



Expression and Enforcement of Confidentiality Policy in Active Databases



MEDES 2010 - Bangkok, Thailand

Julien A. Thomas, Nora Cuppens-Boulahia and Frédéric Cuppens

Institut Télécom ; Télécom Bretagne ; SERES Team
Université Européenne de Bretagne

26-29 October 2010

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

Outline

- 1 Context
- 2 Dynamic Multilevel Systems and Confidentiality
- 3 Multilevel Security Policy: Enforcement of the Confidentiality Property
- 4 Conclusion

Outline

1 Context

- Multilevel Information Systems
- Information Flows
- Dynamic Mechanisms

2 Dynamic Multilevel Systems and Confidentiality

3 Multilevel Security Policy: Enforcement of the Confidentiality Property

4 Conclusion

Multilevel Information Systems

- ➔ Multilevel Information Systems
 - ➔ Process information with different sensitivities
 - ➔ Permit simultaneous access by users with different security clearances
 - ➔ Prevent unauthorized accesses

Multilevel Information Systems

- ➔ Multilevel Information Systems
 - ➔ Process information with different sensitivities
 - ➔ Permit simultaneous access by users with different security clearances
 - ➔ Prevent unauthorized accesses
- ➔ Multilevel Information Systems rely on
 - ➔ Security levels and security level lattices
 - ➔ e.g. Confidentiel Défense \square Secret Défense \square Très Secret Défense
 - ➔ User and object assignments
 - ➔ e.g. classification levels (objects) and clearance levels (users)

Information Flows

➔ Definition

- ➔ An information flow is the transfer of information from a variable x to a variable y in a given *process*

Information Flows

➔ Definition

- ➔ An information flow is the transfer of information from a variable x to a variable y in a given *process*

➔ Information Flows and Security Policies

- ➔ Bell and La Padula [DL75]: confidentiality

- ➔ No Read Up and No Write Down

- ➔ No flow from x at level $L1$ to y of a level $L2$ with $L2 \sqsubset L1$

Information Flows

➔ Definition

- ➔ An information flow is the transfer of information from a variable x to a variable y in a given *process*

➔ Information Flows and Security Policies

- ➔ Bell and La Padula [DL75]: confidentiality
- ➔ Biba [Bib77]: integrity

➔ No Write Up and No Read Down

➔ No flow from x at level $L1$ to y of a level $L2$ with $L1 \sqsubset L2$

Information Flows

➤ Definition

- An information flow is the transfer of information from a variable x to a variable y in a given *process*

➤ Information Flows and Security Policies

- Bell and La Padula [DL75]: confidentiality
- Biba [Bib77]: integrity
- Non Interference [GM82]: confidentiality
 - «the set of events that P offers the low-level user are exactly the same as though the high-level user had never communicated anything»

$$(I1 = I2) \implies [[P]](h1, I1)_L = [[P]](h2, I2)_L$$

Information Flows

➔ Definition

- ➔ An information flow is the transfer of information from a variable x to a variable y in a given *process*

➔ Information Flows and Security Policies

- ➔ Bell and La Padula [DL75]: confidentiality
- ➔ Biba [Bib77]: integrity
- ➔ Non Interference [GM82]: confidentiality
- ➔ Causality [BC91a]: confidentiality

a predicate belongs to the knowledge of a user if the user is explicitly allowed to access the predicate or it causal depends on the knowledge of this user

Dynamic Mechanisms

⊕ Dynamic Mechanisms

⊕ Databases (Oracle [FP09]): Triggers

```
create or replace TRIGGER trigger AFTER INSERT on database
```

⊕ Security Policies (RBAC [SCFY96], OrBAC [ABB⁺03]): Contexts

```
rule(Role × Activity × View × Context)
```

⊕ An action $\alpha_1(P)$ engenders another action $\alpha_2(P_2)$, providing ...

⊕ ECA-rules (*Event-Condition-Action*) [DGG95], L_{active} language [BLT97]

Dynamic Mechanisms

- ➔ Dynamic Mechanisms
- ➔ ECA-rules (*Event-Condition-Action*) [DGG95], L_{active} language [BLT97]

- ➔ actions

```
do( $\alpha$ ) causes op1, ..., opk if q1(X1), ... qn(Xn) ;
```

- ➔ events

```
event_name(X) after do( $\alpha$ ) if r1(X1) rm(Xm) ;
```

- ➔ rules

```
rule: event_name(X) initiates do( $\alpha_1$ ), ... do( $\alpha_k$ )  
if t1(X1), ... tp(Xp);
```

Outline

1 Context

2 Dynamic Multilevel Systems and Confidentiality

- Information System Model
- Insecure Information Flows
- Confidentiality for Dynamic Multilevel Systems

3 Multilevel Security Policy: Enforcement of the Confidentiality Property

4 Conclusion

Information System Model

- ⊕ Information System Model: Active Databases
 - ⊕ Active entities, database states and database operations
 - ⊕ Logical view of the database: set of predicates
 - ⊕ Database state: set of fully instantiated predicates (called facts) p_i
 - ⊕ Dynamic rules: L_{active} language [BLT97]

Information System Model

- ⊕ Information System Model: Active Databases
 - ⊕ Active entities, database states and database operations
 - ⊕ Logical view of the database: set of predicates
 - ⊕ Database state: set of fully instantiated predicates (called facts) p_i
 - ⊕ Dynamic rules: L_{active} language [BLT97]
- ⊕ Multilevel Security Policy
 - ⊕ inf_level , $equal_level$ and inf_equal_level relations
 - ⊕ $clearance_level$ and $classification_level$ relations

known policy constraint

$$\forall (user : USERS, l : LEVELS).(classification_level(clearance_level(user, l), \perp)$$

$$\forall (p : DATABASE, l : LEVELS).(classification_level(classification_level(p, l), \perp)$$

Information System Model (2)

→ Example

→ Bank Scenario

- *client*(*IdClient*, *BankBranch*): client Information
- *account*(*IdClient*, *BankBranch*, *Val*): *Val* is the account balance
- *ageClient*(*IdClient*, *Val*): *Val* is the age of *IdClient*

→ Bank Rules

```
do(createClient(Client, Bank)) causes insert(client(Client, Bank)) ;
do(createAccount(IdClient, BankBranch, V)) causes
  insert(account(IdClient, BankBranch, V)) ;
newClient(Client, Bank) after do(createClient(Client, Bank))
```

- new clients are associated to new accounts

```
newClient(Client, Bank) initiates do(createAccount(Client, Bank, 0))
  if ageClient(Client, Age)  $\wedge$  Age > 20 ;
newClient(Client, Bank) initiates do(createAccount(Client, Bank, 50))
  if ageClient(Client, Age)  $\wedge$  Age  $\leq$  20 ;
```



Dynamic Multilevel Systems and Insecure Information Flows

⊕ First Example: Sensitive Conditions

- ⊕ The security policy aims at strongly protecting client privacy
 - ⊕ $client(C, B)$ and $account(C, B, V)$ are classified *Secret*
 - ⊕ $ageClient(C, V)$ is classified *TopSecret*
- ⊕ The agency proposes offers to young clients

```
do(createAccount(IdClient, BankBranch, 50))  
if ageClient(IdClient, Age)  $\wedge$  Age  $\leq$  20 ;
```

⊕ Consequences

- ⊕ information about $ageClient$ may flow to *Secret*
 - ⊕ *TopSecret* users may control $ageClient$ to create covert channels
- ⊕ Second Example: Unmanaged security level decreases
- ⊕ Insecure Information Flows: action, event, or active rule specification

Dynamic Multilevel Systems and Insecure Information Flows

- ⊕ First Example: Sensitive Conditions
- ⊕ Second Example: Unmanaged security level decreases
 - ⊕ The bank considers the usage of statistics
 - ⊕ $account(C, B, V)$ and $ageClient(C, V)$ are classified TS
 - ⊕ $client(C, B)$ and $accountForYoung(B, V)$
 - ⊕ Statistics are updated whenever an account is created

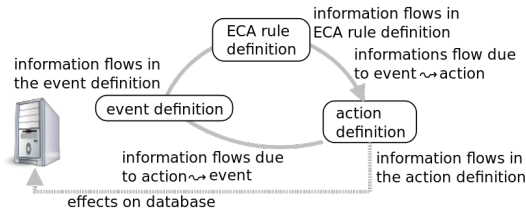
```

setStats(IdClient, BankBranch) after
do(createAccount(IdClient, BankBranch)) ; // TopSecret
setStats(IdClient(BankBranch)) initiates
do(youngStatsUpdate(BankBranch))
if ageClient(IdClient, Age)  $\wedge$  Age  $\leq$  20 ;
  
```

- ⊕ Consequences
 - ⊕ $account(C, B, V)$ is disclosed to S for people less than 20 years old
- ⊕ Insecure Information Flows: action, event, or active rule specifications

Dynamic Multilevel Systems and Insecure Information Flows

- ➔ First Example: Sensitive Conditions
- ➔ Second Example: Unmanaged security level decreases
- ➔ Insecure Information Flows: action, event, or active rule specifications



Information Flows Considerations

Confidentiality for Dynamic Multilevel Systems

- ⊕ Confidentiality Property, using traces [BC91b]
 - ⊕ A finite trace is noted $t_n = \{s_0, \alpha_n\}$
 - ⊕ s_0 : database initial state
 - ⊕ α_n : sequence of the n first actions executed in trace t
 - ⊕ s_n : resulting database state
 - ⊕ Trace equivalence: $t \sim_L t' \Leftrightarrow s_0 \approx_L s'_0 \wedge \alpha \upharpoonright L = \alpha' \upharpoonright L$

Confidentiality for Dynamic Multilevel Systems

⊕ Confidentiality Property, using traces [BC91b]

⊕ Trace equivalence: $t \sim_L t' \Leftrightarrow s_0 \approx_L s'_0 \wedge \alpha[L = \alpha']$

⊕ Standard Information Systems

① $(t, t' : TRACES, n, n' : INTEGER) \Rightarrow t_n \sim_L t'_{n'} \implies s_n \approx_L s'_{n'}$
if the actions and the initial state are equivalent
at L , the results are equivalent at L

Confidentiality for Dynamic Multilevel Systems

⊕ Confidentiality Property, using traces [BC91b]

⊕ Trace equivalence: $t \sim_L t' \Leftrightarrow s_0 \approx_L s'_0 \wedge \alpha \upharpoonright L = \alpha' \upharpoonright L$

⊕ Standard Information Systems

① $(t, t' : TRACES, n, n' : INTEGER) \Rightarrow t_n \sim_L t'_{n'} \implies s_n \approx_L s'_{n'}$

⊕ Dynamic Information Systems

⊕ α_n : user queries ($ext(\alpha_n)$) and the triggered events and actions

⊕ Insecure Information flows:

⊕ (Access Control Issue) $t_n \sim_L t'_{n'} \wedge s_n \not\approx_L s'_{n'}$

⊕ (Security Issue) $s_0 \approx_L s'_0 \wedge ext(\alpha_n) \upharpoonright L = ext(\alpha'_{n'}) \upharpoonright L \wedge s_n \not\approx_L s'_{n'}$

① $t_n \sim_L t'_{n'} \implies s_n \approx_L s'_{n'}$

② $s_0 \approx_L s'_0 \wedge ext(\alpha_n) \upharpoonright L = ext(\alpha'_{n'}) \upharpoonright L \implies t_n \sim_L t'_{n'}$

Outline

1 Context

2 Dynamic Multilevel Systems and Confidentiality

3 Multilevel Security Policy: Enforcement of the Confidentiality Property

- Extended Information System Model
- Security Policy Principles
- Security Policy Specifications
- Validation of the Multilevel Security Policy

4 Conclusion

Extended Information System Model

ECA Rules [DBM88]

$do(\alpha)$ **causes** $f(Y)$ **if** $q_1(X_1), \dots, q_n(X_n)$

$event_name(X)$ **after** $do(\alpha_1)$ **if** $r_1(X_1), \dots, r_m(X_m)$

$rule_name : event_name(X)$ **initiates** $do(\alpha_2), \dots, do(\alpha_k)$ **if** $t_1(X_1), \dots, t_m(X_m)$

Multilevel Information System Issues

ECA Rules Security

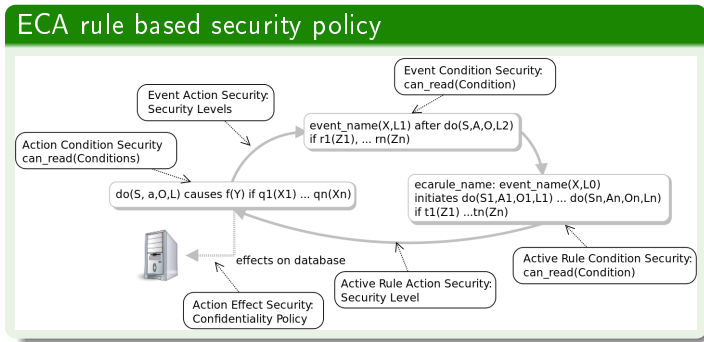
$do(\alpha, L)$ *causes* $f(Y)$ *if* $q_1(X_1) \dots q_n(X_n)$

$event_name(X, L_1)$ *after* $do(\alpha, L_2)$ *if* $r_1(Z_1) \dots r_m(Z_m)$

$rule_name : event_name(X, L_0)$ *initiates* $do(\alpha_1, L_1) \dots do(\alpha_n, L_n)$
if $t_1(Z_1) \dots t_m(Z_m)$

Security Policy Principles

- ➔ Definition of a relevant security policy
 - ➔ Enforcement of confidentiality for dynamic information systems
 - ➔ Enforcement/Definition of the ECA-rules security mechanism



Security Policy Specifications

- ⊕ Confidentiality policy (Bell and La Padula)

Access Law

Access Law w.r.t (o,l) : the classification level of o is less sensitive than l

Modification Law

Modification Law w.r.t (o,l) : l is less sensitive than the classification level of o

Security Policy Specifications

- ⊕ Confidentiality policy (Bell and La Padula)
- ⊕ Action security

Action Condition Security - Information Flows

for each condition $q_i(X_i)$, the Access Law is satisfied w.r.t $q_i(X_i)$ and L

Confidentiality

$$do(\alpha(L), s_i) \wedge do(\alpha(L), s'_i) \wedge s_i \approx_L s'_i \implies s_{i+1} \approx_L s'_{i+1}$$

Action Effect Security - Security Level Leverage

- 1 User Privilege policy w.r.t (s, L) and
- 2 Modification Law w.r.t (p, L) (*insert*), Access Law w.r.t (p, L) (*select*) or Modification and Access Laws w.r.t (p, L) (*delete*)

Confidentiality

$$do(\alpha(L'), s_i) \wedge L' \not\sqsubseteq L \wedge s_i \approx_L s'_i \implies s_{i+1} \approx_L s'_{i+1}$$

Security Policy Specifications

- ⊕ Confidentiality policy (Bell and La Padula)
- ⊕ Action security
- ⊕ Event security

Event Condition Security - Information Flows

for each condition $r_i(Z_i)$, the Access Law is satisfied w.r.t $r_i(Z_i)$ and L_1

Event Action Security - Security Level Leverage

the level L_2 of the action α is lower than or equal to the event level L_1

Confidentiality

for any action α executed at level L_2 in two traces t_i and t'_i , an event classified at level L_1 is triggered in both traces if $s_i \approx_{L_1} s'_i$

Security Policy Specifications

- ⊕ Confidentiality policy (Bell and La Padula)
- ⊕ Action security
- ⊕ Event security
- ⊕ Active rule security

Active Rule Condition Security - Information Flows

for each condition $t_i(X_i)$, the Access Law is satisfied w.r.t $t_i(X_i)$ and l

Active Rule Action Security - Security Level

the security level of the event is less sensitive or equal to the action security levels

Confidentiality

for any event classified L_0 that occurs in both traces t_i and t_i' , any action α_k classified L_k is triggered in both traces if $s_i \approx_{L_k} s_i'$

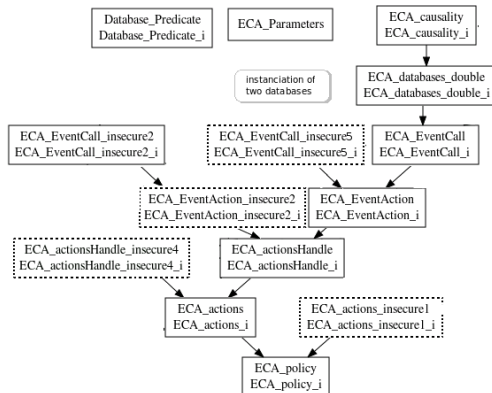
First Order Logic Modeling

⊕ B Method

- ⊕ safe and unsafe specifications called “machines“
- ⊕ unsafe specifications named $X_insecureY$
- ⊕ interfaces named X_event used for the simulations with *ProB* [LB08]

⊕ Objectives

- ⊕ Access Control Policy (Laws) Modeling
- ⊕ Security Policy Modeling
- ⊕ Proofs



Intermediate Invariant Proofs and Information Flows

- ⊕ Insecure models are unprovable
 - ⊕ Proof failures are associated to information flows
 - ⊕ ProB tool [LB08]

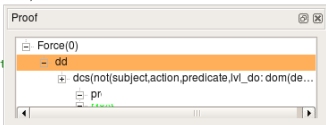
Intermediate Invariant Proofs and Information Flows

- ⊕ Insecure models are unprovable
 - ⊕ Proof failures are associated to information flows
 - ⊕ ProB tool [LB08]
- ⊕ *AtelierB* based Proofs

Action Action Security (Security Level Leverage)

```

action = delete &
  lvl_event_delete: LEVELS &
  subject,action,predicate,lvl_do: dom(decisionget_eventLevel) => lvl_event_delete =
decisionget_eventLevel(subject,action,predicate,lvl_do) &
  not(subject,action,predicate,lvl_do: dom(decisionget_eventLevel)) =>
inf_equal_level(lvl_do, lvl_event_delete) = TRUE &
  not(fun_can_read(subject,DB_I6, lvl_event_delete) = TRUE) &
  fun_can_read(subject,DB_I7, lvl_event_delete) = TRUE &
  DB_I7: dbinst_in &
  not(DB_I7 = predicate) &
  "Check that the invariant (btrue) is preserved by
=>
  inf_equal_level(lvl_do, lvl_event_delete) = TRUE
  -----
  inf_equal_level(lvl_do, lvl_event_delete) = TRUE
  -----
  not(subject,action,predicate,lvl_do: dom(decisionget_eventLevel))
or subject,action,predicate,lvl_do: dom(decisionget_eventLevel)
  -----
  subject,action,predicate,lvl_do: dom(decisionget_eventLevel) & btrue
=>
  inf_equal_level(lvl_do, decisionget_eventLevel(subject,action,predicate,lvl_do)) = TRUE
  
```



Proofs for the Confidentiality Property

- ⊕ With *atelierB*, no matter the state of the database,
 - ⊕ generated events and actions have security levels higher than or equal to the initial one
 - ⊕ associated condition have security levels lower than or equal to the component level
 - ⊕ the security policy is well enforced in our implementation

Proofs for the Confidentiality Property

- ⊕ With *atelierB*, no matter the state of the database,
 - ⊕ generated events and actions have security levels higher than or equal to the initial one
 - ⊕ associated condition have security levels lower than or equal to the component level
 - ⊕ the security policy is well enforced in our implementation
- ⊕ Using inference systems and the aforementioned proofs
 - ⊕ the confidentiality property for two equivalent user queries is satisfied

Proofs for the Confidentiality Property

- ⊕ With *atelierB*, no matter the state of the database,
 - ⊕ generated events and actions have security levels higher than or equal to the initial one
 - ⊕ associated condition have security levels lower than or equal to the component level
 - ⊕ the security policy is well enforced in our implementation
- ⊕ Using inference systems and the aforementioned proofs
 - ⊕ the confidentiality property for two equivalent user queries is satisfied
- ⊕ Based on the sequential execution of the database queries,
 - ⊕ a new function refines traces such that the execution maintains the initial operation order and satisfies the equivalence properties

Outline

- 1 Context
- 2 Dynamic Multilevel Systems and Confidentiality
- 3 Multilevel Security Policy: Enforcement of the Confidentiality Property
- 4 Conclusion

Conclusion

- ➔ Contributions
 - ➔ Formalization of the confidentiality property for active databases

Conclusion

⊕ Contributions

- ⊕ Formalization of the confidentiality property for active databases
- ⊕ Enforcement of confidentiality for active databases
 - ⊕ Definition of an associated security policy
 - ⊕ Usage of the ECA rule [DBM88] paradigm

Conclusion

⊕ Contributions

- ⊕ Formalization of the confidentiality property for active databases
- ⊕ Enforcement of confidentiality for active databases
 - ⊕ Definition of an associated security policy
 - ⊕ Usage of the ECA rule [DBM88] paradigm
- ⊕ Proofs of Enforcement
 - ⊕ First Order Logic based Specifications
 - ⊕ B Method [Abr96] and Inference Systems based Proofs

Conclusion





⊕ Contributions

- ⊕ Formalization of the confidentiality property for active databases
- ⊕ Enforcement of confidentiality for active databases
 - ⊕ Definition of an associated security policy
 - ⊕ Usage of the ECA rule [DBM88] paradigm
- ⊕ Proofs of Enforcement
 - ⊕ First Order Logic based Specifications
 - ⊕ B Method [Abr96] and Inference Systems based Proofs




⊕ Future Works

- ⊕ Control of declassification actions for active databases
- ⊕ Consistency issues due to constraints between predicates





References I

-  A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin, *Organization Based Access Control*, 4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03), June 2003.
-  J.-R. Abrial, *The B-book: Assigning programs to meanings*, Cambridge University Press, August 1996.
-  P. Bieber and F. Cuppens, *A definition of secure dependencies using the logic of security*, Computer Security Foundations Workshop IV, 1991. Proceedings, 18-20 1991, pp. 2 –11.
-  Pierre Bieber and Frédéric Cuppens, *A definition of secure dependencies using the logic of security*, CSFW, 1991, pp. 2–11



References II

-  Biba, *Integrity Considerations for Secure Computer Systems*, MITRE Co., technical report ESD-TR 76-372 (1977).
-  Chitta Baral, Jorge Lobo, and Goce Trajcevski, *Formal Characterizations of Active Databases: Part II*, DOOD, 1997, pp. 247–264.
-  U. Dayal, A. P. Buchmann, and D. R. McCarthy, *Rules are objects too: A knowledge model for an active, object-oriented databasesystem*, Lecture notes in computer science on Advances in object-oriented database systems (New York, NY, USA), Springer-Verlag New York, Inc., 1988, pp. 129–143.

References III

-  Klaus R. Dittrich, Stella Gatzui, and Andreas Geppert, *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*, *Rules in Database Systems*, Springer, 1995, pp. 3–20.
-  D. Bell and L. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, Technical Report ESD-TR-75-306, MTR-2997, 1975.
-  Steven Feuerstein and Bill Pribyl, *Oracle pl/sql programming: Covers versions through oracle database 11g release 2*, O'Reilly Media, Inc., 2009.
-  J. A. Goguen and J. Meseguer, *Security policies and security models*, *Security and Privacy*, IEEE Symposium on **0** (1982), 11.

References IV

-  Michael Leuschel and Michael Butler, *ProB: An Automated Analysis Toolset for the B Method*, Journal Software Tools for Technology Transfer (2008).
-  R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, *Role-based access control models*, Computer **29** (1996), no. 2, 38–47.

Outline

5 B Method Modeling

6 B Method Proofs

First Order Logic Modeling

➔ Access Control

Modeling Modification Law

```
rep,dbinst_out ← DO_insert(subject, predicate,lv_do,action,dbinst_in)  $\triangleq$  PRE ...  
THEN  
  IF ( fun_can_write(subject, predicate,lv_do)=TRUE) THEN  
    dbinst_out := dbinst_in  $\cup$  {predicate} || rep := YES_RESULT  
  ELSE rep := SECURITY_SKIP END END;
```

First Order Logic Modeling

- ➔ Access Control
- ➔ Flow Control

Modeling Action Condition Security

```

rep,dbinst_out ← do(subject, action, predicate, lvl_do,dbinst_in)  $\triangleq$  PRE... THEN ..
IF ( fun_can_read(subject, DB_I2, lvl_do) = TRUE ) THEN
  IF( DB_I2:dbinst_in ) THEN
    user.user_add_cond_knowledge(subject, { (DB_I2  $\mapsto$  lvl_do  $\mapsto$ 
TRUE)}) || rep,dbinst_out ← DO_delete(subject, predicate,lvl_do,action,dbinst_in)
    ELSE user.user_add_cond_knowledge(subject, { (DB_I2  $\mapsto$  lvl_do  $\mapsto$ 
FALSE)}) || rep := NO_RESULT || dbinst_out := dbinst_in
  END
ELSE rep := SECURITY_SKIP || dbinst_out := dbinst_in END END;

```

Outline

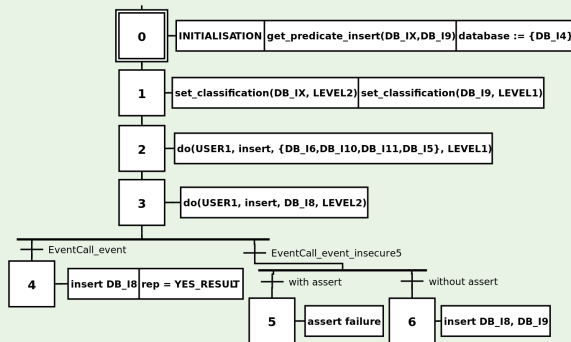
5 B Method Modeling

6 B Method Proofs

Intermediate Invariant Proofs and Information Flows

- ⊕ Insecure models are unprovable
- ⊕ Proof failures are associated to information flows
 - ⊕ ProB tool [LB08]

Security levels and event definitions



- ⊕ MLS System:

entity	level
USER1	LEVEL1
DB_I8	LEVEL1
DB_I9	LEVEL2

- ⊕ $LEVEL1 \not\sqsubseteq LEVEL2$
- ⊕ $LEVEL2 \sqsubset LEVEL1$
- ⊕ the insertion of DB_I8 triggers the insertion of DB_I9

Security Property Proofs

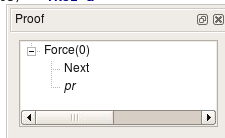
➔ AtelierB based Proofs

➔ Actions Condition Security (Information Flows on Conditions)

```

inf_equal_level(lvl_do, lvl_event_delete) = TRUE &
  not(fun_can_read(subject, DB_I6, lvl_event_delete) = TRUE) &
  fun_can_read(subject, DB_I7, lvl_event_delete) = TRUE &
  DB_I7: dbinst_in &
  not(DB_I7 = predicate) &
  item: DATABASE &
  lvl: LEVELS &
  status: BOOL &
  status = TRUE &
  lvl = lvl_event_delete &
  item = DB_I7 &
  "Check that the invariant (btrue) is preserved by the operation - ref 3.4"
=>
  fun_can_read(subject, item, lvl) = TRUE

```



Security Property Proofs

⊕ AtelierB based Proofs

- ⊕ Actions Condition Security (Information Flows on Conditions)
- ⊕ Action Action Security (Security Level Leverage)

```

action = delete &
lvl_event_delete: LEVELS &
subject,action,predicate,lvl_do: dom(decisionget_eventLevel) => lvl_event_delete =
decisionget_eventLevel(subject,action,predicate,lvl_do) &
not(subject,action,predicate,lvl_do: dom(decisionget_eventLevel)) =>
inf_equal_level(lvl_do,lvl_event_delete) = TRUE &
not(fun_can_read(subject,DB_I6,lvl_event_delete) = TRUE) &
fun_can_read(subject,DB_I7,lvl_event_delete) = TRUE
DB_I7: dbinst_in &
not(DB_I7 = predicate) &
"Check that the invariant (btrue) is preserved by t
=>
inf_equal_level(lvl_do,lvl_event_delete) = TRUE
-----
inf_equal_level(lvl_do,lvl_event_delete) = TRUE
-----
not(subject,action,predicate,lvl_do: dom(decisionget_eventLevel))
or subject,action,predicate,lvl_do: dom(decisionget_eventLevel)
-----
subject,action,predicate,lvl_do: dom(decisionget_eventLevel) & btrue
=>
inf_equal_level(lvl_do,decisionget_eventLevel(subject,action,predicate,lvl_do)) = TRUE

```

