 Coupling Modes

Coupling modes are specified which dictate when a triggered transaction is processed relative to the triggering transaction.

We need to specify when the condition will be evaluated relative to the triggering event. We also need to specify when the action will be executed relative to condition evaluation. Each of these can be specified using coupling modes.

**Coupling Modes between Event and Condition** There are three basic kinds of coupling modes between event and condition.

1. Immediate Coupling Mode between Event and Condition – The rule's condition is evaluated as soon as the event has occurred as a part of the triggering transaction.
2. Deferred Coupling Mode between Event and Condition – The rule's condition is evaluated after the triggering transaction has completed all its operation, but before it has been committed.
3. Detached Coupling Mode between Event and Condition – The rule's condition is evaluated in a different transaction than the triggering one.

**Coupling Mode between Event and Condition in an MLS Active Database**

Let the triggering transaction be denoted as $T_i$ and the triggered rule be denoted as $R_j$. If $L(T_i) < L(R_j)$ and the coupling mode between event and condition is immediate or deferred then this may give rise to a timing channel – by manipulating the time taken to execute the triggered rule, information may be transmitted from the level $L(R_j)$ to level $L(T_i)$. In other words, the immediate and deferred mode can be supported for cases where $L(R_j) = L(T_i)$. When $L(R_j) \neq L(T_i)$, the immediate and deferred modes cause a breach of security.

The detached coupling mode, however, can be supported safely in all cases. The detached coupling mode also provides more concurrency than the immediate or deferred mode. This is because in immediate or deferred mode the triggered rule and the triggering transaction execute as one large transaction and hence this large transaction takes more time to complete; this in turn delays the transactions that are waiting to lock items that are locked by this large transaction. In detached coupling mode the triggered rule is executed as a separate transaction and the performance problem associated with one large transaction does not arise.

**Coupling Modes between Condition and Action** Coupling modes can also be used to specify when the rule's action takes place relative to condition evaluation. Here also three modes are specified.

1. Immediate Coupling Mode between Condition and Action – The rule's action is executed immediately after condition evaluation.
2. Deferred Coupling Mode between Condition and Action – The rule's action is executed as a part of the same transaction as the condition evaluation, but not necessarily immediately.
3. Detached Coupling Mode between Condition and Action – The rule's action and the rule's condition evaluation are executed as two different transactions.

**Coupling Mode between Condition and Action in an MLS Active Database**

Since the level of the transaction evaluating the rule's condition is the same as the level of the transaction executing the rule's action, all the above modes can be supported in a multilevel secure active database system. However, in detached coupling mode we can take advantage of the intra-rule concurrency because the rule's action execution and condition evaluation are two separate transactions.

### 4.5 Iterative Versus Recursive Algorithms

Often times the condition evaluation or action execution of a rule signals events which in turn trigger other rules. In such a scenario, how should the rule processing proceed? There are two choices:

1. recursive rule processing – the original condition evaluation or action execution is suspended and any immediate rules triggered by the (condition evaluation or action execution) event are chosen for execution,
2. iterative rule processing – the original rule proceeds to completion after which other rules are chosen for execution.

**Iterative vs. Recursive Algorithms in an MLS Active Database** Recursive rule processing can be supported only if the triggering rule and the triggered rule have the same security level. Recursive rule processing may introduce timing channels if the triggered rule and the triggering rule belong to different security levels. This happens because the dominated triggering rule is suspended to allow for the completion of the dominating triggered rule – the dominating rule can manipulate the time taken to complete and convey information to the dominated rule. Iterative rule processing, on the other hand, does not suffer from this security breach and can safely be supported in an MLS Active Database System.

### 4.6 Error Handling

An error may occur while a rule is being processed. Widom and Ceri [38] elaborates on why an error may be generated during rule processing. Their reasons include (i) data required by a rule for condition evaluation or action execution has been deleted, (ii) authorization privileges required for rule condition evaluation or action execution have been revoked, (iii) a rule's condition or action reveals an error condition, (iv) the number of rules processed exceeds the systems limits, (v) concurrent execution of transaction creates a deadlock, or (vi) a system generated interrupt or error occurs.

Once an error occurs during rule processing, the question is how is the error handled. There are various options for handling the error.

1. Abort the transaction responsible for the event that in turn triggered the rule.
2. Ignore the rule that caused the error and continue processing.
3. Backtrack to the state when rule processing started and either restart rule processing or continue with the transaction.
4. Adopt a contingency plan that attempts to recover from the error state.

**Error Handling in an MLS Active Database System** All the above options can be supported for cases where the level of the rule is the same as the level of the transaction. However, if the level of the triggering transaction is dominated by the level of the rule, then option 1 cannot be supported because aborting a dominated level transaction because of an error in a dominating rule constitutes a covert channel. In such a scenario option 4 may be the best one.

## 5 Architecture

The architecture of an active database system will depend on the knowledge model and execution model of the active database system. In general, there are two approaches to building active database systems. One is the layered approach where the active database components are built on top of an existing passive database system. The alternate approach is of a built-in architecture where active database components become a part of the database itself. The layered approach is easier to construct than the built-in one, but it is not as efficient as the built-in one.

### 5.1 Architecture for an MLS Active Database System

In a multilevel secure active database system the built-in approach is preferred. The main reason for this preference is performance. Also, for the layered approach we need an existing efficient multilevel secure database as an underlying database which is not widely available.

As mentioned previously, the actual architecture of an MLS active database system will depend on the knowledge and execution model. However, we do describe an architecture at a very abstract level of an MLS active database system (refer to figure 1). This will give an idea of the extra components that must be supported for processing MLS rules. The principal components are depicted by rectangles and the data stores are depicted by ellipses.

The main components are:

1. Event Detector at Level $l_i$ – This is responsible for detecting events at level $l_i$. Note that, event detector at level $l_i$ can detect events at all levels that are dominated by $l_i$. The database notifies the event detector al level $l_i$ about all primitive events occurring at levels dominated by $l_i$. Composite events are constructed using the knowledge about incoming primitive events and past events obtained from the history.
2. Condition monitor at Level $l_i$[2] – This is responsible for evaluating the conditions of rules that are triggered by the events detected by the event detector $l_i$. The condition monitor sends a request to the query evaluator to evaluate the condition. On the

[2] We can have a single event detector and condition monitor instead of having event detectors and condition monitors for each of the security levels. However, this would require a trusted event detector and a trusted condition monitor. Developing trusted components requires considerable effort. Hence, we choose to have event detectors and condition monitors for each security level.

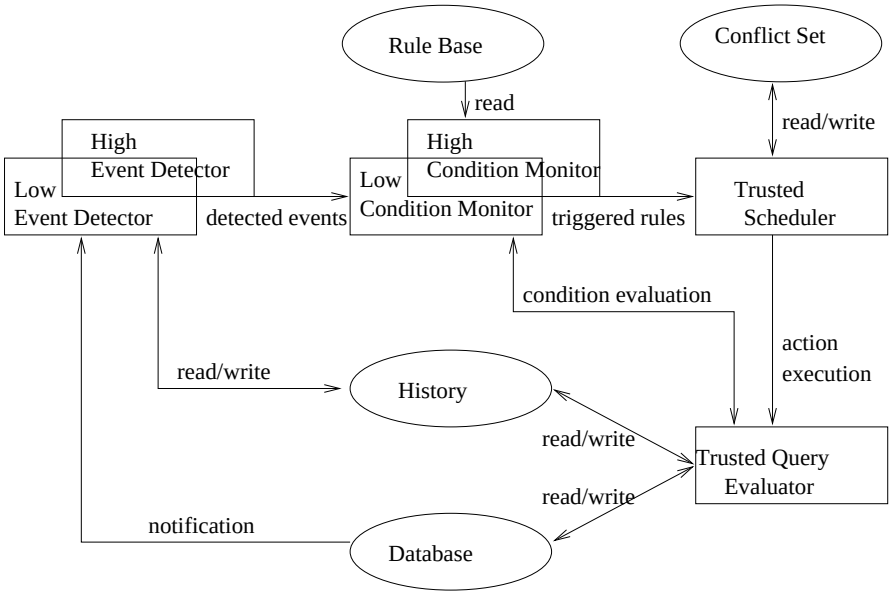


**Fig. 1.** A High-Level Architecture for Processing MLS Rules

basis of the response from the query evaluator, the condition monitor decides which rules are triggered. These rules are then sent to the scheduler.

3. Trusted Scheduler – This is responsible for scheduling which rule is to be executed. Since the scheduler accesses rules at different levels, it must be a trusted component.
4. Trusted Query Evaluator – This is responsible for executing database actions and queries. We have shown the query evaluator to be a trusted component because it is responsible for executing database operations at all levels. In real world, the query evaluator will have several components, some of which are trusted and others which are not.

The above is a very abstract view of rule processing. Each of these components will be composed of sub-components and how these sub-components are connected will constitute the low-level architecture of the MLS active database system.

## 6 Related Work

### 6.1 Related Work in Multilevel Secure Active Databases

Very little work appears in the area of multilevel secure active databases. The major work in this area is by Smith and Winslett [33]. The authors show how an MLS relational model can be extended to incorporate active capabilities. The underlying MLS relational model supports polyinstantiation: that is, all MLS entities in this model can

exist at multiple security levels simultaneously. An MLS rule being an MLS entity can also exist at multiple security levels. The main difference with our work is that this work is based on the polyinstantiation model.

### 6.2 Related Work in Expert Systems

Morgenstern [26] considers the problem of covert channels in deductive databases that are subject to mandatory security requirements. Berson and Lunt [4] describe the problems that must be solved in incorporating mandatory security requirements in a product rule system. Garvey and Lunt [14] extend an MLS object-oriented database system with productions rules. Expert system rules differ from active database rules. Expert system rules are executed upon an explicit request for information; active database rules are executed as side effects. Expert system rules are used for inferencing – the order of rule execution is not important. This is not so with active databases. Both forward chaining and backward chaining rules are supported in an expert system. Active databases, on the other hand, support only forward chaining rules.

### 6.3 Related Work in Active Databases

Many work has been performed in the area of active databases. We will only describe a few of these works. Most of these work differ in the knowledge model and execution model. Some of these active databases use a relational model as an underlying database and others use an object-oriented one.

Most commercial systems support some form of triggering mechanisms. Some of the prototypes based on the relational model are Starburst [37], Ariel [17], POSTGRES [35]. Starburst rule system is quite conservative – supports a limited set of facilities. The most important feature of Starburst is its set-based execution model. Whenever an event that is of importance to some rule takes place, the event is logged in a transition table. At rule assertion points, this transition table is checked and the net-effect of all the logged events is taken into consideration before firing a rule. The POSTGRES project included a number of extensions to the relational model – providing active capabilities is one such extension. POSTGRES supports immediate rule processing and does not support the deferred modes. Ariel supports both ECA and condition-action rules. This has important consequences for the implementation of the rule processing system.

Notable among the object-oriented models are HiPAC [24, 30], NAOS [10], Chimera [8], Ode [1] , SAMOS [15], Sentinel [9] and REACH [7]. HiPAC is one of the earlier projects in active database systems and they have contributed to many of the pioneering ideas: coupling modes, composite events, parallel execution of triggered rules in the form of subtransactions, to name a few. Also, this project identified real-time applications that can benefit from active database systems. NAOS is an active rule system for the $O_2$ commercial OODB database. However, NAOS has been implemented as a part of kernel of $O_2$. The NAOS execution model supports depth-first, recursive processing of immediate rules and breadth-first iterative processing of deferred rules. SAMOS provides active capabilities to the ObjectStore commercial OODB. The most significant feature of SAMOS is the event detector, the semantics of which is based on petri nets. The event language is also expressive and allows the specification of composite

events. Sentinel and REACH extends the C++ based OpenOODB system from Texas Instruments with active capabilities. Chimera builds upon a deductive object-oriented database system.

### 6.4 Related Work in Multilevel Secure Databases

A large number of works also appears in multilevel secure database system. Majority of these works [11–13, 16, 18–21, 31, 32, 34] are in the area of relational database systems and some [5, 6, 19, 22, 25, 27, 36] are in the area of object oriented database systems.

In a multilevel relational database system the data are classified into different security levels, and the users are associated with different security clearances. An important issue is the granularity at which the data can be classified. There are four possibilities: the data may be classified (i) at the element level, (ii) at the attribute level, (iii) at the tuple level, or (iv) at the relation level. A relation in which the data has been classified as above is a multilevel relation.

Most of the work differ in the way they solve integrity and polyinstantiation problems. Entity integrity ensures that no user sees a null value for the primary key of a relation. Unless, no action is taken, entity integrity would be violated when a user with a low clearance sees null for the cases where the primary key of a tuple is classified at a level higher than the user. One solution is that all the primary key elements be classified at the same level. The other solution is that the primary keys should be as low as any other elements in the tuple. Referential integrity requires that a tuple of low security level cannot reference a tuple of a high security level because the referenced tuple would appear to be non existent to users with low clearance.

There are three kinds of polyinstantiation [12]. A polyinstantiated relation [36] occurs when two subjects at different security levels try to create a relation of the same name. A polyinstantiated tuple occurs when a user inserts a tuple that has the same primary key value as an existing tuple at the higher level. A polyinstantiated element [23] is created if a user updates an element in a tuple which appears to be null (because the element has a higher security classification than the user).

A lot of work appears in the area of multilevel secure object oriented databases. Some of these work [5, 25, 36] consider single-level objects: each object is assigned a security level and this security level applies to all the properties and the methods of this object. The main advantage of this approach is its simplicity. The security kernel remains small so that it can be efficiently verified. The disadvantage is that in real-world there is a need for multilevel objects (where the different attributes of an object may be associated with different security levels) which cannot be adequately modeled.

The alternate approach is of supporting multilevel objects. The main problem of supporting multilevel object is that the security kernel becomes complex. Verifying the trusted security kernel becomes a difficult task. To keep the security kernel simple, researchers [5, 6] have proposed the idea of decomposing multilevel objects into single-level ones and storing them as single-level objects. In such cases, mechanisms are developed that ensure that the users get the multi-level view of the object, even though they are stored as single-level ones.

Many security models pertaining to object-oriented database systems have been proposed. The notable ones are the Jajodia-Kogan filter model [19], SORION model

[36], Millen-Lunt model [25] and the SODA model [22]. Olivier et al. [28] identify security issues and properties that are relevant to the modeling and implementation of a secure object-oriented database system.

## 7 Conclusion and Future Work

The absence of a multilevel secure active database limits the applicability of active technologies to applications that use an underlying MLS database system. Our work is a first step towards fulfilling this gap. We have identified what kinds of rules can be supported and how these rules can be classified into security levels. Next, we have identified what impact these rules have on the rule execution semantics. In many cases it turns out that some features of active databases cannot be supported without introducing illegal information flow. We have not based our work on any relational or object-oriented models; our observations, therefore, will be useful to developers of any MLS active database system.

A lot of work remains to be done. The next step will be to finalize on the rule execution semantics. Consider, for example, iterative versus recursive rule processing. For an MLS active database there are two possibilities: (i) support only iterative rule processing or (ii) support iterative as well as recursive rule processing and limit recursive rule processing to rules having the same security level. Option (ii) offers more flexibility but is more complicated and potentially more time consuming. Note that, we need to study the kinds of applications that will be executed on MLS active database systems before finalizing on the rule execution semantics.

Once the rule execution semantics have been finalized we will develop a detailed architecture for an MLS active database system. The detailed architecture will identify the rule processing components and their interactions with the underlying MLS database. This will also provide guidelines as to which of these components must be trusted. The final step is to implement each of these components such that rule processing can be performed in a secure and efficient manner.

## References


1. R. Agarwal and N. Gehani. Ode (Object database and environment): The language and the data model. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 36–45, Portland, OR, May 1989.
2. V. Atluri, S. Jajodia, T.F̃. Keefe, C. McCollum, and R. Mukkamala. Multilevel Secure Transaction Processing: Status and Prospects. In P. Samarati and R.S̃. Sandhu, editors, *Database Security X: Status and Prospects*, chapter 6, pages 79–98. Chapman & Hall, 1997.
3. D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corporation, Bedford, MA, July 1975.
4. T. A. Berson and T. F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 235–242, Oakland, CA, April 1987.
5. E. Bertino and S. Jajodia. Modeling Multilevel Entities using Single Level Objects. In *Proceedings of the Third International Conference on Deductive and Object-Oriented Databases*, volume 760 of *Lecture Notes in Computer Science*, pages 416–428, Phoenix, AZ, December 1993. Springer-Verlag.

6. N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Virtual View Model to Design a Secure Object-Oriented Database. In *Proceedings of the National Computer Security Conference*, pages 66–76, Baltimore, MD, October 1994.
7. A.P. Buchman, H. Branding, T. Kundrass, and J. Zimmermann. REACH: A REal-time ACtive and Heterogeneous Mediator System. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4), December 1992.
8. S. Ceri and R. Manthey. Consolidated specification of Chimera, the conceptual interface of idea. Technical Report IDEA.DD.2P.004, Politecnico di Milano, Milan, Italy, June 1993.
9. S. Chakravarthy, E. Hanson, and S.Y.W. Su. Active data/knowledge base research at the University of Florida. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4):35–39, December 1992.
10. C. Collet, T. Coupaye, and T. Svensen. NAOS– efficient and modular reactive capabilities in an object-oriented database system. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 132–143, Santiago, Chile, 1994.
11. O. Costich and J. McDermott. A multilevel transaction problem for multilevel secure database system and its solution for the replicated architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 192–203, Oakland, CA, May 1992.
12. D. Denning and T. F. Lunt. A multilevel relational data model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 220–234, Oakland, CA, May 1987.
13. P. A. Dwyer, G. D. Gelatis, and M. B. Thuraisingham. Multilevel security in database management systems. *Computers and Security*, 6(3):252–260, June 1987.
14. T. D. Garvey and T. F. Lunt. Multilevel Security for Knowledge-Based Systems. In *Proceedings of the Sixth Computer Security Applications Conference*, pages 148–159, Tucson, AZ, December 1990.
15. S. Gatziu, A. Geppert, and K. R. Dittrich. Integrating active concepts into an object-oriented database system. In *Proceedings of the Third International Workshop on Database Programming Languages*, Nafplion, Greece, August 1991.
16. J. T. Haigh, R. C. O'Brien, and D. J. Thomsen. The LDV Secure Relational DBMS Model. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 265–279. Elsevier Science Publishers B.V. (North-Holland), 1991.
17. E. Hanson. Rule condition testing and action execution in Ariel. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–58, San Diego, CA, June 1992.
18. D. K. Hsiao, M. J. Kohler, and S. W. Stround. Query Modifications as Means of Controlling Access to Multilevel Secure Databases. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 221–240. Elsevier Science Publishers B.V. (North-Holland), 1991.
19. S. Jajodia and B. Kogan. Transaction Processing in Multilevel Secure Databases using Replicated Architecture. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 360–368, Oakland, CA, May 1990.
20. S. Jajodia and R. Sandhu. Polyinstantiation Integrity in Multilevel elations Revisited. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 297–307. Elsevier Science Publishers B.V. (North-Holland), 1991.
21. S. Jajodia and R. Sandhu. Toward a Multilevel Relational Data Model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 50–59, Denver, CO, 1991.
22. T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham. A Multilevel Security Model for Object-Oriented Systems. In *Proceedings of the National Computer Security Conference*, pages 1–9, Baltimore, MD, October 1988.
23. T. F. Lunt and E. B. Fernandez. Database Security. *SIGMOD Record*, 19(4):90–97, December 1990.

24. D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 215–224, Portland, OR, May 1989.
25. J. K. Millen and T.F. Lunt. Security for Object-Oriented Database Systems . In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 260–272, Oakland, CA, May 1992.
26. M. Morgenstern. Security and Inference in Multilevel Database and Knowledge-Base Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 357–373, San Francisco, CA, May 1987.
27. M. Morgenstern. A Security Moddel for Multilevel Object with Bidirectional Relationship. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 53–71. Elsevier Science Publishers B.V. (North-Holland), 1991.
28. M.S. Olivier and S. H. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Transactions on Database Systems*, 19(1):3–46, March 1993.
29. N.W. Paton and O. Diaz. Active Database Systems. *ACM Computing Surveys*, 31(1):63–103, 1999.
30. A. Rosenthal, S. Chakravarthy, B. Blaustein, and J. Blakeley. Situation monitoring for active databases. In *Proceedings of the Fifteenth International Conference On Very Large Databases*, pages 455–464, Amsterdam, The Netherlands, August 1989.
31. R. Sandhu and S. Jajodia. Referential Integrity in Multilevel Secure Databases. In *Proceedings of the National Computer Security Conference*, pages 39–52, Baltimore, MD, September 1993.
32. L. M. Schlipper, J. Filsinger, and V. M. Doshi. A Multilevel Secure Database Management System Benchmark. In *Proceedings of the National Computer Security Conference*, pages 399–408, Baltimore, MD, October 1992.
33. K. Smith and M. Winslett. Multilevel secure rules: Integrating the multilevel and the active data model. Technical Report UIUCDCS-R-92-1732, University of Illinois, Urbana-Champaign, IL, March 1992.
34. P. D. Stachour and M. B. Thuraisingham. Design of LDV: A Multilevel Secure Relational Database Management System. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):190–209, June 1990.
35. M. Stonebraker and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92, October 1991.
36. M. B. Thuraisingham. Mandatory Security in Object-Oriented Database Systems. In *Proceedings of the International Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 203–210, New Orleans, LA, October 1989.
37. J. Widom. The Starburst Rule System: Language Design, Implementation and Application. *Bulletin of the IEEE Technical Committee on Data Engineering*, 15(4):15–18, December 1992.
38. J. Widom and S. Ceri. *Active Database Systems Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, San Francisco, CA, 1996.