Refinement and security
Ongoing works

Dominique Méry and the DESIRS project
Université Henri Poincaré Nancy 1 and LORIA

IFIP WG1.3 Meeting

Swansea, September 2005

Starting point

- Integration of security properties into action systems
- Security properties: permission, interdiction, obligation
- Links with deontic logic
- Integration of state-based approach and IA-based approach
- Access control and Flow control

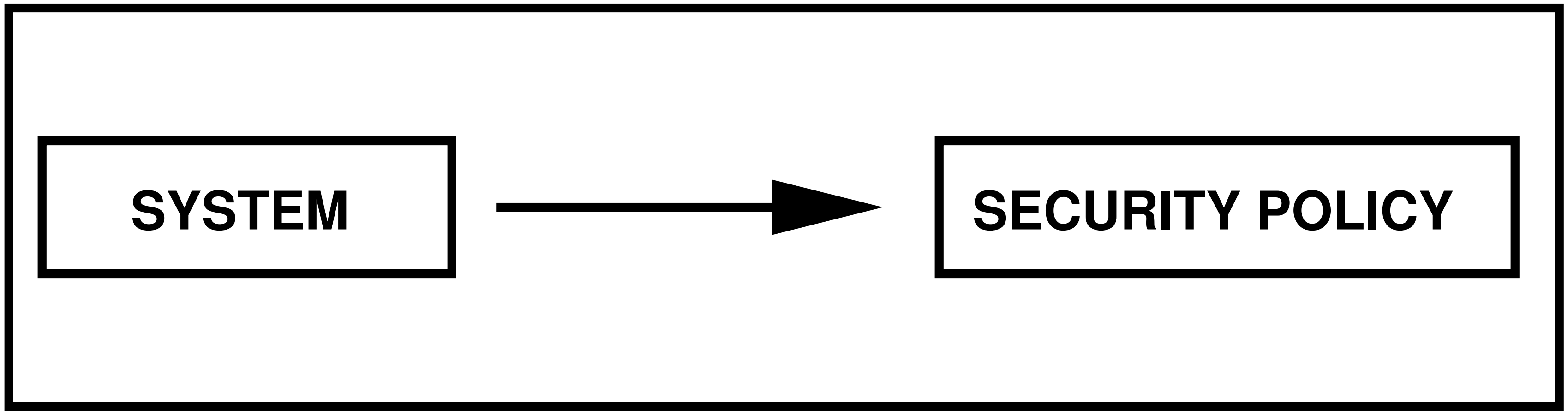
Case studies

- Control of transactions in a bank
- Management of patients records in a hospital (OrBaC model)
- Access control in a general settlement

Three points on the topics

- Relating ORBAC models to B models
- Extending events models for permissions, interdiction and obligation
- Management of permissions/interdictions

Construct a system wrt to (security) requirements



☼ **security policy: permissions, interdictions, obligations**

☼ **system: actions, states, variables, . . .**

Refine a system wrt to (security) requirements

- ☼ **security policy: permissions, interdictions, obligations**
- ☼ **system: actions, states, variables, . . .**
- ☼ **system may have different levels of abstractions and some parts may exist**
- ☼ **generation of the control part**

$$\textbf{system\_part}, \textbf{control\_part} \longrightarrow^{+} \textbf{SYSTEM} \longrightarrow \textbf{security policy} \tag{1}$$

Resetting the problem with respect to modelling languages

- ☼ **Expressing the security policy with OrBaC (Organisation-based Access Control Model)**
- ☼ **ORBAC model identify roles, activities, contexts . . .**
- ☼ **ORBAC model assigns permanent permissions**
- ☼ **Defining the system in event B**
- ☼ **Establishing the (proved) link between the security policy and the system**
- ☼ **. . . without obligation**
- ☼ **Next case study.....**

Problem of the accounting office for a social security company

- When a patient sends a request, the record is first processed by an administrative agent, then validated by the head of the service and finally a check is written by the accounter for the patient.

  1. Processing of the record by the administrative agent.
  2. Validation by the head of the service.
  3. Emission of a check by the accounter.

- Workflow problem

Problem of the accounting office for a social security company

Rules for accessing files of patients

- Files of the personnel can be accessed by the head of office and the accounter.
- Files of accounting can be accessed by the head of office and the accounter but can only modify by the accounter.
- Separation of duties: processing and writing a check can not be done by the same person

Problem of the accounting office for a social security company

Rules for accessing files of patients

- **The accounter should update files of accounting, after the writing of the check (obligation).**
- **At the end of the validation, a record can be rejected and no check is written.**

ORBAC modelling

```
ROLES={agt_admin,ch_serv,account}
ACTIONS={consult,traiter,validate,emit,modify}
VIEWS={f_patient,f_account,f_personnel,cheques}

OBJECTS={fm1,fm2,fm3,fp1,fp2,fc,cheque}
SUBJECTS={emp1,emp2,emp3,chef_service,accountable}
```

ORBAC modelling

```
use(f_patient,fm1)
use(f_patient,fm2)
use(f_patient,fm3)
use(f_patient,fm1)
use(f_personnel,fp1)
use(f_personnel,fp2)
use(f_account,fc)
use(cheques,cheque)

empower(agt_admin,emp1)
empower(agt_admin,emp2)
empower(agt_admin,emp3)
empower(ch_serv,chef_service)
empower(account,accountable)
```

ORBAC modelling

```
//  permissions
        permission(office,agt_admin,traiter,f_patient)
        permission(office,ch_serv,validate,f_patient)
        permission(office,ch_serv,consult,f_personnel)
        permission(office,ch_serv,consult,f_account)
        permission(office,account,emit,cheques)
        permission(office,account,consult,f_account)
        permission(office,account,modify,f_account)
//  hierarchy of  roles :
        specialized_role(account,agt_admin)
        senior_role(ch_serv,agt_admin)
```

Checking the ORBAC model

- Checking the absence of inconsistency
- Eliminating inconsistencies by automatic checking using a PROLOG-like expression
- We do not care of this phase and we assume that the ORBAC model is sound.
- Applications of fusion techniques . . .

Events System Models

An **event system model** is made of

- State **constants** and state **variables** constrained by a state **invariant**

- A finite set of **events**

Proofs ensures the consistency between the invariant and the events

An event system model can be **refined**

Proofs must ensure the correctness of refinement

B models

```
MODEL
    m
SETS
    s
CONSTANTS
    c
PROPERTIES
    P(s, c)
VARIABLES
    x
INVARIANT
    I(x)
ASSERTIONS
    A(x)
INITIALISATION
    <substitution>
EVENTS
    <list of events>
END
```

- ☑ **A model has a name m**
- ☑ **the clause SETS, CONSTANTS and the clause PROPERTIES introduce information related to the mathematical structure of the problem to solve**
- ☑ **The invariant $I(x)$ types the variable x, which is assumed to be initialized with respect to the initial conditions and which is preserved by events (or transitions) of the list of events.**

Meaning of the model

- ◇ s, c and $P(s,c)$ define the mathematical structure of the problem: $\Gamma(s,c)$.
- ◇ Each computation starts by a state satisfying $Init(x)$.
- ◇ The list of possible events is $\{e_1, \ldots, e_n\}$ and any event e is characterized by a binary relation $BA(e)(x,x')$ over possible values of x.
- ◇ For each event e, there is a condition called a guard which is true, when the event is observed.

Events

Event : E	Before-After Predicate
BEGIN $x : P(x_0, x)$ **END**	$P(x, x')$
SELECT $G(x)$ **THEN** $x : P(x_0, x)$ **END**	$G(x) \quad \wedge \quad P(x, x')$
ANY t **WHERE** $G(t, x)$ **THEN** $x : P(x_0, x, t)$ **END**	$\exists t \cdot (\, G(t, x) \quad \wedge \quad P(x, x', t)\,$

Guards of event

Event : E	Guard: grd(E)
BEGIN S **END**	$TRUE$
SELECT $G(x)$ **THEN** T **END**	$G(x)$
ANY t **WHERE** $G(t,x)$ **THEN** T **END**	$\exists t \cdot G(t,x)$

MODEL abstract_model

```
SETS
ROLES={agt_admin,ch_serv,account};
ACTIONS={consult,traiter,validate,emit,modify};
VIEWS={f_patient,f_account,f_personnel,cheques}
CONSTANTS
permission
PROPERTIES
permission<:ROLES*ACTIONS*VIEWS    &
(agt_admin|->traiter|->f_patient):permission  &
(ch_serv|->validate|->f_patient):permission &
!(aa,vv).((aa:ACTIONS & vv:VIEWS & (agt_admin|->aa|->vv):permission)=>(ch_se
(ch_serv|->consult|->f_personnel):permission &
(ch_serv|->consult|->f_account):permission  &
(account|->emit|->cheques):permission &
!(aa,vv).((aa:ACTIONS & vv:VIEWS & (agt_admin|->aa|->vv):permission)=>(accou
(account|->consult|->f_account):permission &
(account|->modify|->f_account):permission &
! Perm. ((Perm <: ROLES*ACTIONS*VIEWS  &
(agt_admin|->treat|->f_patient):Perm &
(ch_serv|->validate|->f_patient):Perm &
!(aa,vv).((aa:ACTIONS & vv:VIEWS & (agt_admin|->aa|->vv):Perm)=>(ch_serv|->a
(account|->emite|->cheques):Perm &
!(aa,vv).((aa:ACTIONS & vv:VIEWS & (agt_admin|->aa|->vv):Perm)=>(account|->a
     )=> permission<:Perm)
```

MODEL abstract_model

```
VARIABLES
        history
INVARIANT
history<:ROLES*ACTIONS*VIEWS &
history<:permission  &
!(rr,vv).((rr:ROLES & vv:VIEWS &
          (rr|->validate|->vv):history)
           =>(#rr2.(rr2:ROLES &(rr2|->treat|->vv):history))) &
!(rr,vv).((rr:ROLES & vv:VIEWS &
         (rr|->emit|->vv):history)
           =>(#rr2.(rr2:ROLES &(rr2|->validate|->vv):history)))
```

event Action

```
Action =
        ANY rr,vv,aa WHERE
        rr:ROLES &
        vv:VIEWS &
        aa:ACTIONS &
        (rr|->aa|->vv):permission &
        aa/=traiter &   aa/=emit & aa/= validate
        THEN
        history:=history\/{(rr|->aa|->vv)}

        END;
```

event Treatment

```
Treatment=
        ANY rr,vv,aa WHERE
        rr:ROLES &
        vv:VIEWS &
        aa:ACTIONS &
        (rr|->aa|->vv):permission &
        aa=traiter
        THEN
        history:=history\/{(rr|->aa|->vv)}
        /* treatment of the record */
        END;
```

event Validation

```
Validation =
        ANY rr,vv,aa WHERE
        rr:ROLES &
        vv:VIEWS &
        aa:ACTIONS &
        (rr|->aa|->vv):permission &
        aa=validate &
        #rr2.(rr2:ROLES &(rr2|->traiter|->vv):history)
        THEN
        history:=history\/{(rr|->aa|->vv)}
        /* validation of record */
        END;
```

event Emission

```
Emission =
        ANY rr,vv,aa WHERE
        rr:ROLES &
        vv:VIEWS &
        aa:ACTIONS &
        (rr|->aa|->vv):permission &
        aa=emit &
        #rr2.(rr2:ROLES &(rr2|->validate|->vv):history)
        THEN
        history:=history\/{(rr|->aa|->vv)}
        /* emission of a check */
        END
```

First step: relating ORBAC models to B models

- **Constants and properties are defined from constants and properties of ORBAC model: ORBAC models are supposed to be consistent with respect to permissions and interdictions.**

- **Two B models are produced: an abstract one with roles . . . and a refinement with subjects (no problem of proof)**

- $history \subseteq permission$ **is the relation to maintain through further refinement steps.**

- **The concrete model** CM **refines the abstract one** AM **and both satisfy security properties (only permissions and interdictions).**

What to do with the concrete model?

- Expression of ORBAC requirements in the B world.
- Refinement can start from the concret model and go further (workflow properties, for instance)
- Replaying the game by instanciating constants for instance

Proof obligations for a model

	Proof obligation
(INV1)	$\Gamma(s,c) \ \vdash \ Init(x) \ \Rightarrow \ I(x)$
(INV2)	$\Gamma(s,c) \ \vdash \ I(x) \ \wedge \ BA(e)(x,x') \ \Rightarrow \ I(x')$
(DEAD)	$\Gamma(s,c) \ \vdash \ I(x) \ \Rightarrow \ (\mathsf{grd}(e_1) \ \vee \ \ldots \ \mathsf{grd}(e_n))$
(SAFE)	$\Gamma(s,c) \ \vdash \ I(x) \ \Rightarrow \ A(x)$
(FIS)	$\Gamma(s,c) \ \vdash \ I(x) \ \wedge \ \mathsf{grd}\,(E) \ \Rightarrow \ \exists x' \cdot P(x,x')$

Second: Extending the model

- **Extending the description of events by annotation of events with permissions and interdictions**

- **Adding new proof obligations for checking the resulting extended model**

	Proof obligation
(PERMISSION)	$\Gamma(s,c) \vdash I(x) \;\wedge\; PERM(e)(x) \Rightarrow G(e)(x)$
(INTERDICTION)	$\Gamma(s,c) \vdash I(x) \;\wedge\; INT(e)(x) \Rightarrow \neg G(e)(x)$
P/I	$\Gamma(s,c) \vdash I(x) \;\wedge\; INT(e)(x) \Rightarrow \neg PERM(e)(x)$

Third: void permissions

- **Constants become variables for the management of security policy: void permissions**
- $history \subseteq permission$ **can become now wrong....**
- **Example: Bob is a PhD student and has an access card for moving from a university unit to another one and, when the end of the academic time happens, he looses the access to the laboratory. However, he is in the laboratory when he looses his status!**
- $history \subseteq (permission \vee void\_permissions)$ **is now the good invariant.**

Summary on the three points

- **Relating ORBAC models to B models**
- **Extending events models for permissions, interdiction and obligation: question on refining permissions**
- **Management of void permissions**

Future works

- Mechanizing the translation from ORBAC models to B models
- General proved development of B models wrt to access control (case studies)
- Extending events models for permissions, interdiction and obligation: question on refining permissions
- Management of void permissions: relation to obligations, what to do with the renegats, squatters, ...
- Flow Control