Model Based Testing for Security Checking

Wissam Mallouli and Prof. Ana Cavalli
National Institute of Telecommunications, France
November 21, 2007

Outline

- Introduction
- Active/Passive Testing
- Active Testing Technique
  - Preliminaries
  - An integration based approach
  - The integration methodology
  - Use case : a Weblog
- Passive Testing Technique
  - Ongoing Work
- Conclusion

Introduction and motivation

- Security as critical issue
- Need to define a security policy
- A security policy is a set of rules that regulates the nature and the context of actions that can be performed within a system, according to specific roles.
- If the one of rules in the security policy is not respected, all the system can be vulnerable.

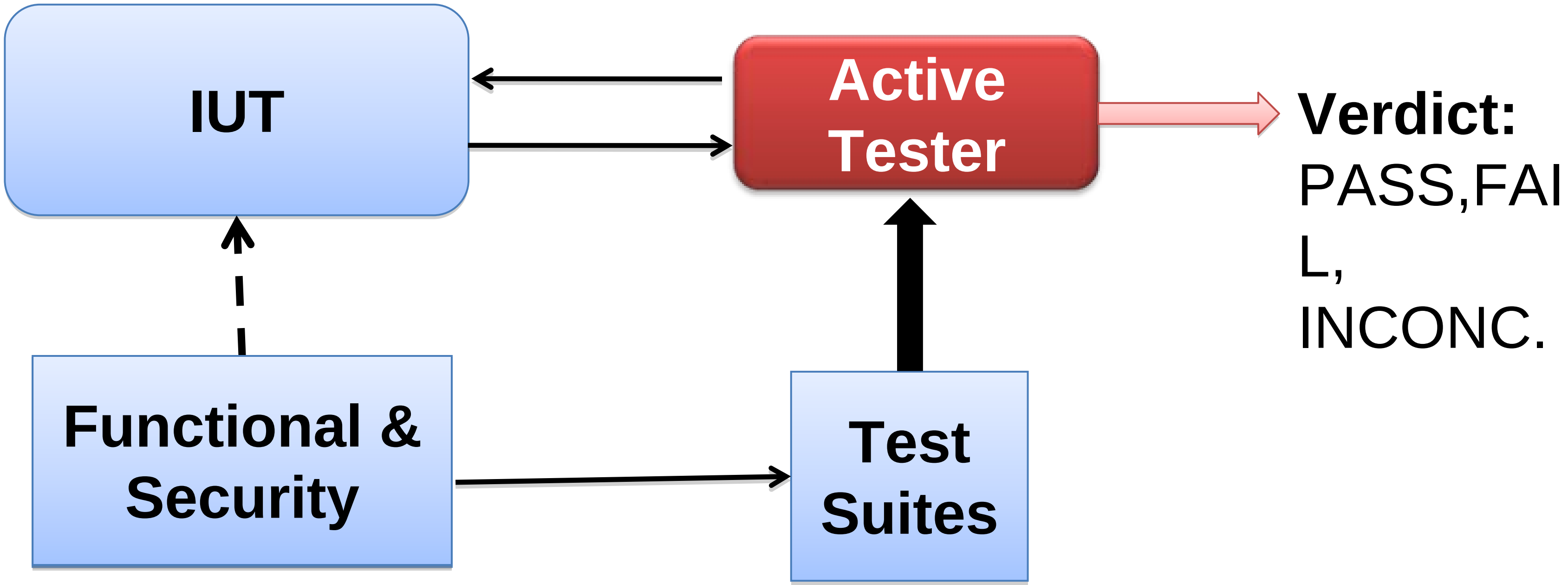
Introduction and motivation

- Checking if a system implements its security policy
  - Generating proofs
  - Injecting the policy within the system implementation
  - Model Based testing methods
  - etc

Outline

- Introduction
- Active/Passive Testing
- Active Testing Technique
  - Preliminaries
  - An integration based approach
  - The integration methodology
  - Use case : a Weblog
- Passive Testing Technique
  - Ongoing Work
- Conclusion

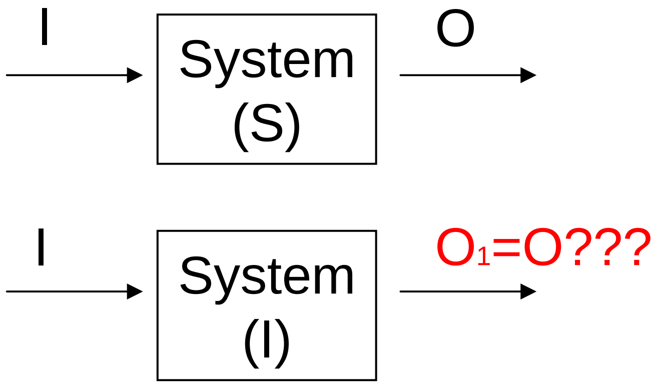
Active Testing



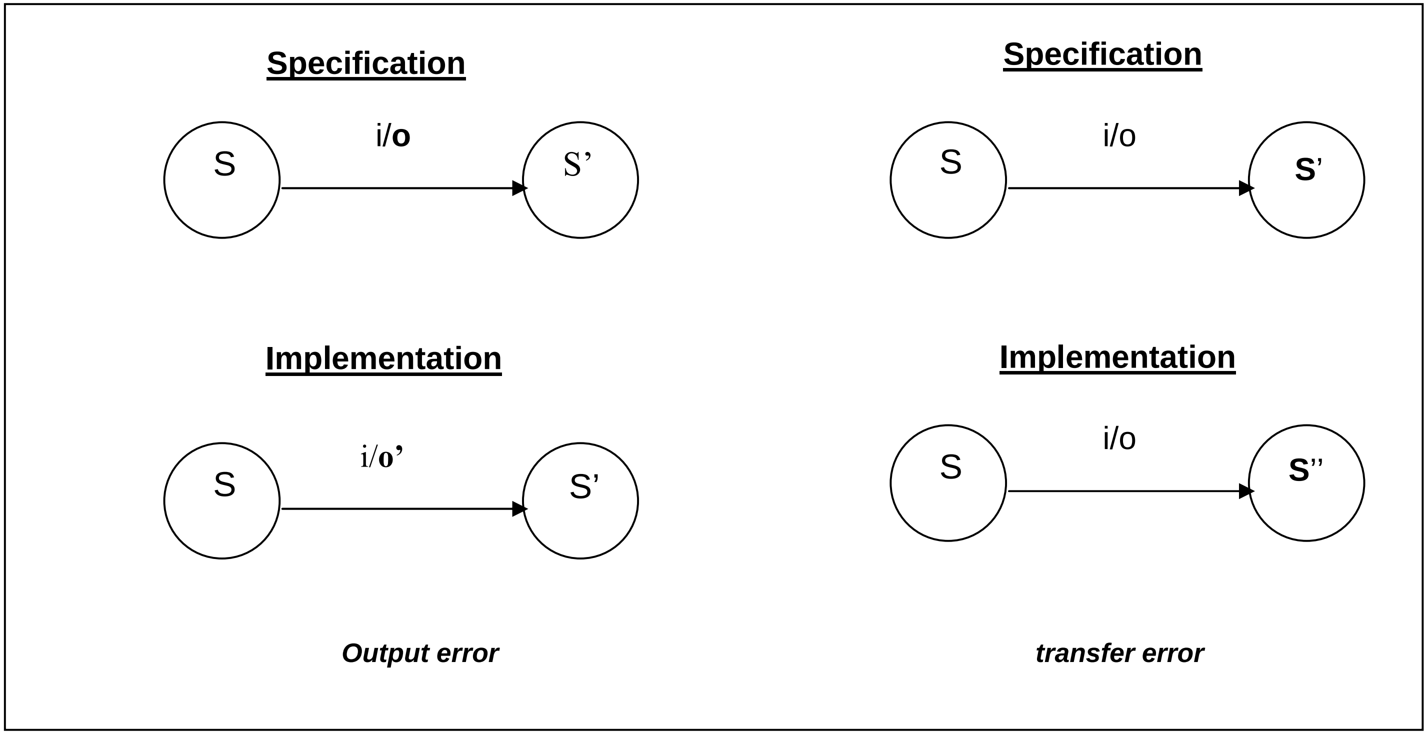
Automatic test generation based on formal descriptions

Conformance Testing(1/2)

- Check if the implementation of a system conforms to its specification



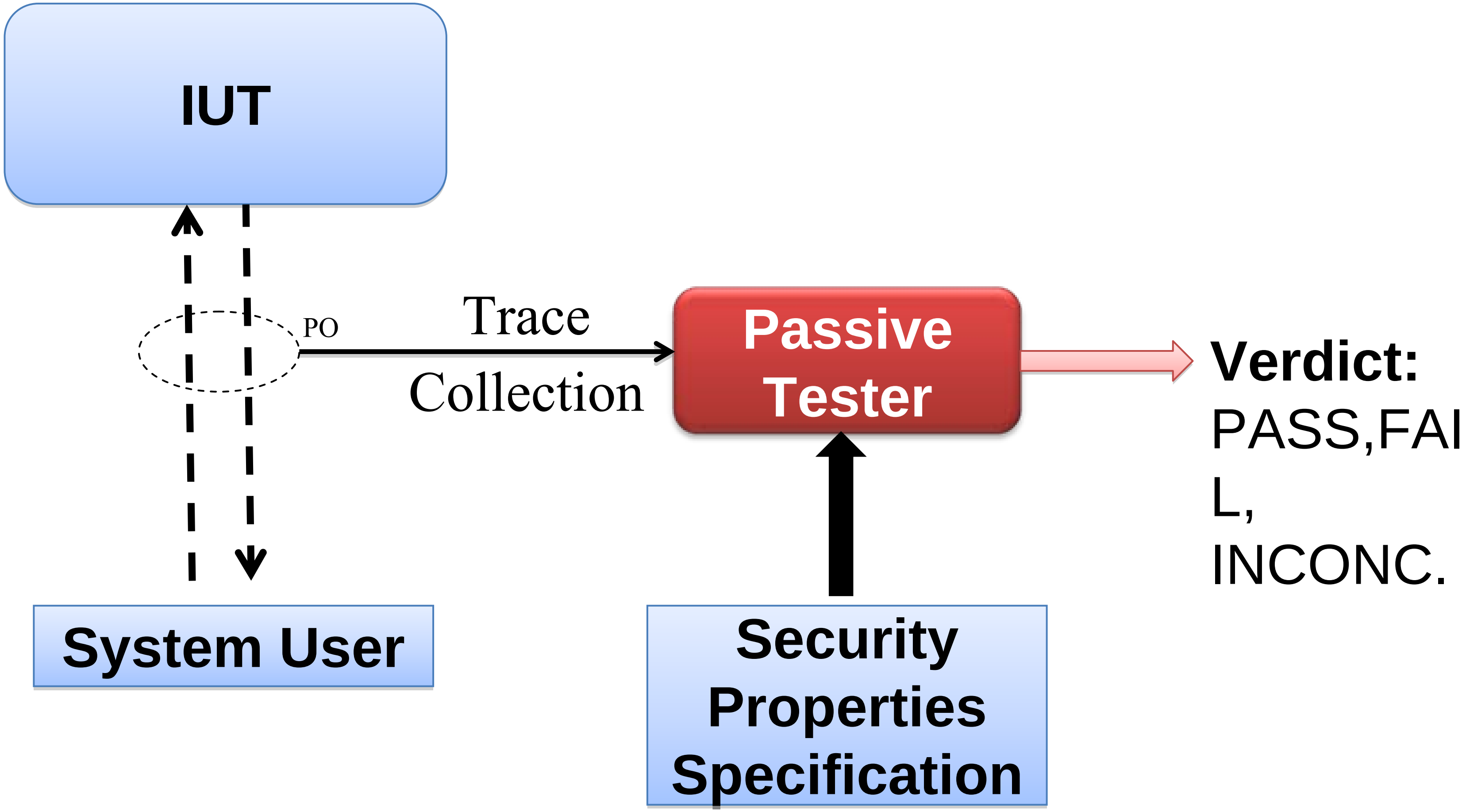
Conformance Testing (2/2)



Generation of a : - reasonable test scenarios number (Execution)

- Complete (to cover all the system transitions)

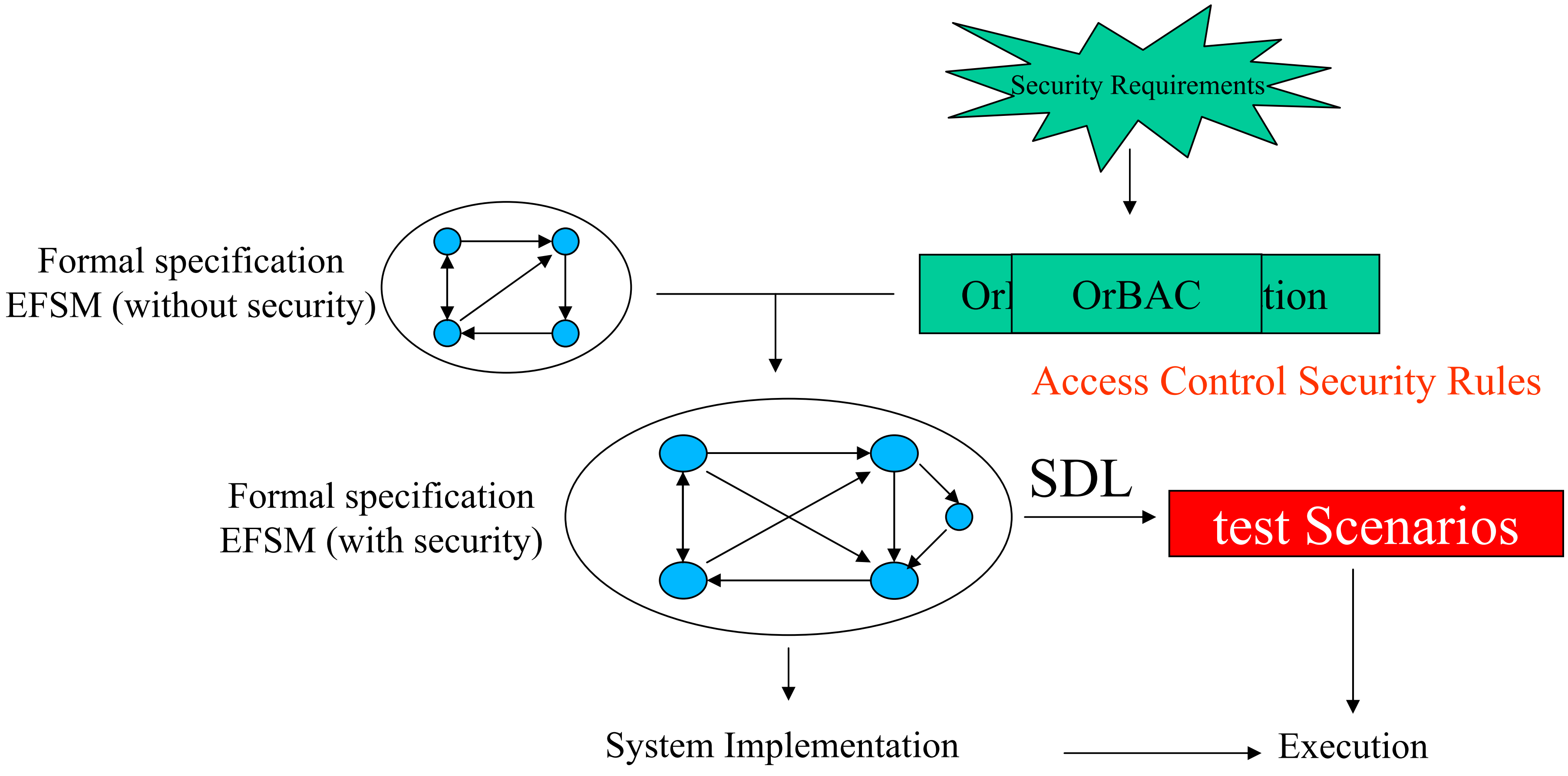
Passive Testing



Outline

- Introduction
- Active/Passive Testing
- Active Testing Technique
  - Preliminaries
  - An integration based approach
  - The integration methodology
  - Use case : a Weblog
- Passive Testing Technique
  - Ongoing Work
- Conclusion

Problem Inputs/Output



Security Requirements

Formal specification EFSM (without security)

Or OrBAC tion

Access Control Security Rules

Formal specification EFSM (with security)

SDL

test Scenarios

System Implementation

Execution

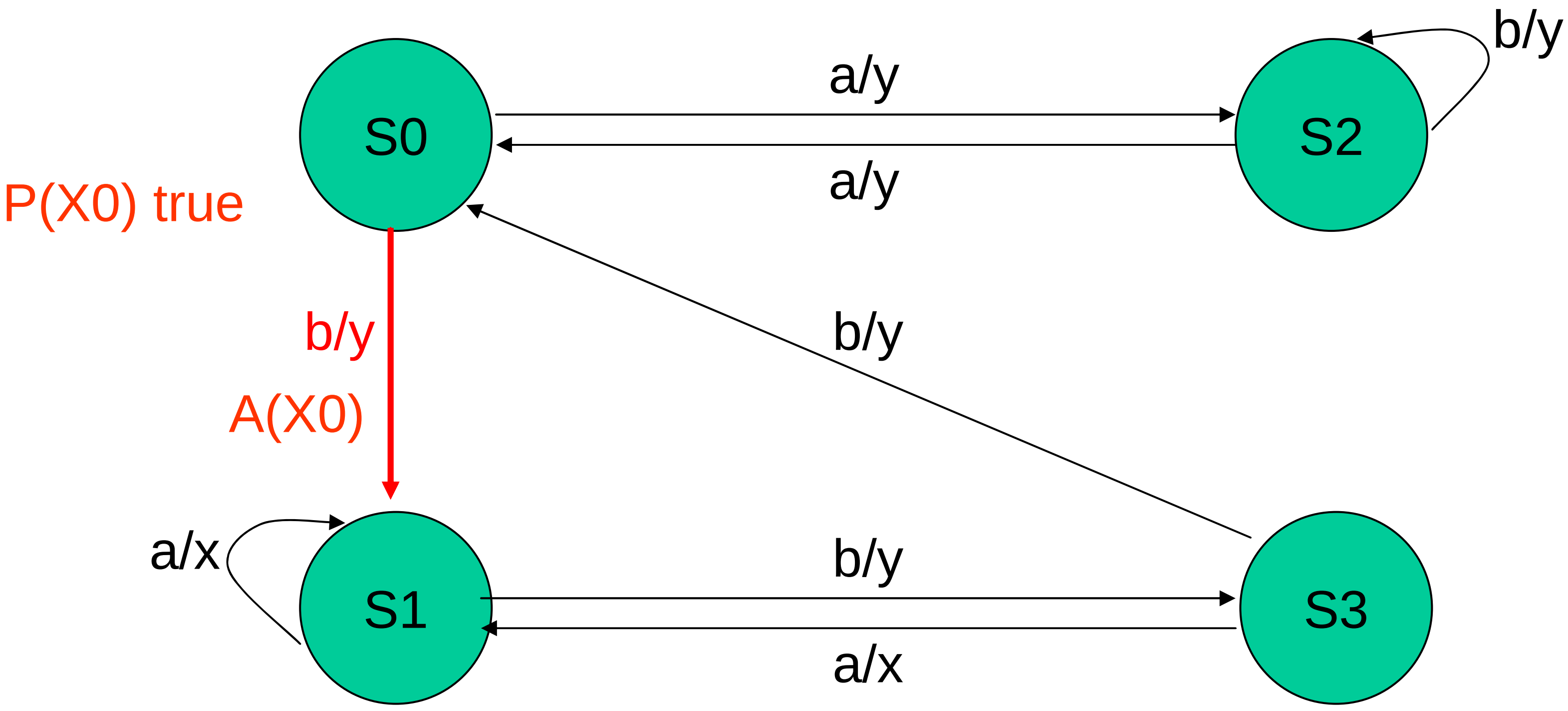
EFSM Formalism (1/2)

- Extended Finite States Machine is a 6-tuple M=(I,O,S_0,S,û,T) where:
  - I is a non empty set of input symbols
  - O is a non empty set of output symbols

An EFSM is an automaton with variables and predicates

-
  - s is the current state
  - q is the next state
  - i □ I is an input symbol
  - o □ O is an output symbol
  - P(û) is a predicate on the current values of the variables
  - A(û) is a sequence of actions over the variables

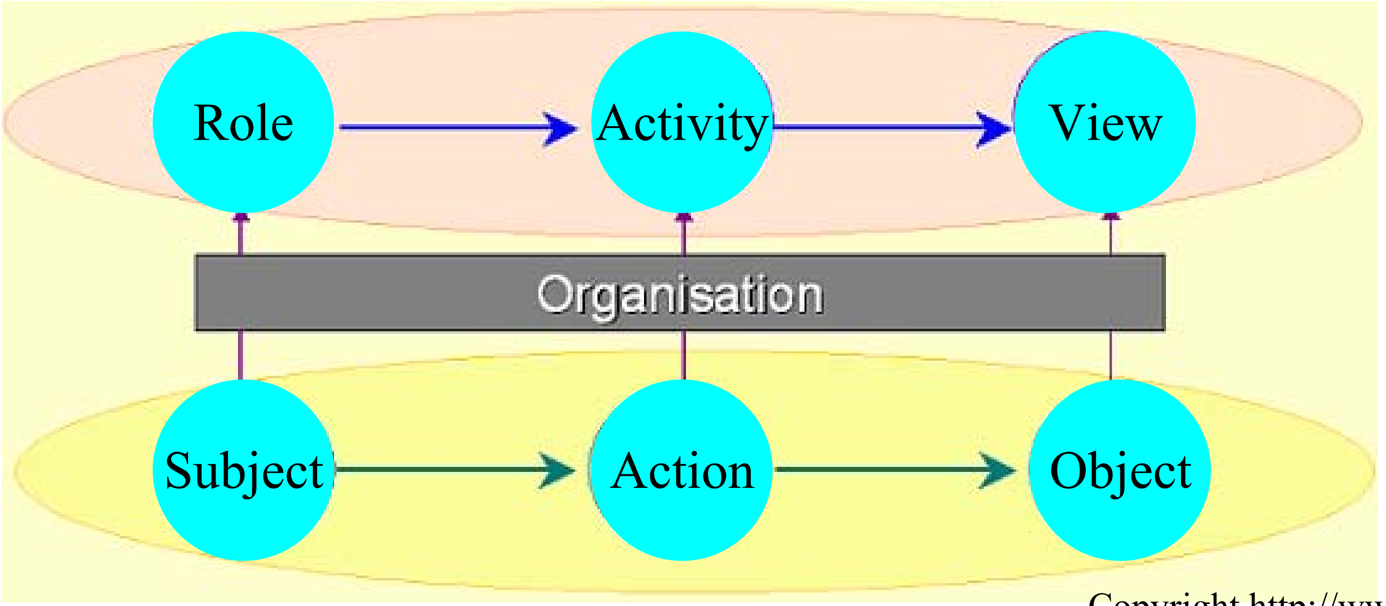
EFSM Formalism (2/2)



S=(S0,S1,S2,S3) I=(a,b) O=(x,y)

Orbac (1/2)

- An access and usage control model
- Obligation/Permission/Prohibition



Copyright http://www.orbac.org

Orbac (2/2)

- Permission/Prohibition/Obligation (S,R,A,V,C)

- This rule means that within the system *S*, the role *R* is permitted/prohibited/obliged to perform the activity *A* targeting the objects of view *V* in the context *C*.

Orbac Interpretation to Fit the EFSM Formalism (1/2)

- Permission (system1, role1, call delete, text, input=req_delete(text) and text_exists=true)

- The activity and the context have to be described in the same language of the functional specification of the system.
- In our case, we used SDL language and call and input= are SDL commands

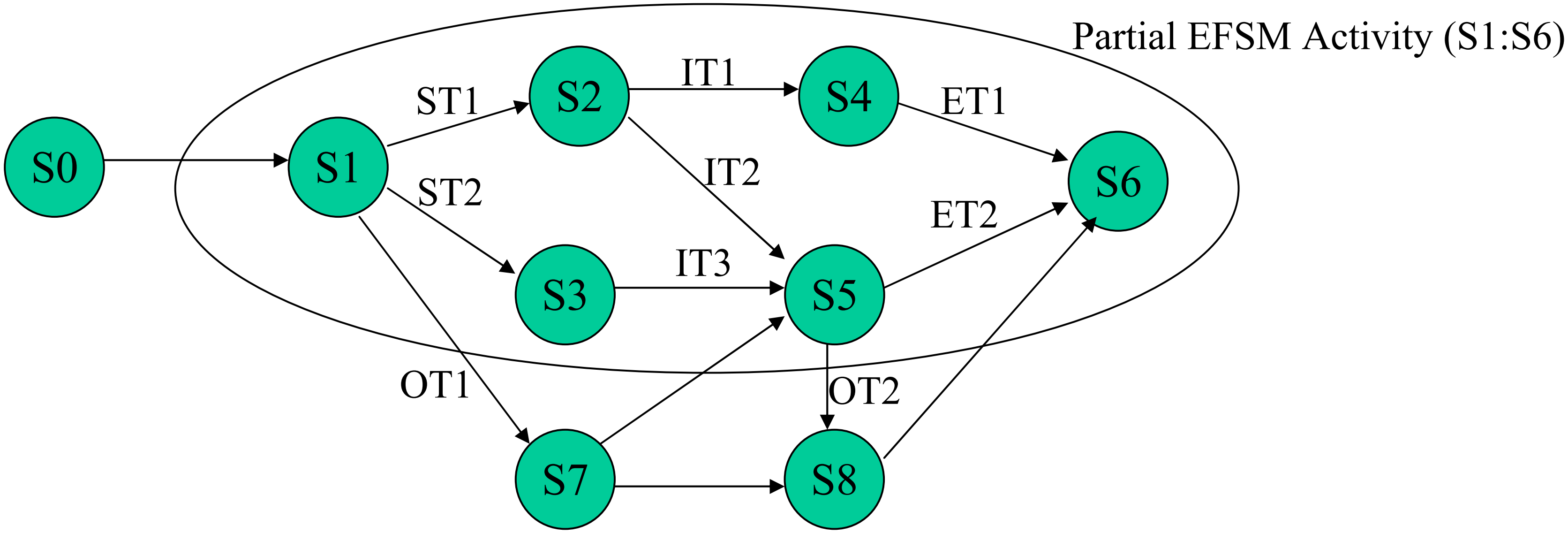
Orbac Interpretation to Fit the EFSM Formalism (2/2)

- If the roles and variables are not already defined in the initial specification, precise definitions have to be added (type, default value, etc.).
- A rule context is divided into two parts:
  - an EFSM context with conditions related to the position in the EFSM (e.g. input=a)
  - a variables context with conditions related to variables values (e.g. $variable_1=0$).

Activity Definition

- refers to a possible action within the EFSM functional description of the system. It can be either :
  - An Atomic Activity : is a basic part of an EFSM transition. It is defined as an SDL command like an input, a task or an output etc.
  - A Decomposable Activity : is an activity which can be composed of a set of atomic activities.
    - It can correspond to one transition (1_tr activity) or to a set of transitions (n_tr activity)

Decomposable Activity



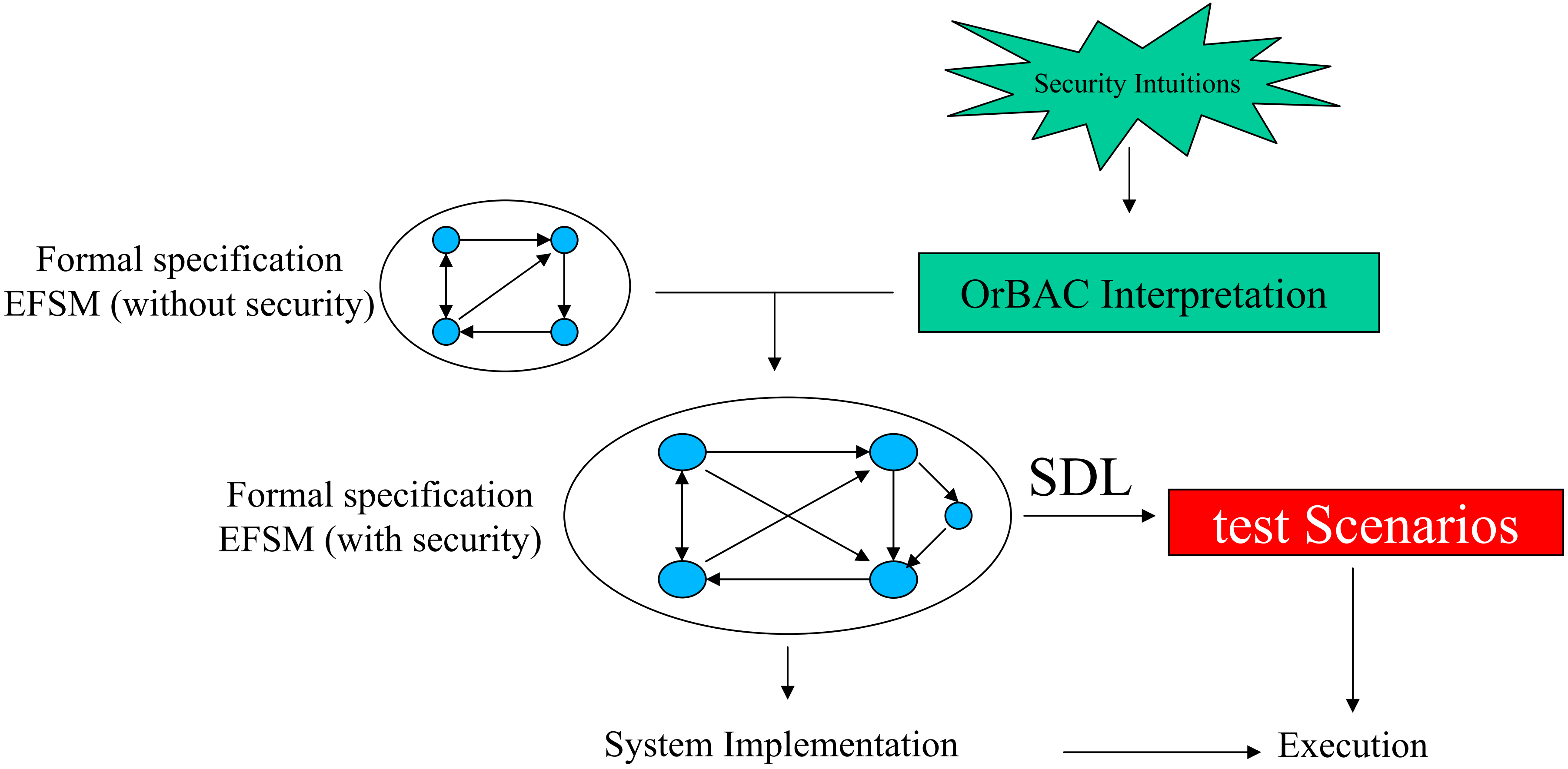
ST : Starting Transition

IT : Intermediate Transition

ET : Ending Transition

OT : Outgoing Transition

Our approach main idea



Security Intuitions

Formal specification
EFSM (without security)

OrBAC Interpretation

Formal specification
EFSM (with security)

SDL

test Scenarios

System Implementation

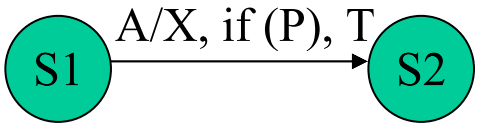
Execution

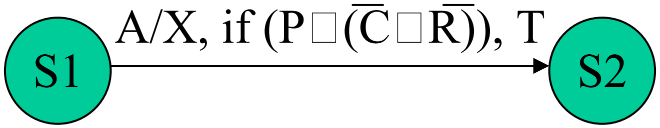
Integration methodology

- To parse the EFSM specification
- For each transition, to identify the rules that
    - map the activity and the EFSM context in the case of permissions and prohibitions
    - map the EFSM context in the case of obligations

IM : Prohibition

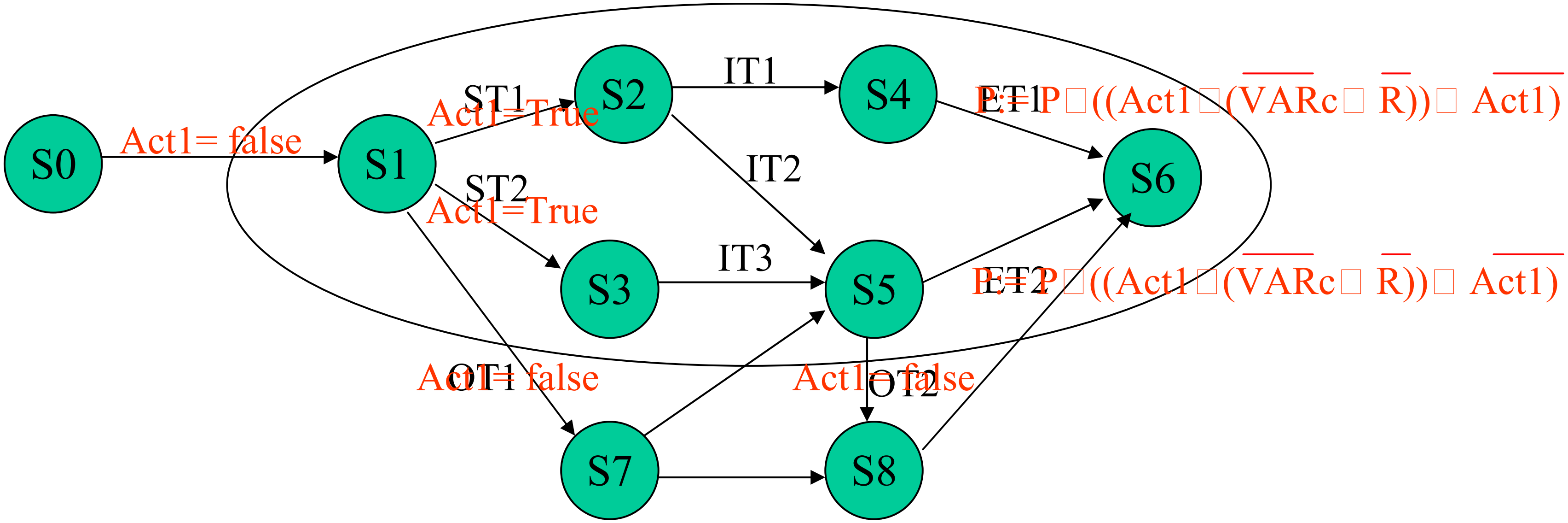
- Example of 1_transition activity
- Prohibition (S, R, T, _ , C) where C is a variables context
  - The activity T exists in the functional specification
  - To restrain the predicate





IM : Prohibition

- Example of n_transition activity :
- Prohibition (S, R, Activity1, _ , C)


S0
Act1= false
S1
ST1
Act1=True
S2
IT1
S4
ET1
$P:=P□((Act1□(\overline{VARc}□ \overline{R}))□ \overline{Act1})$
S6
IT2
ST2
Act1=True
S3
IT3
S5
ET2
$P:=P□((Act1□(\overline{VARc}□ \overline{R}))□ \overline{Act1})$
OT1
Act1= false
OT2
Act1= false
S7
S8

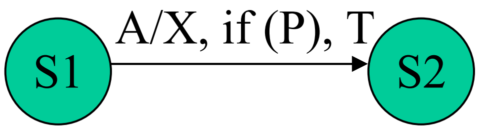
IM : Prohibition Algorithm

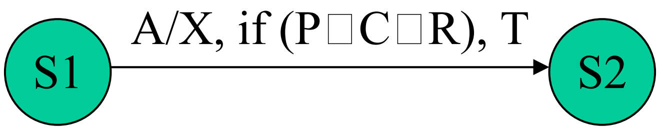
- **Require:** The permission with role R, variable context $VARc$ and activity i that maps the transition(s).
- **if** (1_Tr activity) **then**
- Revise the associated predicated to the transition: $P := P \square (\square VARc \square \square R)$
- (Note that if no predicate is associated to this transition, we create a new one $P := \square VARc \square\square R$)
- **end if**
- **if** (n_Tr activity) **then**
- Add the task *Acti* := *true*; to the *STS*.
- Add the task *Acti* := *false*; to the *OTS*
- Duplicate the *ETS* into *ETS*1 and *ETS*2
- Revise the associated predicated to the *ETS*1: $P := P \square Acti \square (\square VARc \square \square R)$
- Revise the associated predicated to the *ETS*2: $P := P \square (Acti = false)$
- Add the task *Acti* := *false*; to the *ETS*1.
- **end if**

- If many prohibitions : logical product

IM : Permission

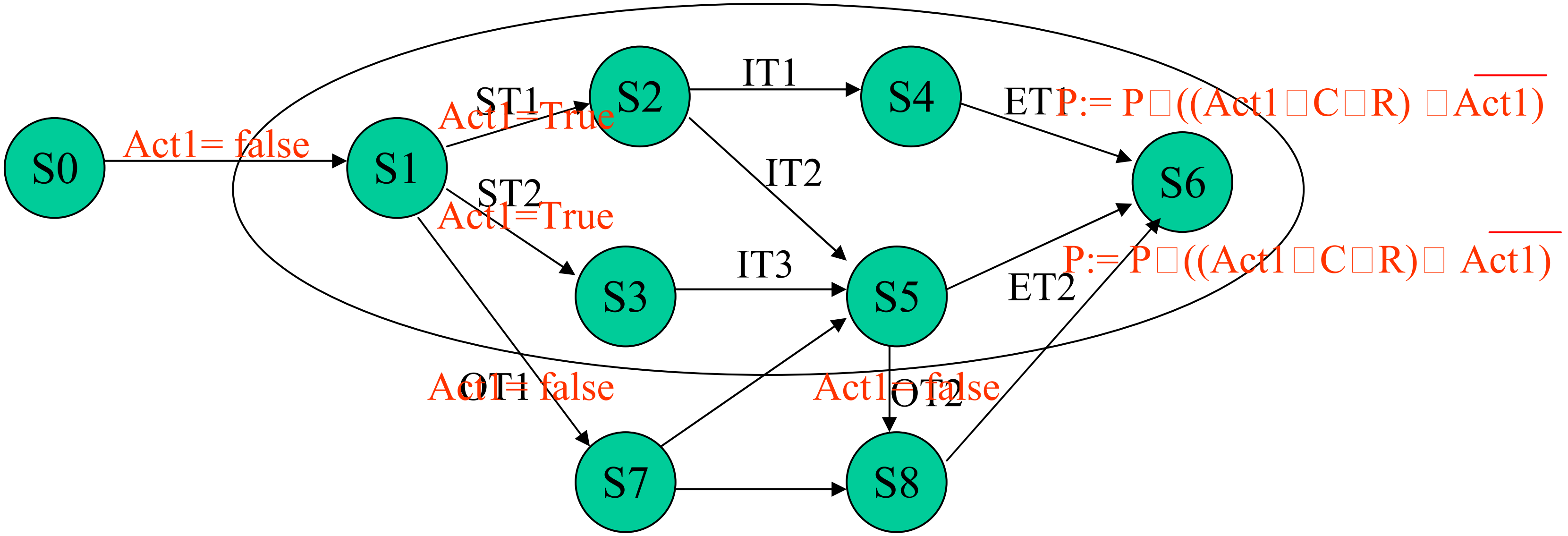
- Example of 1_transition activity :
- Permission (S, R, T, _ , C) where C a condition related to variables
  - The activity T exists in the functional specification
  - To restrain the predicate





IM : Permission

- Example of n_transition activity :
- Permission (S, R, Activity1, _ , C)



IM : Permission Algorithm

- **Require:** The permission with role R, variable context $V\ ARc$ and activity i that maps the transition(s).
- **if** (1_Tr activity) **then**
- Revise the associated predicated to the transition: $P := P$ □ ($V\ ARc$ □ R)
- (Note that if no predicate is associated to this transition, we create a new one $P := V\ ARc$ □ R)
- **end if**
- **if** (n_Tr activity) **then**
- Add the task $Acti := true$; to the STS.
- Add the task $Acti := false$; to the OTS
- Duplicate the ETS into $ETS1$ and $ETS2$
- Revise the associated predicated to the $ETS1$: $P := P$ □ $Acti$ □ ($V\ ARc$ □ R)
- Revise the associated predicated to the $ETS2$: $P := P$ □ ($Acti = false$)
- Add the task $Acti := false$; to the $ETS1$.
- **end if**

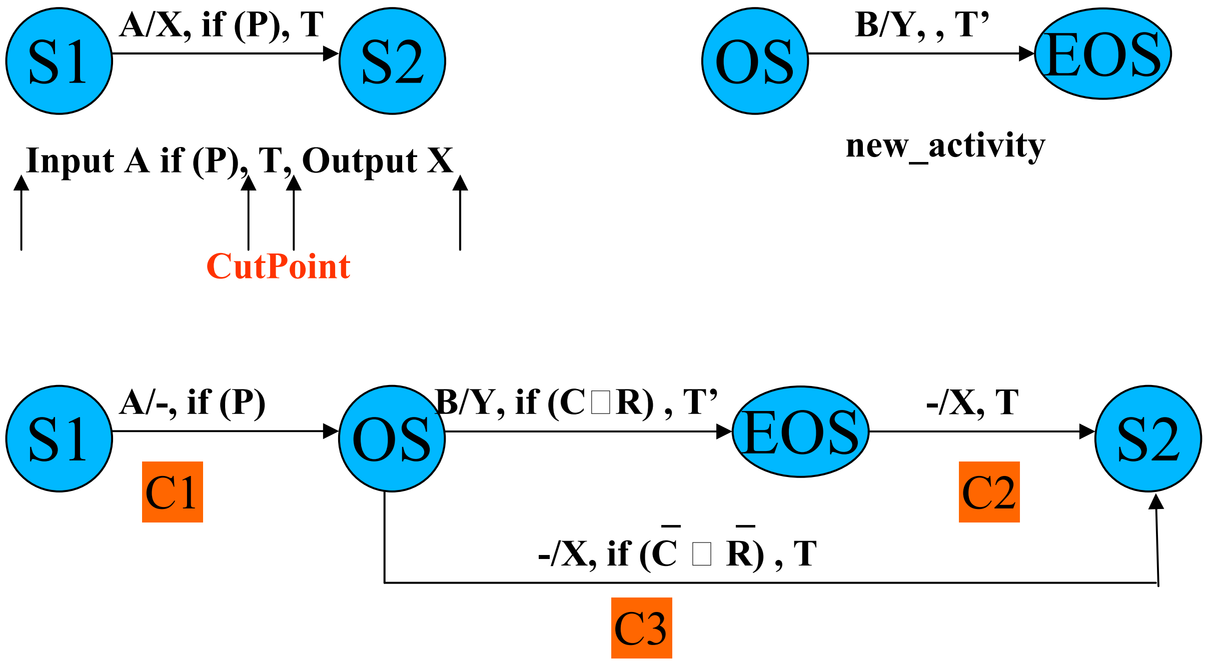
- If many permissions : logical sum

IM : Obligation (1/2)

- Example : Obligation (S, R, new_activity, _, (Input = A) and C)
  - Assumption : new_activity is a new activity
  - New_activity can be formally described using a partial EFSM (OS → EOS)
  - To determine the Cut Point
  - To add the activity and to connect transitions

IM : Obligation (2/2)

- Example : Obligation (S, R, new_activity, _, (Input = A) and C)

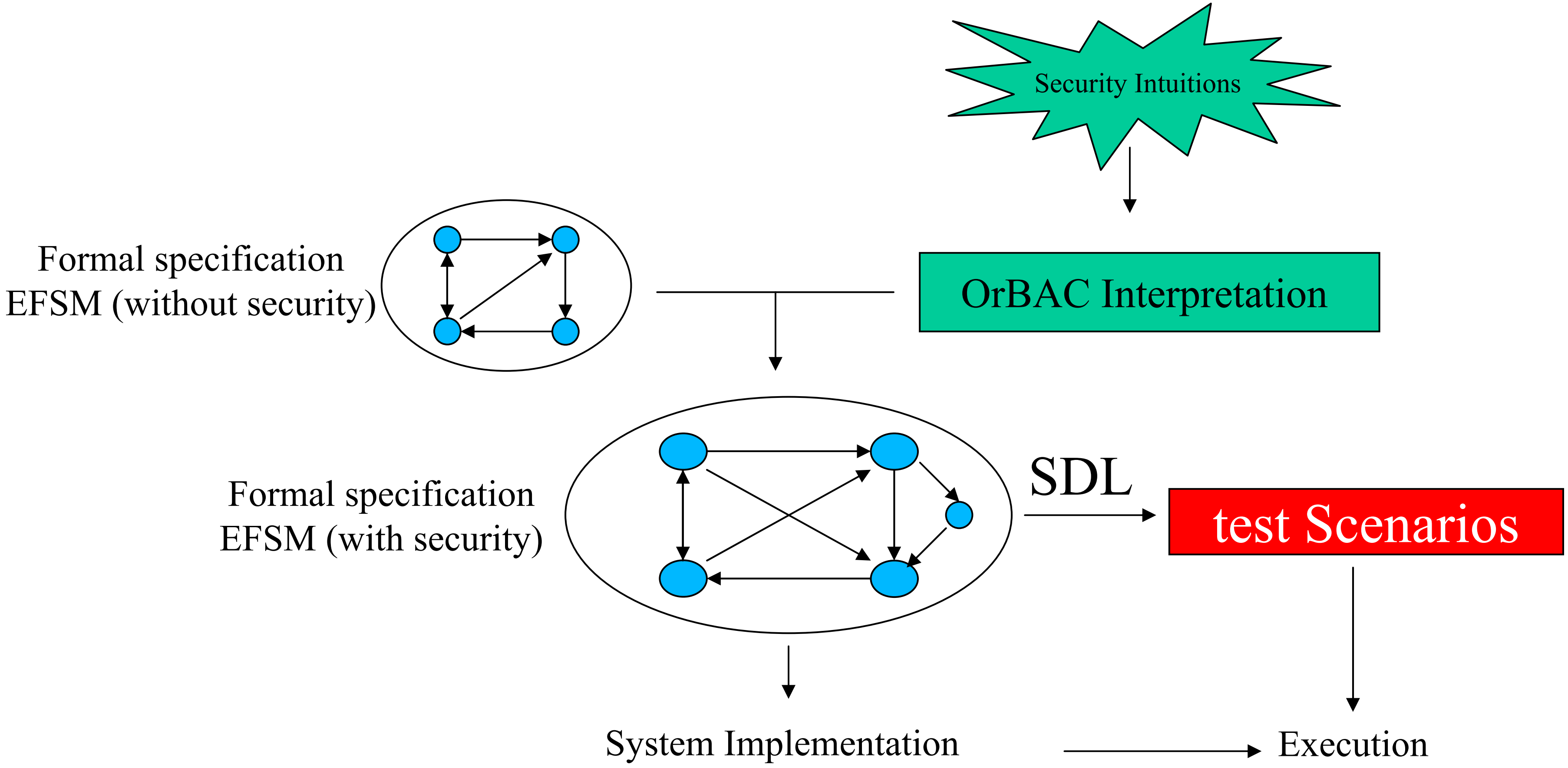


MI : Algorithme Obligation

- Input : EFSM M , Obligation and new activity

1. To restrain all transitions from OS with (role and ‘variables context’)
2. For each transition that maps the ‘EFSM context’, identify the Cut Point
3. Create transitions C1, C2 et C3

The main idea



Security Intuitions

Formal specification
EFSM (without security)

OrBAC Interpretation

Formal specification
EFSM (with security)

SDL

test Scenarios

System Implementation

Execution

Testing methodology

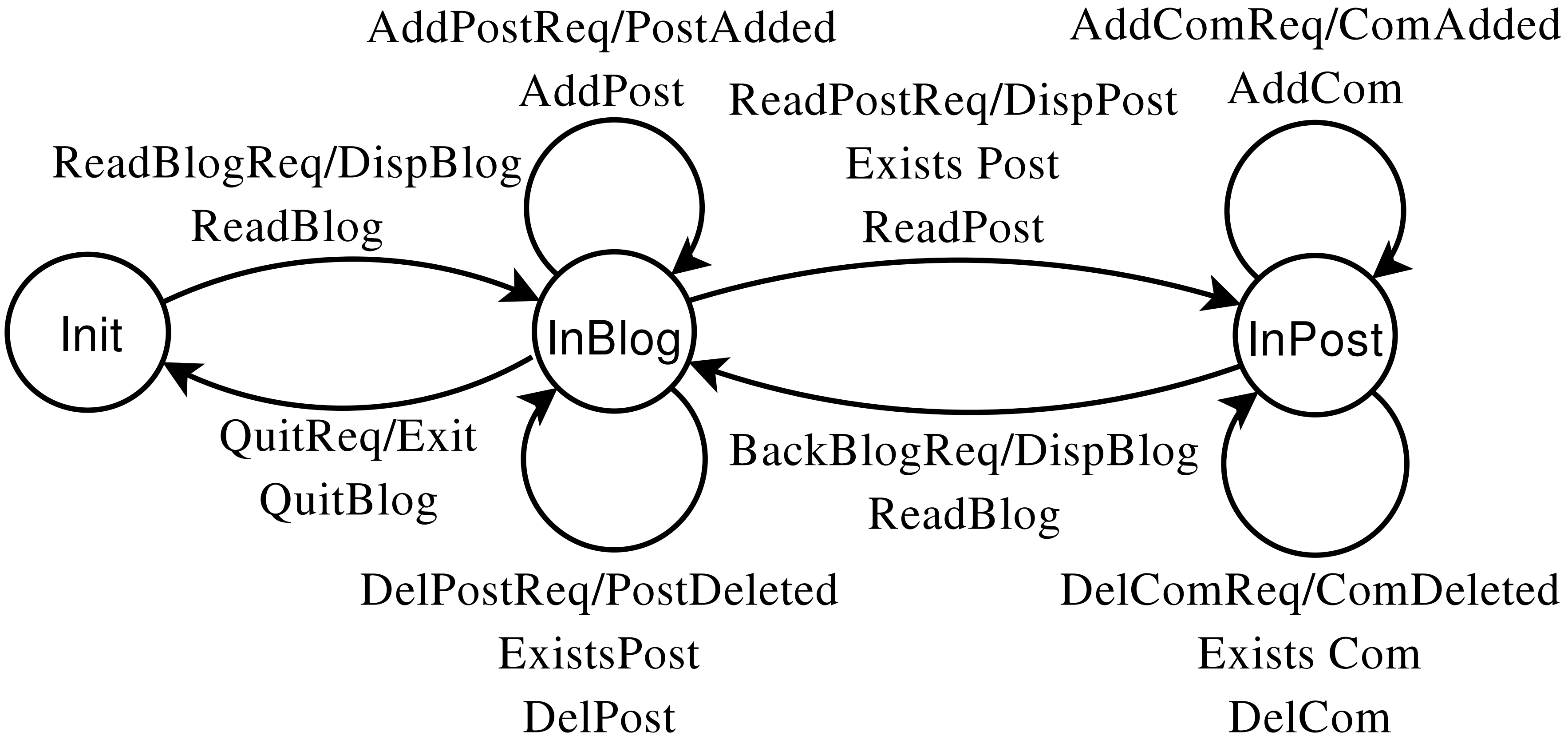
- A methodology based on the ISO9646 standard
    - Description of the system behavior using a formal language : SDL (ObjectGEODE)
    - Characterization of test objectives and test generations (security oriented objectives) (SIRIUS)
    - Definition of testing architecture
    - Execution

Case study : Weblog

Definition :

- A weblog is a website where entries are written in chronological order and displayed in reverse chronological order.
- Blogs provide commentary or news on a particular subject such as food, politics, or local news; some function as more personal online diaries. The ability for readers to leave comments in an interactive format is an important part of many blogs. (Wikipedia)

Weblog : formal specification



Weblog : SDL

State

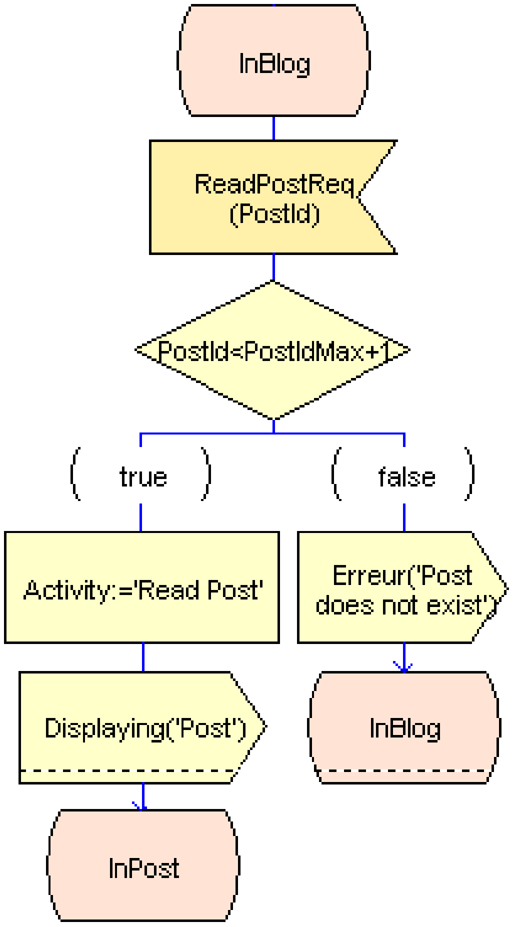
Input

Predicate

Task

Output

State



Specification Verification

- Model Checking
- Exhaustive simulation
- Absence of deadlocks and livelocks …
- Guided simulation

Security policy definition

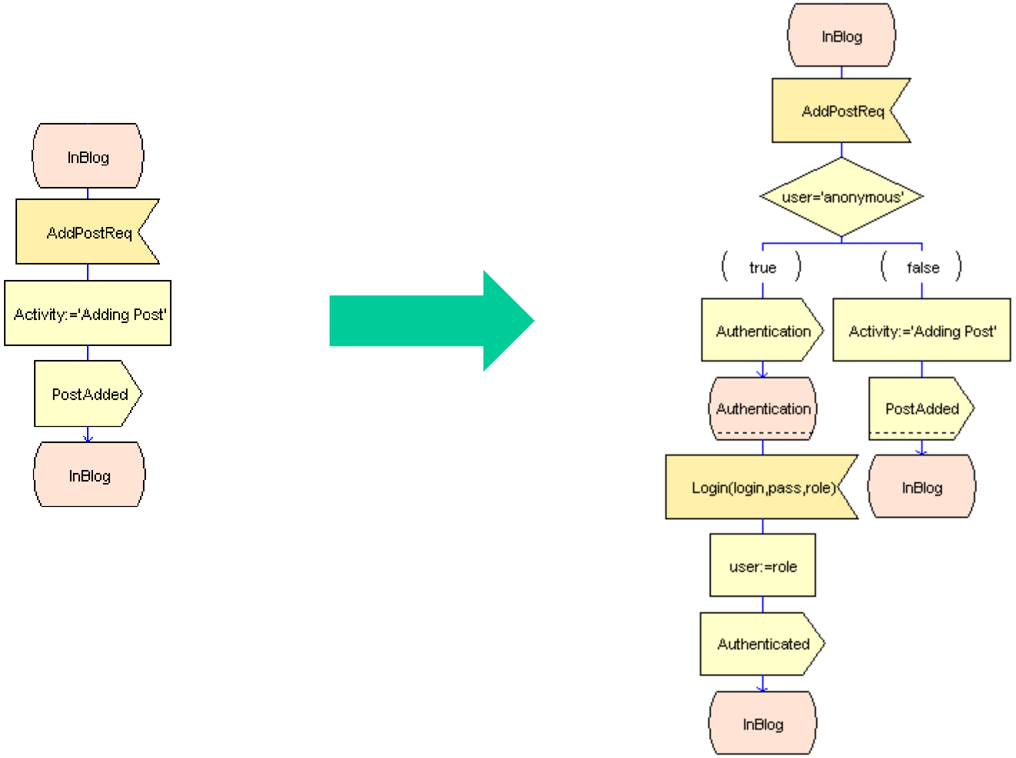
- 3 possibles roles : administrator, blogger and visitor
- An administrator can do any thing
- A blogger can only read and write but not delete
- A visitor can only read
- To write or delete, the user has to be authenticated

Security rules in OrBAC

- Obligation (Website, visitor, Authentication, _ , input = AddPostReq)
- Permission (Website, admin, ‘Deleting Comment’, Comment, _)
- Prohibition (Website, visitor, ‘Adding Comment’, Comment, _)
- …

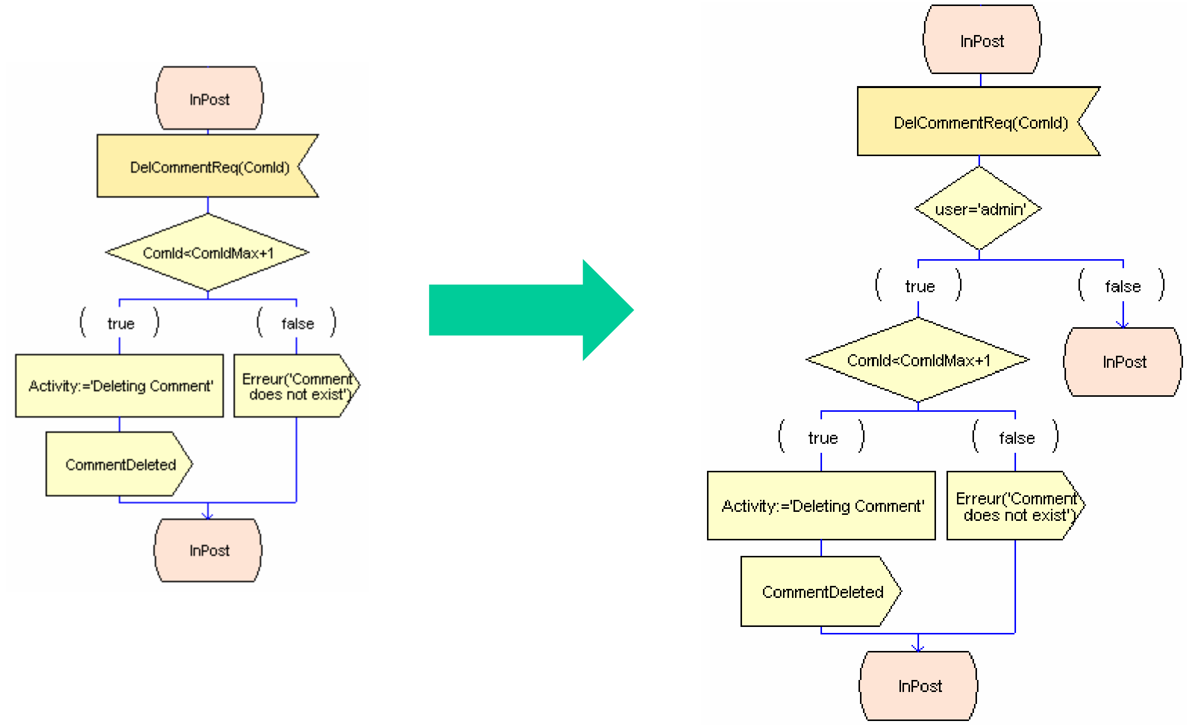
Rules integration (1/3)

- Obligation (Website, visitor, Authentication, _ , input = AddPostReq)



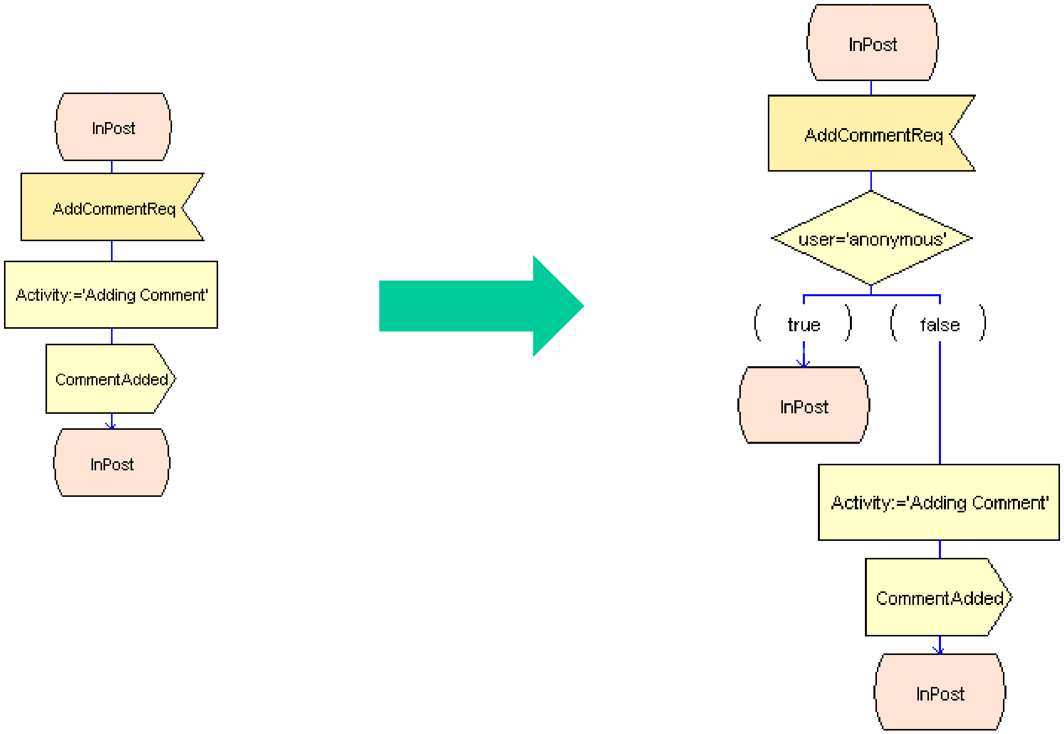
Rules integration (2/3)

- Permission (Website, admin, ‘Deleting Comment’, Comment, _)



Rules integration (3/3)

- Prohibition (Website, anonymous, 'Adding Comment', Comment, _)



Specifications: Before/After

	States	Transitions	Signals	Lines
Before	3	15	15	350
After	4	23	18	594

Test objectives determination

- Written in SDL
- Combinative choices
- Ex : An administrator tries to add a content, the activity is permitted and the content is added.
- 17 test objectives that represents 95% of the specification transitions.

Generation of test scenarios

- Using SIRIUS test generation tool
- A tool based on Hit-or-Jump algorithm that allows to avoid combinative explosion
- BFS (Breath First Search)
- Quick generation (3s) and short scenarios (7 transitions)
- Test scenarios can be provided in TTCN or MSC standard. => Portability

Outline

- Introduction
- Active/Passive Testing
- Active Testing Approach
  - Preliminaries
  - An integration based approach
  - The integration methodology
  - Use case : a Weblog
- Passive Testing Approach
  - Ongoing Work
- Conclusion

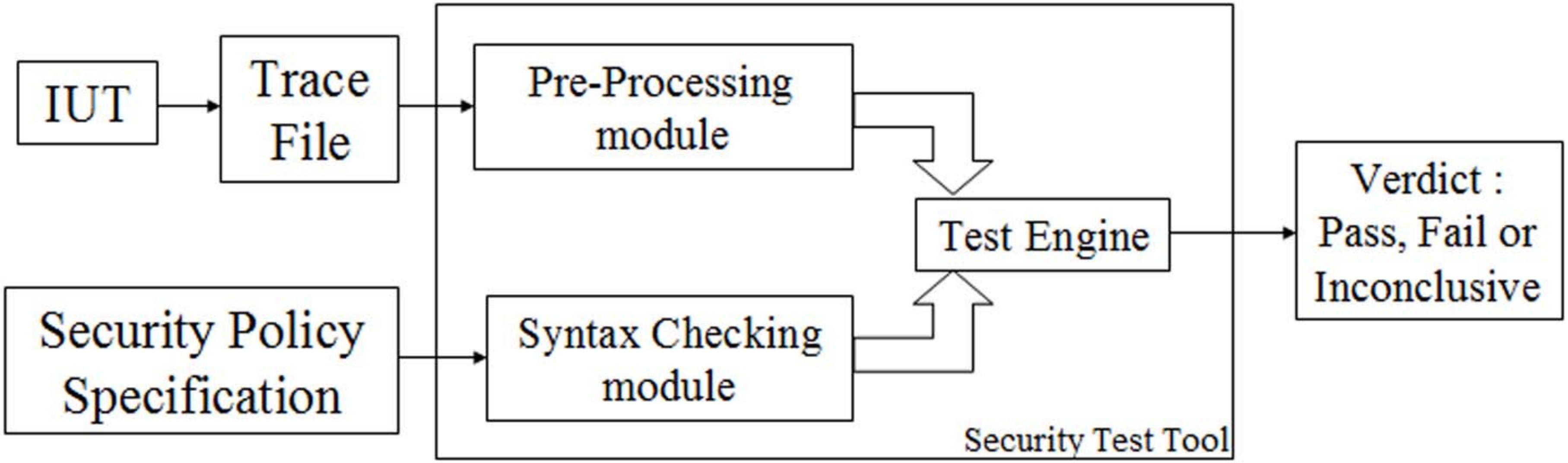
Our Aim

- Definition of Passive test techniques for security checking
- Detection of violations of security policies

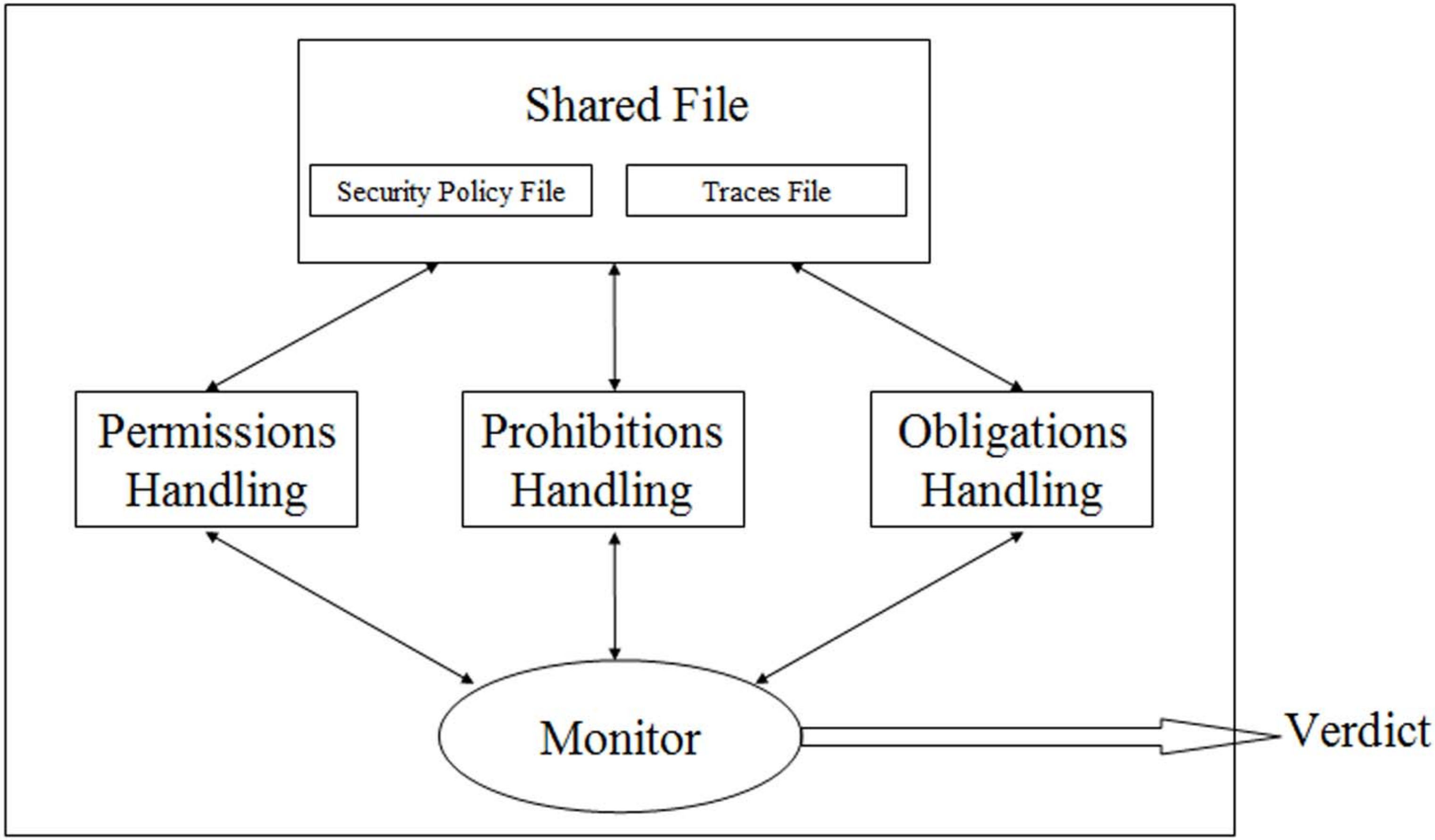
Security Rules Specification

- A formalism well adapted passive testing
- Syntax inspired by Nomad (Non atomic actions and deadlines)
- Specification of permissions, prohibitions and obligations concerning non atomic actions using a combination of deontic and temporal logics

Passive Testing Methodology

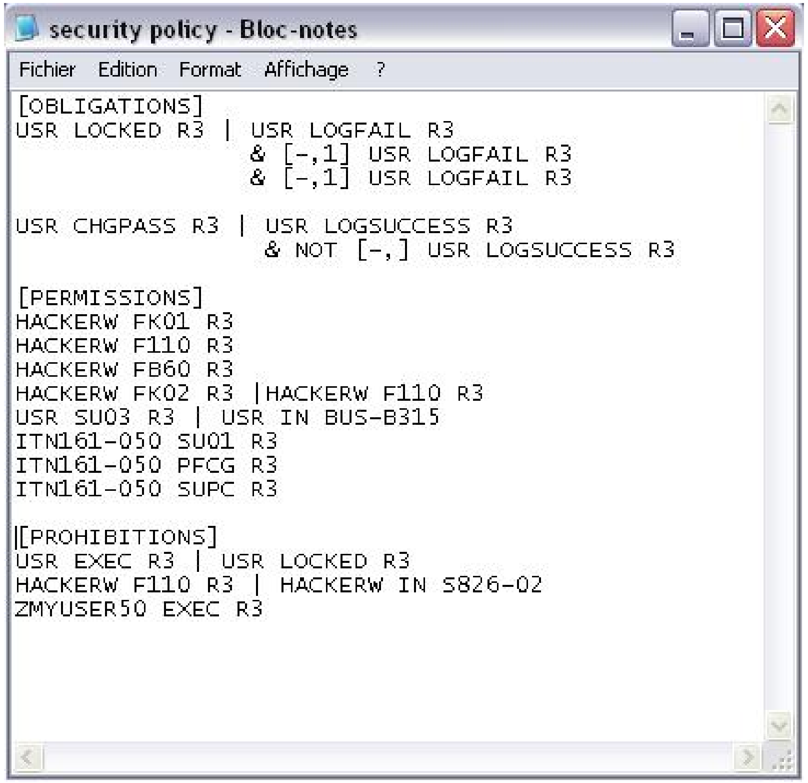


Test Engine


Shared File
Security Policy File
Traces File
Permissions Handling
Prohibitions Handling
Obligations Handling
Monitor
Verdict

SAP Case Study

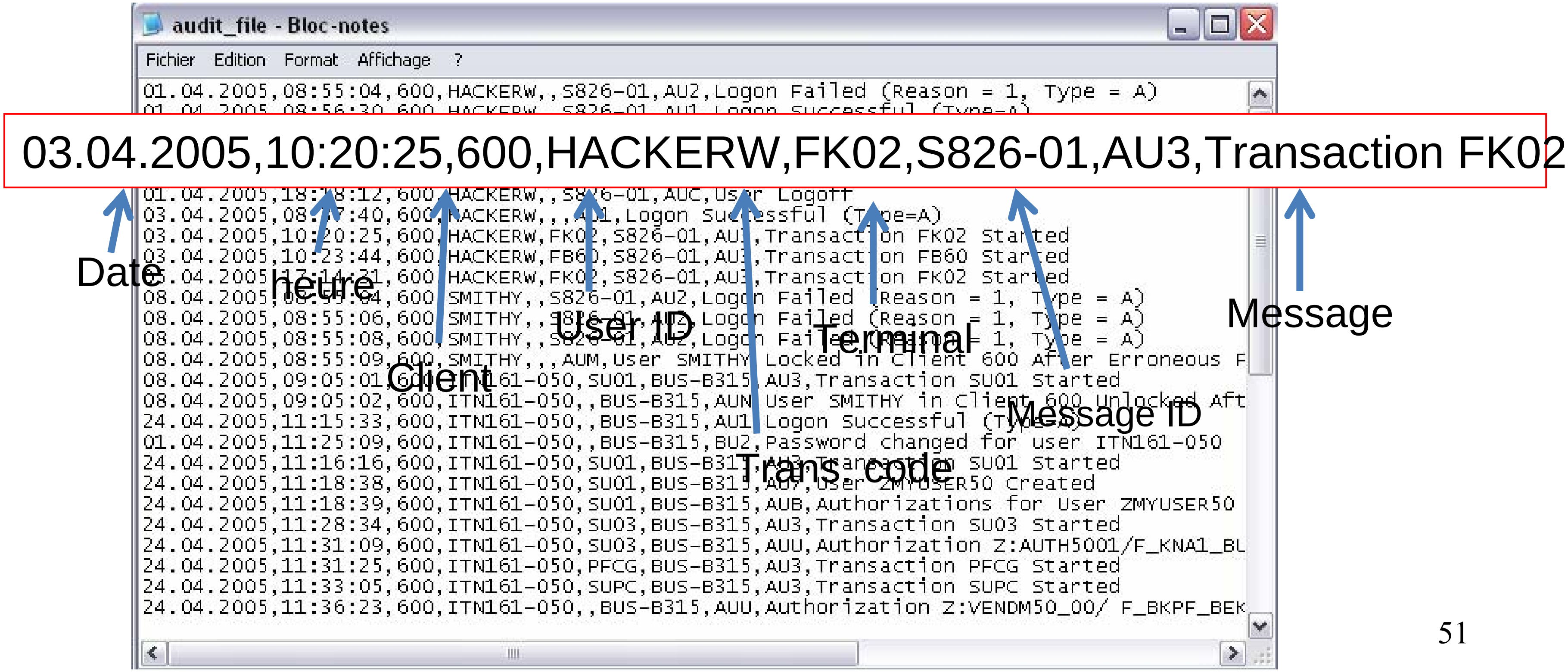
- 13 rules have been selected to be specified in our formalism
  - 2 Obligations
  - 3 Prohibitions
  - 8 Permissions



```
[OBLIGATIONS]
USR LOCKED R3 | USR LOGFAIL R3
                & [-,1] USR LOGFAIL R3
                & [-,1] USR LOGFAIL R3

USR CHGPASS R3 | USR LOGSUCCESS R3
                 & NOT [-,] USR LOGSUCCESS R3

[PERMISSIONS]
HACKERW FK01 R3
HACKERW F110 R3
HACKERW FB60 R3
HACKERW FK02 R3 |HACKERW F110 R3
USR SU03 R3 | USR IN BUS-B315
ITN161-050 SU01 R3
ITN161-050 PFCG R3
ITN161-050 SUPC R3

[PROHIBITIONS]
USR EXEC R3 | USR LOCKED R3
HACKERW F110 R3 | HACKERW IN S826-02
ZMYUSER50 EXEC R3
```

Results

- Trace file of the Audit application (25000 lines)

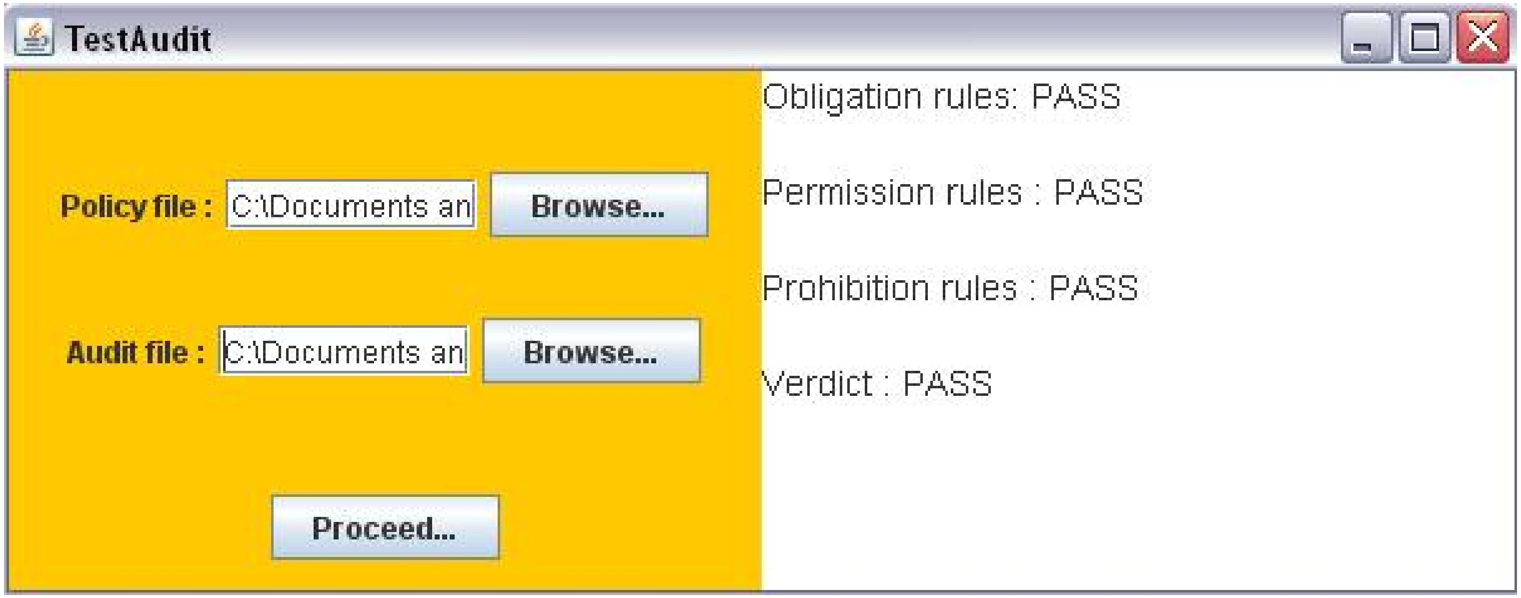


```
01.04.2005,08:55:04,600,HACKERW,,S826-01,AU2,Logon Failed (Reason = 1, Type = A)
01.04.2005,08:56:30,600,HACKERW,,S826-01,AU1,Logon Successful (Type=A)
01.04.2005,18:18:12,600,HACKERW,,S826-01,AUC,User Logoff
03.04.2005,08:17:40,600,HACKERW,,,AU1,Logon Successful (Type=A)
03.04.2005,10:20:25,600,HACKERW,FK02,S826-01,AU3,Transaction FK02 Started
03.04.2005,10:23:44,600,HACKERW,FB60,S826-01,AU3,Transaction FB60 Started
05.04.2005,17:14:31,600,HACKERW,FK02,S826-01,AU3,Transaction FK02 Started
08.04.2005,08:55:04,600,SMITHY,,S826-01,AU2,Logon Failed (Reason = 1, Type = A)
08.04.2005,08:55:06,600,SMITHY,,S826-01,AU2,Logon Failed (Reason = 1, Type = A)
08.04.2005,08:55:08,600,SMITHY,,S826-01,AU2,Logon Failed (Reason = 1, Type = A)
08.04.2005,08:55:09,600,SMITHY,,,AUM,User SMITHY Locked in Client 600 After Erroneous F
08.04.2005,09:05:01,600,ITN161-050,SU01,BUS-B315,AU3,Transaction SU01 Started
08.04.2005,09:05:02,600,ITN161-050,,BUS-B315,AUN,User SMITHY in Client 600 Unlocked Aft
24.04.2005,11:15:33,600,ITN161-050,,BUS-B315,AU1,Logon Successful (Type=A)
01.04.2005,11:25:09,600,ITN161-050,,BUS-B315,BU2,Password changed for user ITN161-050
24.04.2005,11:16:16,600,ITN161-050,SU01,BUS-B315,AU3,Transaction SU01 Started
24.04.2005,11:18:38,600,ITN161-050,SU01,BUS-B315,AU7,User ZMYUSER50 Created
24.04.2005,11:18:39,600,ITN161-050,SU01,BUS-B315,AUB,Authorizations for User ZMYUSER50
24.04.2005,11:28:34,600,ITN161-050,SU03,BUS-B315,AU3,Transaction SU03 Started
24.04.2005,11:31:09,600,ITN161-050,SU03,BUS-B315,AUU,Authorization Z:AUTH5001/F_KNA1_BL
24.04.2005,11:31:25,600,ITN161-050,PFCG,BUS-B315,AU3,Transaction PFCG Started
24.04.2005,11:33:05,600,ITN161-050,SUPC,BUS-B315,AU3,Transaction SUPC Started
24.04.2005,11:36:23,600,ITN161-050,,BUS-B315,AUU,Authorization Z:VENDM50_00/ F_BKPF_BEK
```

```
03.04.2005,10:20:25,600,HACKERW,FK02,S826-01,AU3,Transaction FK02 S
```

Results

- The system checks its security policy



Obligation rules: PASS

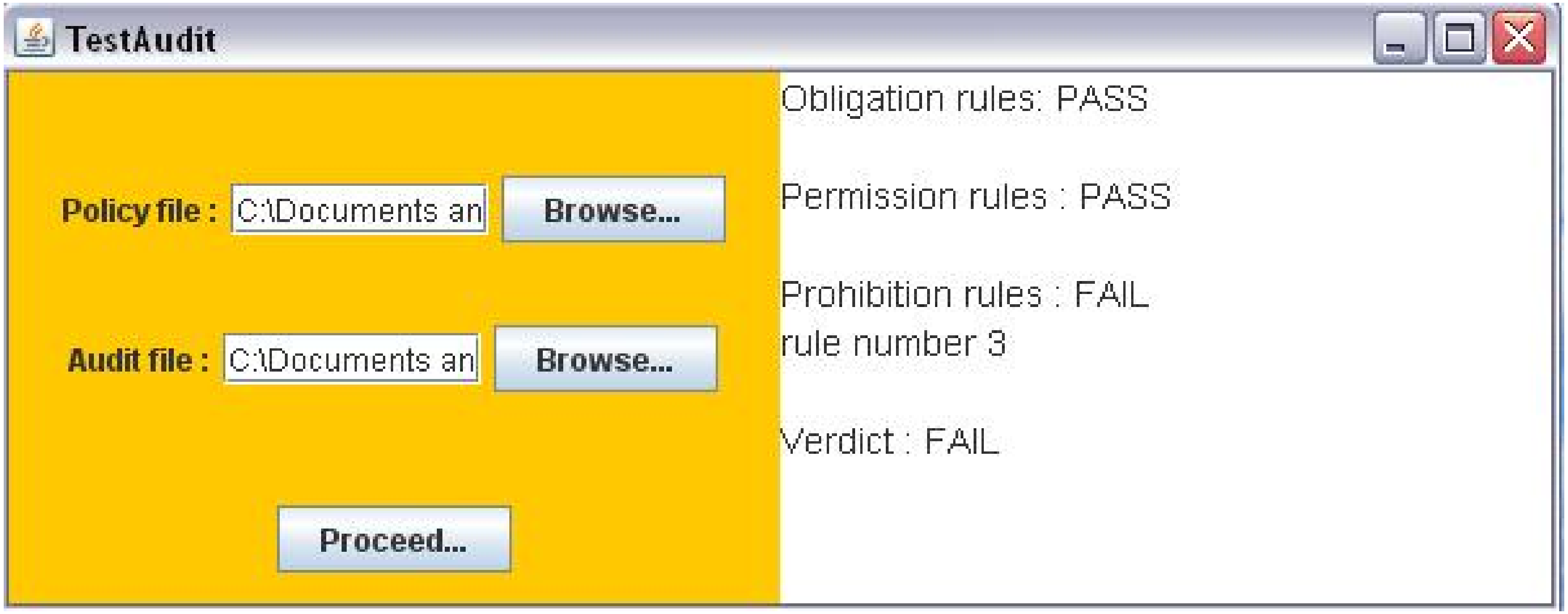
Permission rules : PASS

Prohibition rules : PASS

Verdict : PASS

Results

- Modifications in the Audit File



Obligation rules: PASS

Permission rules : PASS

Prohibition rules : FAIL
rule number 3

Verdict : FAIL

Conclusion and future work

- The security testing is still complex
- Automatic test generation for access control security rules (permission, prohibition, and obligation)
- Handling decomposable activities
- 3 algorithms
- Weblog and “A Travel Agency” case studies
- Passive testing (ongoing work)

Questions ?