Expression and Enforcement of Confidentiality Policy in Active Databases

Julien A. Thomas  Nora Cuppens-Boulahia  Frédéric Cuppens
firstname.lastname@telecom-bretagne.eu
Télécom Bretagne ; LUSI Department
Université Européenne de Bretagne
Cesson Sévigné, France

ABSTRACT
Many research works focused on modeling relational database management systems (DBMS) that support multilevel security (MLS) policies. However, most of these previous proposals only consider static aspects of relational databases and do not address dynamicity provided by mechanisms like triggers. Since such mechanisms introduced specific security problems, in particular they create new information flows, it is necessary to extend traditional MLS models designed for relational databases to handle these problems. However, it has been shown in many papers that triggers lack a formal model to support them and so they are not free of ambiguities. To address these theoretical limitations of trigger, our work is based on a formal model that applies MLS policies to active databases. Active databases provide a more expressive and formal framework than triggers. In this paper, we first define an information flow model for active databases. Based on this security model, we then present security requirements that are sufficient to prevent illegal information flows and prove them using the B method.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—Formal methods; H.2.0 [Database Management]: General—Security, integrity, and protection

General Terms
Theory, Security, Verifications

Keywords
Active Databases, ECA Rules, Security Policies, Information Inference, B Method

1. INTRODUCTION
Security modeling and evaluation of Database Management Systems (DBMS) have concrete applications and are active research domains [15] [3] [6] [7]. Previous studies generally focus on modeling confidentiality [11] and integrity [4] in different database models, such as relational, object-oriented and logic databases [6] [7] [8]. These works focus on how to securely evaluate database queries with respect to a given security policy, generally a Multilevel Security (MLS) Policy. However, many database models also provide dynamic mechanisms like for instance the trigger mechanism in relational database. A database trigger is a procedural code automatically executed in response to certain events on a particular table or view in a database. Triggers introduce specific security problems that are not addressed in previous aforementioned database security models.

In this paper, we actually define an MLS security model for Active Databases. Since triggers are not free of ambiguities, Active Databases have been defined [9] [12] [2] and formalized, using for instance the ECA (Event-Condition-Action) rules paradigm [9]. ECA rules provide a more formal and expressive framework than triggers to model dynamic database behaviors. We then show that dynamic concepts embedded in active databases may engender information flows not taken into account in previous studies. Thus, we present an extended MLS security model for Active Databases and define security requirements that this model must support.

This paper is organized as follows. In section 2, we present the ECA rules paradigm as a formal framework for Active Databases, show the existing issues associated with information flow control in Active Databases and present a security model which address these issues. In section 3, we propose an ECA rule based security policy for active databases which enforces our security model. In section 4, we prove that these security policies are sufficient to avoid illegal information flows in active databases and that the confidentiality property specified in our security model is satisfied. In section 5, we compare our work with existing security models for DBMS. Finally, section 6 concludes the paper.

2. CONFIDENTIALITY POLICY AND ACTIVE DATABASES

2.1 Active Database Model
The database model consists of active entities (USERS), database states (DATABASE) and database operations (OP). We adopt a logical view of the database. Thus the database is modeled using a set of predicates. For example, we shall
consider the following predicates to model a bank database:

- \text{client}(\text{IdClient}, \text{BankBranch}): \text{client Information}
- \text{account}(\text{IdClient}, \text{BankBranch}, \text{Val}): \text{Val is the account balance of client identified by IdClient in BankBranch}
- \text{ageClient}(\text{IdClient}, \text{Val}): \text{Val is the age of IdClient}
- \text{accountForYoung}(\text{BankBranch}, \text{Nb}): \text{Nb is the number of young people accounts in BankBranch}

The database state then corresponds to a set of fully instantiated predicates (called facts) \( p_i \) and we say that \( p_i \) holds if \( p_i \in \text{database} \). For example \text{ageClient}(\text{John}, 20) means that John is 20 years old.

The database operations are the \text{select}, \text{delete} and \text{insert} operations. The update operation is modeled as a sequence of a \text{delete} operation followed by an \text{insert} operation. For example, updating John’s balance account in bank \text{Bpo} from 50 to 80 is modeled by the operation \text{delete}(\text{account}(\text{John}, \text{bpo}, 50)) followed by the operation \text{insert}(\text{account}(\text{John}, \text{bpo}, 80)).

In \cite{10}, active databases are defined as databases supporting the ECA-rule (Event-Condition-Action) paradigm. The ECA rule paradigm is introduced in \cite{9} and extended by several works such as \cite{10,2}. In \cite{2}, Baral, Lobo and Trajcevski propose the \text{Lactive} language to model ECA rules. In the following section, we take our inspiration in \text{Lactive} to represent the ECA rule concept and then introduce security issues associated with \text{Lactive}.

### 2.1.1 Actions

When considering active rules, the first dimension to consider is the action definition. In \text{Lactive}, the definition of actions rules \( \text{do}(a) \) is modeled as follows:

\[
\text{do}(a) \text{ causes } op_1, \ldots, op_k
\]

\[
\text{if } q_1(X_1), \ldots, q_m(X_m)
\]

where \( op_1, \ldots, op_k \) corresponds to sequential execution of \text{select}, \text{delete} or \text{insert} operations and \( q_1(X_1), \ldots, q_m(X_m) \) is the conjunctive pre-condition to execute the action.

Based on our bank database predicates, examples of action definition are the following:

- \text{do}(\text{createClient}(\text{IdClient}, \text{BankBranch})) \text{ causes } \text{insert}(\text{client}(\text{IdClient}, \text{BankBranch})) \text{ if } \neg \text{client}(\text{IdClient}, \text{BankBranch})
- \text{do}(\text{createAccount}(\text{IdClient}, \text{BankBranch}, \text{V})) \text{ causes } \text{insert}(\text{account}(\text{IdClient}, \text{BankBranch}, \text{V})) \text{ if } \text{accountForYoung}(\text{BankBranch}, \text{Nb})
- \text{do}(\text{youngStatsUpdate}(\text{BankBranch})) \text{ causes } \text{delete}(\text{accountForYoung}(\text{BankBranch}, \text{Nb})) \text{ if } \text{accountForYoung}(\text{BankBranch}, \text{Nb})

### 2.1.2 Events

As we consider active databases, actions may trigger events. Trigger events are defined as follows:

\[
\text{event}_{-}\text{name}(X) \text{ after } \text{do}(a) \text{ if } r_1(X_1) \ldots r_m(X_m)
\]

Example of events are the following \text{newClient} and \text{setStats}:

\[
\text{newClient}(\text{IdClient}, \text{BankBranch}) \text{ after } \text{do}(\text{createClient}(\text{IdClient}, \text{BankBranch})) \text{ if } \neg \text{client}(\text{IdClient}, \text{BankBranch})
\]

\[
\text{setStats}(\text{IdClient}, \text{BankBranch}) \text{ after } \text{do}(\text{createAccount}(\text{IdClient}, \text{BankBranch}))
\]

### 2.1.3 Active Rules

Finally, the occurrence of events may in turn initiate the execution of a sequence of new actions through an active rule \( \text{rule}_{-}\text{name} \).

\[
\text{rule}: \text{event}_{-}\text{name}(X) \text{ initiates } \text{do}(\alpha_1) \ldots, \text{do}(\alpha_k)
\]

\[
\text{if } t_1(X_1), \ldots, t_p(X_p)
\]

We consider for example the following active rules which initialize the client account depending on the age of the client and update statistics of accounts created for young clients.

\[
\text{newClient}(\text{IdClient}, \text{BankBranch}) \text{ initiates } \text{do}(\text{createAccount}(\text{IdClient}, \text{BankBranch}, 0)) \text{ if } \text{ageClient}(\text{IdClient}, \text{Age}) \land \text{Age} > 20
\]

\[
\text{newClient}(\text{IdClient}, \text{BankBranch}) \text{ initiates } \text{do}(\text{createAccount}(\text{IdClient}, \text{BankBranch}, 50)) \text{ if } \text{ageClient}(\text{IdClient}, \text{Age}) \land \text{Age} \leq 20
\]

\[
\text{setStats}(\text{IdClient}(\text{BankBranch})) \text{ initiates } \text{do}(\text{youngStatsUpdate}(\text{BankBranch})) \text{ if } \text{ageClient}(\text{IdClient}, \text{Age}) \land \text{Age} \leq 20
\]

### 2.1.4 Multilevel Security Policy

The security model is based on a Multilevel Security Policy. Thus, we add the set LEVELS of security levels and the \text{inf} \_level, \text{equal} \_level and \text{inf} \_equal \_level relations. These relations are defined on \( \text{LEVELS} \times \text{LEVELS} \rightarrow \text{BOOL} \). Regarding the last relation, \text{inf} \_equal \_level(\text{lv1}, \text{lv2}) is a transitive and reflexive relation and is abbreviated as \( \text{lv1} \sqsubseteq \text{lv2} \) in the following.

We also define functions to assign security levels to subjects and objects: \text{clearance} \_level \in \text{USERS} \rightarrow \text{LEVELS} and \text{classification} \_level \in \text{DATABASE} \rightarrow \text{LEVELS}.

We require the security policy to satisfy the \text{known} \_policy constraint, which means that we suppose that the multilevel security policy definition is itself unclassified. In order to meet this requirement, the values of the security functions must be classified at the lowest level denoted \( \bot \). This is formalized as follows:

\[
\forall (\text{user}, l), (\text{user} : \text{USERS}, l : \text{LEVELS}) \Rightarrow \text{classification} \_level(\text{user}, l) \sqsubseteq \bot
\]

\[
\forall (p, l), (p : \text{DATABASE}, l : \text{LEVELS}) \Rightarrow \text{classification} \_level(\text{classification} \_level(p, l), l)
\]

These two assumptions prevent the \text{classification} \_level information to flow while consulting the policy, as it is classified at the lowest level \( \bot \) and already known by all users.

We also assume that the predicates \text{classification} \_level and \text{clearance} \_level must not be modified by user queries. This assumption is done as we consider such queries to be part of the policy administration. Policy administration, including declassification, is out of the scope of this paper but this issue will be addressed in a forthcoming paper.

### 2.2 Security Issues for Active Databases

When considering active databases, events are responsible for new and potentially insecure information flows that are not controlled by traditional confidentiality models. In traditional databases, confidentiality is enforced by access control when select, insert or delete operations submitted by users are executed. We will show in this section that active databases need additional security mechanisms.

First, consider the following policy: \text{client}(C, B) and \text{account}(C, B, V) are classified \text{Secret} and \text{ageClient}(C, V)
is classified TopSecret, with TopSecret ⊈ Secret. Due to the defined rules, information about the age of the client \((ageClient(C, V) \land V \leq 20)\) is disclosed to Secret whenever a young client is registered. Besides, a user cleared TopSecret may create a covert channel of type TopSecret ~ Secret. This issue occurs because actions on accountForYoung. This illustrates a security issue associated with decreasing security level throughout the triggering action chain.

From these two examples, we see that active databases create additional and potentially insecure information flows. In current studies, these flows are not taken into account as they rely on databases without active mechanisms. For instance, the work by Sicherman and al. and Biskup consider information flows when querying the databases \([7, 15]\), they do not take into account the presented information flows.

In the rest of the paper, we shall control information flows in the action, event and active rule definitions, and the associated triggerings action ~ event, event ~ action. We shall define our security properties and their enforcement as summarized in figure 1.

### 2.3 Security Model for Active Databases

As explained in section 1, active databases create new, potentially insecure, information flows. In order to pinpoint the differences between standard and active databases, we rely on the trace based security modeling defined in 5. We assume that the Active DBMS executes user requests sequentially, i.e. without parallelism (see figure 2). Thus we consider a set TRACES of traces defined as follows:

- A trace \(t = \{s_0, \alpha\}\) is composed of the database initial state \(s_0\) and an infinite sequence \(\alpha\) of actions executed in trace \(t\).
- If \(t = \{s_0, \alpha\}\) is a trace, then \(t_n = \{s_0, \alpha_n\}\) is a finite trace composed of the database initial state \(s_0\) and the sequence \(\alpha_n\) of the first actions executed in trace \(t\).
- If \(t_n = \{s_0, \alpha_n\}\) is a finite trace, then \(s_n\) represents the database state obtained after executing the sequence of actions \(\alpha_n\) starting from the initial state \(s_0\).

- We assume that every action in a trace is executed at a given security level. If \(\alpha\) is a finite or infinite sequence of actions and \(L\) is a security level, then \(\alpha[L]\) is the subsequence of actions whose security level is lower than or equal to \(L\).
- We consider that two database states \(s\) and \(s'\) are equivalent at level \(L\) (denoted \(s \approx_L s'\)) if (1) the classification of predicates classified lower than or equal to \(L\) is identical in \(s\) and \(s'\) and (2) the truth value of predicates classified lower than or equal to \(L\) is identical in \(s\) and \(s'\).
- Equivalence between finite or infinite traces at some security level \(L\) is defined as follows: \(t \sim_L t' \iff s_0 \approx_L s'_0 \land \alpha(L) = \alpha'(L)\). This definition says that two (finite or infinite) traces are indistinguishable at level \(L\) if the initial states of the database in these two traces are equivalent at level \(L\) and the subsequence of actions whose security level is lower than or equal to \(L\) is identical in \(t\) and \(t'\).

Now, in order to define our security properties, we consider two traces \(t_n = \{s_0, \alpha_n\}\) and \(t'_n = \{s'_0, \alpha'_n\}\).

In standard databases, the confidentiality property at level \(L\) is defined as the equivalence between the state of the system when the considered traces are equivalent:

1. \(\forall t, t', n, n' : TRACES, n, n' : INTEGER \implies t_n \sim_L t'_n \iff s_n \approx_L s'_n\).

This definition says that if two traces are indistinguishable at level \(L\), then the resulting state should be equivalent at level \(L\).

In standard databases, traces only contain user queries. However, in active databases, traces do not only contain the user queries but also the triggered events and actions. We refer to the subset of queries in \(\alpha\) issued by the users by \(ext(\alpha)\) (user external actions). In order to avoid the issues mentioned in the previous section, we require the database traces to only depend on the user queries and the initial database states. The confidentiality property is thus defined as the equivalence of the traces and the database states when the external actions are equivalent:

1. \(s_0 \approx_L s'_0 \land ext(\alpha_n)[L] = ext(\alpha'_n)[L] \implies t_n \sim_L t'_n\),

2. \(t_n \sim_L t'_n \iff s_n \approx_L s'_n\).

The second property is identical to the one required for standard databases. The first property requires that the sequence of actions executed at level \(L\) in a given trace (including both user external actions and event triggered...
actions) should be determined by the sequence of user external actions executed at level $L$ and information classified at level $L$ in the initial state. As shown in [5], condition $s_0 \approx_L s'_0 \wedge ext(a_{n})/L = ext(a'_{n})$ actually represents the authorized knowledge of users at level $L$. Thus the first property requires that the occurrence of event triggered actions should be determined by authorized knowledge. This first property also requires that, when several triggered actions are activated simultaneously, then the order of execution of these triggered actions should be determined by authorized knowledge. For this purpose, we follow [2] and assume that active rule definitions are associated with a total order relation which determines the order of execution of triggered actions. This total order relation is fixed and represents public knowledge. So, we see that security policies for active databases must satisfy additional properties to control information flows produced by the user queries through the execution of event triggered actions.

3. SECURITY PROPERTY ENFORCEMENT IN ACTIVE DATABASES

We have presented security requirements to control information flow in multilevel active databases. In this section, we show how to enforce these security requirements.

3.1 ECA Rules Security Model

As previously mentioned, insecure information flows occur because of incorrect access to information or because of actions launched at incorrect security levels. In order to avoid these flows, we first need to extend the active database definitions with security levels.

The first definition is the action definition. We extend the $do$ operator by adding a security level which describes at which level the action is performed.

Definition 1 (Multilevel Action) An action is defined with the $do$ operator on $\text{ACTIONS} \times \text{LEVELS}$ and is modeled by the following template:

$$\text{do}(\alpha, L) \text{ causes } op_1 \ldots op_k \text{ if } q_1(X_1) \ldots q_n(X_n)$$

We assume that $\forall (\alpha, L). (\text{classification}_L.(\text{do}(\alpha, L), L).$.

The second definition is the event definition. As for the action definition, we extend the previous event definition by adding a level parameter to the definition.

Definition 2 (Multilevel Event) An event is defined by a name and a header on $\text{PARAMETERS} \times \text{LEVELS}$. It is modeled by the following template: $\text{event}_\text{name}(X, L_1)$ after $\text{do}(\alpha, L_2)$ if $r_1(Z_1) \ldots r_m(Z_m)$.

Finally, we revise the active rule definition by adding security levels.

Definition 3 (Multilevel Active Rule) An active rule is modeled by the following template: $\text{rule}_\text{name} : \text{event}_\text{name}(X, L_0)$ initiates $\text{do}(\alpha_1, L_1) \ldots \text{do}(\alpha_j, L_j)$ if $t_1(Z_1) \ldots t_p(Z_p)$.

3.2 Confidentiality Policy

In order to enforce confidentiality, we define the Access Law and Modification Law to respectively control access to and modification of the database. These laws are sufficient conditions to enforce the following requirement expressed in section 2.3 “the database state at level $L$ must be determined by queries classified at most at level $L'$. These two laws are closely related to the strict Bell and LaPadula property.

Definition 4 (Access Law) If $p$ is a predicate and $l$ a security level, the Access Law is satisfied w.r.t. $(p, l)$ if the classification level of $p$ is lower than or equal to $l$.

Definition 5 (Modification Law) If $p$ is a predicate and $l$ a security level, the Modification Law is satisfied w.r.t $(p, l)$ if $l$ is lower than or equal to the classification level of $p$.

In order to enforce that users send queries at the right security level, we also define the User Privilege Law. This third Law must be satisfied for each query issued by the users.

Definition 6 (User Privilege Law) If $s$ is a user and $l$ is a security level, the User Privilege Law is satisfied w.r.t $(s, l)$ if $l$ is the clearance level of $s$.

3.3 Security Policy for Action Definitions

As presented in section 2.2, insecure information flows may occur because of insecure action definitions. Such flows occur when an action relies on conditions the user is not supposed to have access to or if the action launched at level $l$ is not allowed to perform the operation on the database. In order to control such information flows, we present security properties for the conditions, namely the Action Condition Security, and for the conditions, namely the Action Effect Security. The Action Effect Security consists in the enforcement of the confidentiality properties (definitions 4 and 5) for every access to the database. So let $\text{do}(\alpha, L)$ causes $op_1 \ldots op_k$ if $q_1(X_1) \ldots q_n(X_n)$ be a multilevel action definition.

Definition 7 (Action Condition Security) The execution of an action by user $s$ at level $L$ satisfies the Action Condition Security if for each condition $q_i(X_i)$, the Access Law is satisfied w.r.t $q_i(X_i)$ and $L$.

This requirement enforces that for any action $\alpha$ classified at level $l$, if $\alpha$ is executed in both states $s_i$ and $s'_i$, then $s_i \approx_L s'_i \implies s_{i+1} \approx_L s'_{i+1}$, as the action effect depends on conditions which are equivalent in $s_i$ and $s'_i$.

Definition 8 (Action Effect Security) The execution of an action by user $s$ at level $L$ satisfies the Action Effect Security if it satisfies (1) the User Privilege policy w.r.t $(s, L)$, (2) the Modification Law w.r.t $(p, L)$ if one $op$, operation is an insert of predicate $p$, (3) the Access Law w.r.t $(p, L)$ if one $op$, operation is a select of predicate $p$ and (4) both the Modification and Access Law w.r.t $(p, L)$ if one $op$, operation is a delete of predicate $p$.

This requirement enforces that for any action $\alpha$ executed in state $s_i$, if $\alpha$ is classified at level $L$, then for any level $L'$ such that $L \not\subseteq L'$, we have $s_i \approx_L s'_i \implies s_{i+1} \approx_L s'_{i+1}$.

3.4 Security Policy for Event Definitions

As shown in section 2.2, events may create insecure information flows. First, the execution of an event may disclose the associated condition states. This issue is taken into account by the Event Condition Security property. Second, in order to avoid flows that would reveal information about the initial action, we also state that the triggered actions must satisfy the Event Action Security property. So let $\text{event}_\text{name}(X, L_1)$ after $\text{do}(\alpha, L_2)$ if $r_1(Z_1) \ldots r_m(Z_m)$ be a multilevel event definition.
Definition 9 (Event Condition Security) The event definition satisfies the Action Condition Security if for each condition \( r_i(Z_i) \), the Access Law is satisfied w.r.t \( r_i(Z_i) \) and \( L_1 \).

Definition 10 (Event Action Security) The event definition satisfies the Event Action Security if the level \( L_2 \) of the action \( \alpha \) is lower than or equal to the event level \( L_1 \). These properties assure that for any action \( \alpha \) executed at level \( L_2 \) in two traces \( t_\alpha \) and \( t_\alpha' \), an event classified at level \( L_1 \) is triggered in both traces if \( s_i \approx s_i' \).

3.5 Security Policy for Active Rule Definitions
As for the action and the event definitions, active rule definitions have to satisfy several security properties to prevent insecure information flows. The first security property is the Active Rule Condition Security property, which says that the conditions of the active rule must be accessible at the execution security level before being checked. The second property is the Active Rule Action Security property. As for events, we must specify that actions launched by an event do not disclose any illegal information of the original action state. So let: rule_name : event_name(X, L_0) initiates do(\( \alpha_1, L_1 \)) ... do(\( \alpha_j, L_j \)) if \( t_1(Z_1) \) ... \( t_p(Z_p) \) be an active rule definition.

Definition 11 (Active Rule Condition Security) The active rule definition satisfies the Active Rule Condition Security if for each condition \( t_i(X_i) \), the Access Law is satisfied w.r.t \( t_i(X_i) \) and the greatest lower bound \( \text{glb}(L_1, ..., L_j) \).

Definition 12 (Active Rule Action Security) The active rule definition satisfies the Active Rule Action Security if the security level of the event \( L_0 \) is lower than or equal to each security action level \( L_1, ..., L_j \).

These properties assure that for any event classified \( L_0 \) that occurs in both traces \( t_\alpha \) and \( t_\alpha' \), any action \( \alpha_k \) classified \( L_k \) is triggered in both traces if \( s_i \approx s_i' \).

4. FORMAL SECURITY POLICY ENFORCEMENT AND SECURITY PROOFS
In the previous section, we define security policies that refine the Bell and La Padula policy when considering action \( \sim \) event and event \( \sim \) action information flows created by events and active rule execution in active databases. In order to assert that active databases based on the previous policies do not create illegal information flows, we shall now prove that these policies enforce the confidentiality property for active databases, as defined in section 3.2. For this purpose, we use the B Method [1].

4.1 The B Model
To prove our confidentiality property, we first develop a model for ECA Rule Specification using the B Method. This model, described in figure 3, is composed of safe and unsafe specifications called “machines” in the B Method. Each machine is named \( X \_\text{insecureY} \) (with \( X \) the prefix of the associated safe machine and \( Y \) an integer). The unsafe machines show that insecure traces may be generated for insecure implementations. Our model also relies on interfaces named \( X \_\text{event} \) used for the simulations with ProB [1].

We say that the result is YES_RESULT if the operation succeeded, NO_RESULT if it failed (such as a delete of an invalid predicate), SKIP_RESULT if it produced no results and SECURITY_SKIP if it failed for security reasons (such as unreadable conditions).

4.2 Confidentiality Policy Enforcement
As presented in section 4.2, we define the security policy for the access to the database predicates. Its enforcement relies on the ECA_policy machine with the specification of invariants on the select \( (\text{fun}_\text{can_read}) \), the insert \( (\text{fun}_\text{can_write}) \) and the delete \( (\text{fun}_\text{can_delete}) \) authorization. For instance, the invariant on \( \text{fun}_\text{can_delete} \) says that a delete is allowed if and only if it satisfies the Modification Law and Access Law properties: \( \forall (\text{user}, \text{item}, \text{lvl}) :: (\text{item} : \text{DATABASE} \land \text{lvl} : \text{LEVELS} \land \text{user} : \text{USERS}) \Rightarrow (\text{fun}_\text{can_delete}(\text{user}, \text{item}, \text{lvl}) = \text{TRUE} \Leftrightarrow \text{equal}_{\text{level}}(\text{lvl}, \text{classification}_{\text{level}}(\text{item}) = \text{TRUE}) \).

Based on these invariants, the proof of the policy relies on Assert predicates wherever the database instance is modified (in the ECA_actions machine). For instance, we assure that for each delete, \( \text{fun}_\text{can_delete} \) is satisfied. As we succeeded to prove all the proof obligations of the ECA_actions machine, the access control policy is assured for the action definitions.

4.3 Security Policy for Action Definitions
According to the definitions [2] and [3], we say that \( \text{do}(\alpha, L) \) defined by Def. [1] satisfies the active database policy if

\[
\begin{align*}
&\{ \text{Def. [1]} \} \\
&\forall x \in \{1, n\}, (\text{classification}_{\text{level}}(q_x(X_\alpha), L_\alpha) \land \text{inf}_{\text{equal}_{\text{level}}}(L_\alpha, L) = \text{TRUE}) \Rightarrow \text{inf}_{\text{equal}_{\text{level}}}(L_\alpha, L) = \text{TRUE} \\
&\{ \text{Def. [2]} \}
\end{align*}
\]

As presented in the previous section, definition [3] is satisfied on the ECA_actions machine. The remaining property to satisfy is thus the Action Condition Security. In order to prove this property, proof obligations are generated with \( \text{fun}_\text{can_read} \) for each condition. We illustrate the proof obligation generation in figure 3 of the appendix. The B-machine ECA_actionsHandle has been proved. For the associated B-machine ECA_actionsHandle_insecure4, which does not implement the controls related to the action conditions, the proof is not possible. Besides, using ProB, we generated insecure execution traces for this model, as shown in figure 3. In this figure, we show that the following delete rule may produce safe (step 5) or insecure information flows (step 4 and 7):

\[
\text{do} (\text{delete}(\text{p}), L) \text{ causes delete(\text{p}) if DB.L2}
\]

4.4 Security Policy for Event Definitions
As defined in section 4.4, events satisfy the security policies if they satisfy the definitions [9] and [10]. We thus say that in our model, event(\( X, L \)) defined by Def. [2] satisfies the active database policy if

\[
\begin{align*}
&\{ \text{Def. [9]} \} \\
&\forall x \in \{1, m\}, (\text{classification}_{\text{level}}(r_x(X_\alpha), L_\alpha) \land \text{inf}_{\text{equal}_{\text{level}}}(L_\alpha, L) = \text{TRUE}) \Rightarrow \text{inf}_{\text{equal}_{\text{level}}}(L_\alpha, L) = \text{TRUE} \\
&\{ \text{Def. [10]} \}
\end{align*}
\]
The B-machine \texttt{ECA\_eventCall} machine specifies event definitions and takes into account the security properties. It has been proved to satisfy the proof obligations associated to both condition accesses and the event security level. The associated B-machine \texttt{ECA\_eventCall\_insecure5}, which specifies event definitions without considering the security properties, cannot be proved. Besides, we can generate insecure execution traces, as shown in figure 5. In this figure, we show that the following ECA rules may produce safe (step 5) or insecure information flows (step 4 and 6):

\begin{verbatim}
  ev(p, L) after do(insert(p), L) if DB_I3
  ev(p, L) initiates do(insert(DB_I2), L)
\end{verbatim}

4.5 Security Policy for Active Rule Definitions

As defined in section 3.4, active rules satisfy the security policies if they satisfy the definitions 11 and 12. We thus say that in our model, active rules defined by Def. 9 satisfies the active database policy if

\begin{align*}
  &\text{(Def 11)} \forall x \in \{p\}, (\text{classification\_level}(t_x(X_x), L_c)) \\
  &\text{(Def 12)} \forall y \in \{k\}, \text{inf\_eq\_level}(L_c, L_y) = \text{TRUE})
\end{align*}

The B-machine \texttt{ECA\_eventAction} has been proved to satisfy the associated proof obligations while the B-machine \texttt{ECA\_eventAction\_insecure2} could not be proved.

4.6 Confidentiality property: Discussion

We specified the confidentiality property defined in section 2.3 in several steps. We first defined the database equivalence w.r.t level $L$, which specifies when two database states are equivalent. We then implemented the confidentiality property which controls the operations on the database, provided they are equivalent.

With atelierB, we proved that queries from user generate events and actions whose security levels are higher than or equal to $L$ and that the associated condition security levels are lower than or equal to $L$. These formal proofs assure that the formal properties we presented in section 3.4 are indeed satisfied by our implementation. Finally, we proved using inference systems that the database equivalence invariant and the aforementioned proofs are sufficient to ensure that our model satisfies the confidentiality property for two equivalent traces w.r.t to requests at level $L$. Thus, we assure that $\forall \alpha, \alpha', s_0 \sim_L s_0' \wedge \alpha[L] = \alpha'[L] \implies s_0 = s_0' \wedge t_k \sim_L t_k'$, where $t_k$ and $t_k'$ are the subtraces produced by the execution of the sequence of queries $\alpha$ and $\alpha'$.

In order to assert the confidentiality property for traces, as specified in section 2.3, we need a new function which complies with the sequential order required by the confidentiality property, as shown by the $B$-like formulae in figure 7. This function preserves the equivalence property and builds traces such that the sequential execution maintains the initial operation order and satisfies the equivalence property. Using this function and previous proofs, we thus proved the confidentiality property for active databases which relies on
our security policy.

5. RELATED WORKS

Database security is an active research domain and many works have already defined security models for different database models (relational, object oriented, deductive). In [5], Bertino and al. presented an overview of the database security challenges. They considered several issues like modeling of access control policy (System R, RBAC), management of multilevel databases and access control for recent database models (object oriented, XML). They also present various issues regarding privacy and IPR (Intellectual Property Rights) management.

In [6], Biskup and al. studied security issues for multilevel databases. Based on observations about the polynomial technique and its limitations (expressiveness for partial ordered security lattice, ambiguous interpretation), they proposed a new model for multilevel databases with an explicit definition of cover stories.

In [7], Sicherman and al. focused on modeling uncontrolled information leakage due to responses to queries. They showed that if attackers know the database policy, they may infer additional information, for instance in the case of a refusal based policy. Based on this observation, they defined a refusal based policy and proved its safety. In [8], Biskup and al. formally defined security properties for databases and proposed a model to enforce confidentiality for databases, which extends the work by [7]. They studied the impact of known or unknown policies. They also evaluate the consequences of the databases security responses (lying or refusal) and secrets modeling. In their study, they define the notion of information leakage as the possibility to infer a secret or a part of it. They have shown that if attackers know the database policy, they may infer additional information, for instance in the case of a refusal based policy.

In future works, we shall focus on the control of declassification actions for active databases [16]. We shall also consider constraints between predicates and their consequences on database security.

6. CONCLUSION

In this paper, we have presented, formalized and enforced security properties for active databases using the ECA rule paradigm. We first presented new information flows which rely on the condition and event concepts of the active databases. We then defined security properties for active databases and formalized a security policy which enforce them. Finally, we proved using the B Method that active databases which rely on our security policy are secure w.r.t to the defined confidentiality properties.

In other studies as the ones by Bell and LaPadula [11] and Goguen and Meseguer [13] models, insecure flows were considered for generic information systems. However, applications to (active) databases were up to our knowledge not considered. Thus, no concrete security policy for active databases with information flows has been proposed yet, as for security properties modeling for active databases and their comparisons with standard databases.

7. ACKNOWLEDGEMENTS

The study is partially funded by the Délégation Générale pour l’Armement under a DGA / CNRS grant

8. REFERENCES

APPENDIX

<table>
<thead>
<tr>
<th>Machine Name</th>
<th>St.</th>
<th>POs</th>
<th>Proved</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECA_actions</td>
<td>OK</td>
<td>10</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_actions_i</td>
<td>OK</td>
<td>12</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_actionsHandle</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_actionsHandle_1</td>
<td>OK</td>
<td>36</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_actionsHandle_event</td>
<td>OK</td>
<td>2</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_actionsHandle_insecure1</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_actionsHandle_insecure2</td>
<td>INC</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>ECA_actionsI</td>
<td>OK</td>
<td>18</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_causality</td>
<td>OK</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>ECA_causality_event</td>
<td>OK</td>
<td>5</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_causality_event_1</td>
<td>OK</td>
<td>28</td>
<td>20</td>
</tr>
<tr>
<td>ECA_databases_double</td>
<td>OK</td>
<td>17</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_databases_double_insecure1</td>
<td>INC</td>
<td>13</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_EventAction</td>
<td>OK</td>
<td>2</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_EventAction_i</td>
<td>OK</td>
<td>238</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_EventCall</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_EventCall_i</td>
<td>R</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ECA_EventCall_event</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_EventCall_event_insecure2</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_EventCall_insecure5</td>
<td>OK</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>ECA_EventCall_insecure2</td>
<td>INC</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>ECA_EventDefinition</td>
<td>OK</td>
<td>1</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_Parameters</td>
<td>OK</td>
<td>50</td>
<td>100 %</td>
</tr>
<tr>
<td>ECA_policy</td>
<td>OK</td>
<td>1895</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Table 1: Summary of the B Model Proofs

rep ← do(subject, action, predicate, lvl_do) \Rightarrow
PRE subject:USERS ∧ action:ACTIONS ∧ predicate:
DATABASE ∧ rep: RESULTS ∧ lvl_do:LEVELS THEN
IF (action = delete) THEN
  /*IF (fun_can_read(subject, DB_I1, lvl_do) = TRUE ∧
   fun_can_read(subject, DB_I2, lvl_do) = TRUE ) THEN*/
  IF (DB_I1:database ∧ DB_I2:database THEN
    user.user_addCond_knowledge(subject, {(DB_I1 \to
    lvl_do |->TRUE),(DB_I2 \to
    lvl_do |->TRUE)}) ||
    rep := DO_delete(subject, predicate, lvl_do, action)
ELSE
  user.user_addCond_knowledge(subject, {(DB_I1 \to
    lvl_do |->FALSE),(DB_I2 \to
    lvl_do |->FALSE)})
END
/*ELSE rep := SECURITY_SKIP END*/
...
END END.

Figure 6: Assertion failures for the Action Condition Security Property

Some of the proofs are considered as proved because they are trivial, though they were not fully proved with AtelierB.