The Generalized Temporal Role Based Access Control Model

Security Management

Lecture 7

March 21, 2006

Outline

- Introduction and Motivation
- Overview of the Generalized Temporal RBAC Model
- Expressiveness and Design Considerations
- Related Work
- Conclusion and Future Work

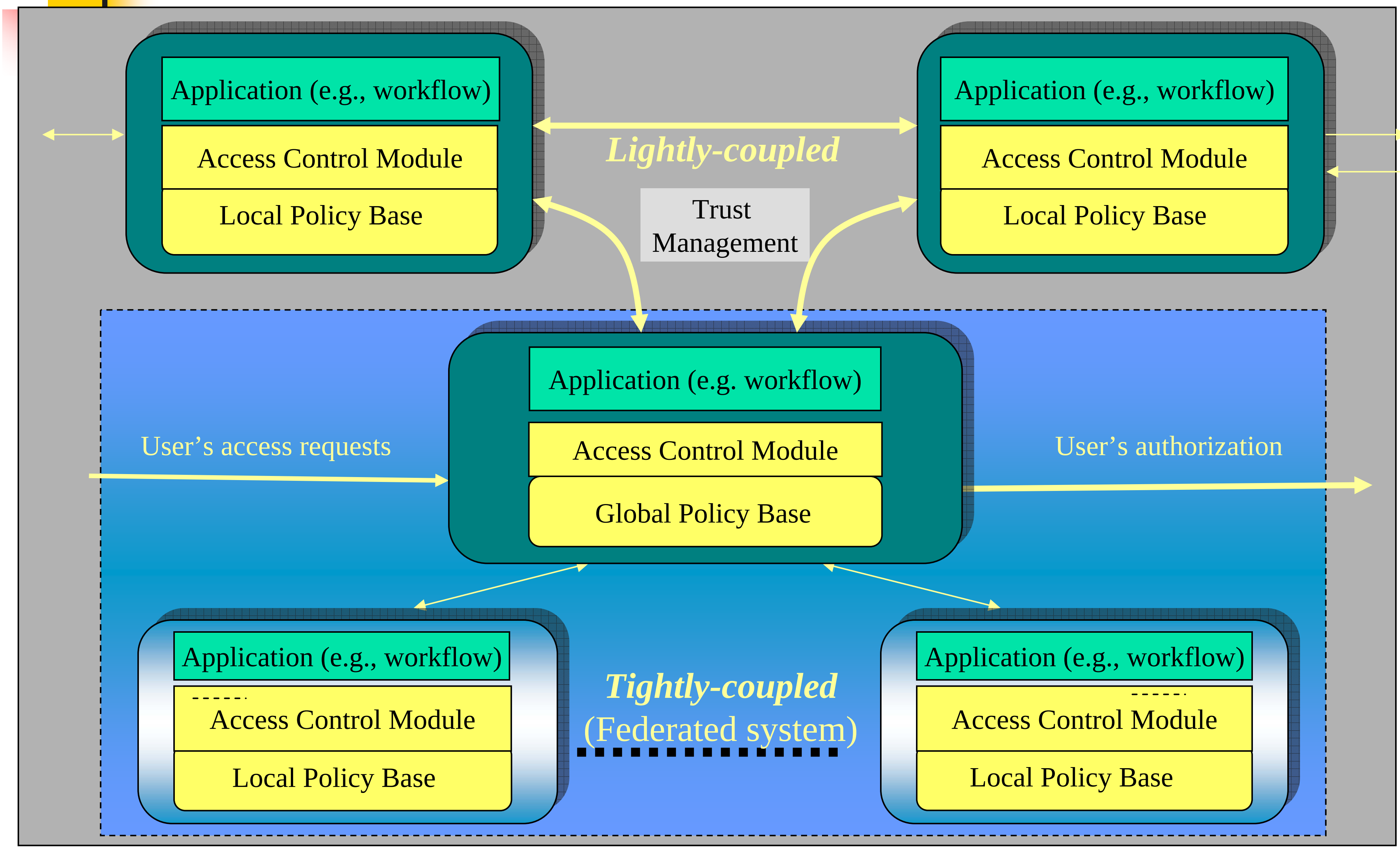
Research Motivation

- Insider attack is a major threat in organizational systems (CSI/FBI Survey)
- Traditional discretionary and mandatory access control (DAC & MAC) approaches have limitations
- Context-based access control is a critical need for emerging applications

Research Motivation

- “To realize the Department of Defense's vision for the Global Information Grid (GIG), information assurance (IA) requirements include robust identity, authentication and privilege management, policy for dynamic access control, security management, and 'persistence monitoring' or continual monitoring throughout the network, according to Daniel G. Wolf, the director of information assurance for the National Security Agency (NSA).”

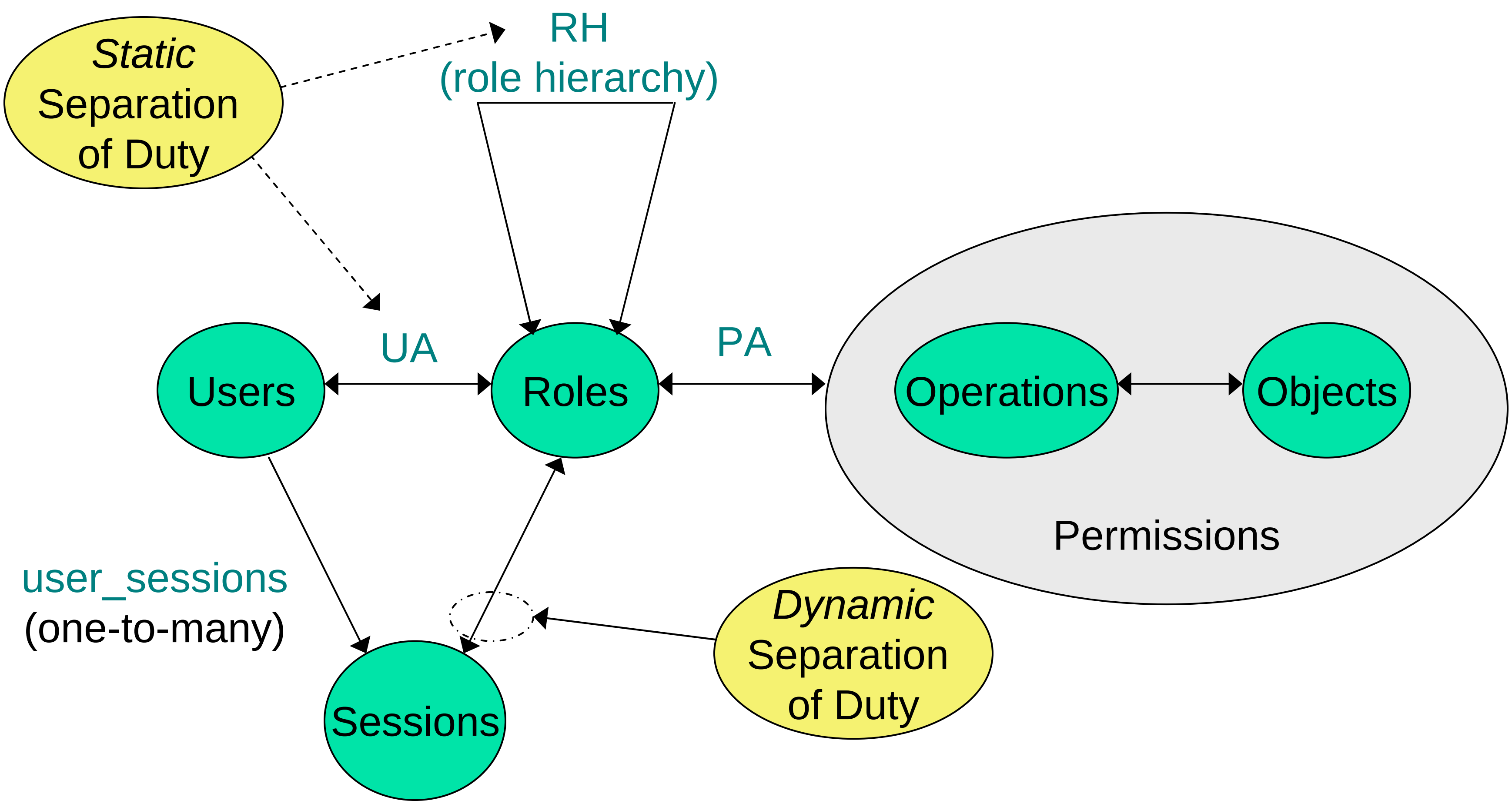
Security Management in Multidomain Environment



Role Based Access Control (RBAC)

- RBAC is a promising approach for addressing diverse security needs
- Access control in organizations is based on "roles that individual users take on as part of the organization"
- A role is "is a collection of permissions"
- Constraints are applied to all the links

NIST Constrained RBAC



Advantages of RBAC

- Allows efficient security management
- Supports principle of least privilege
- Separation of duty constraints
- Policy-neutral and provides generality
- Encompasses traditional discretionary and mandatory policies
- Suitable for use in multidomain environments such as digital government, multi-enterprise venture and international coalition

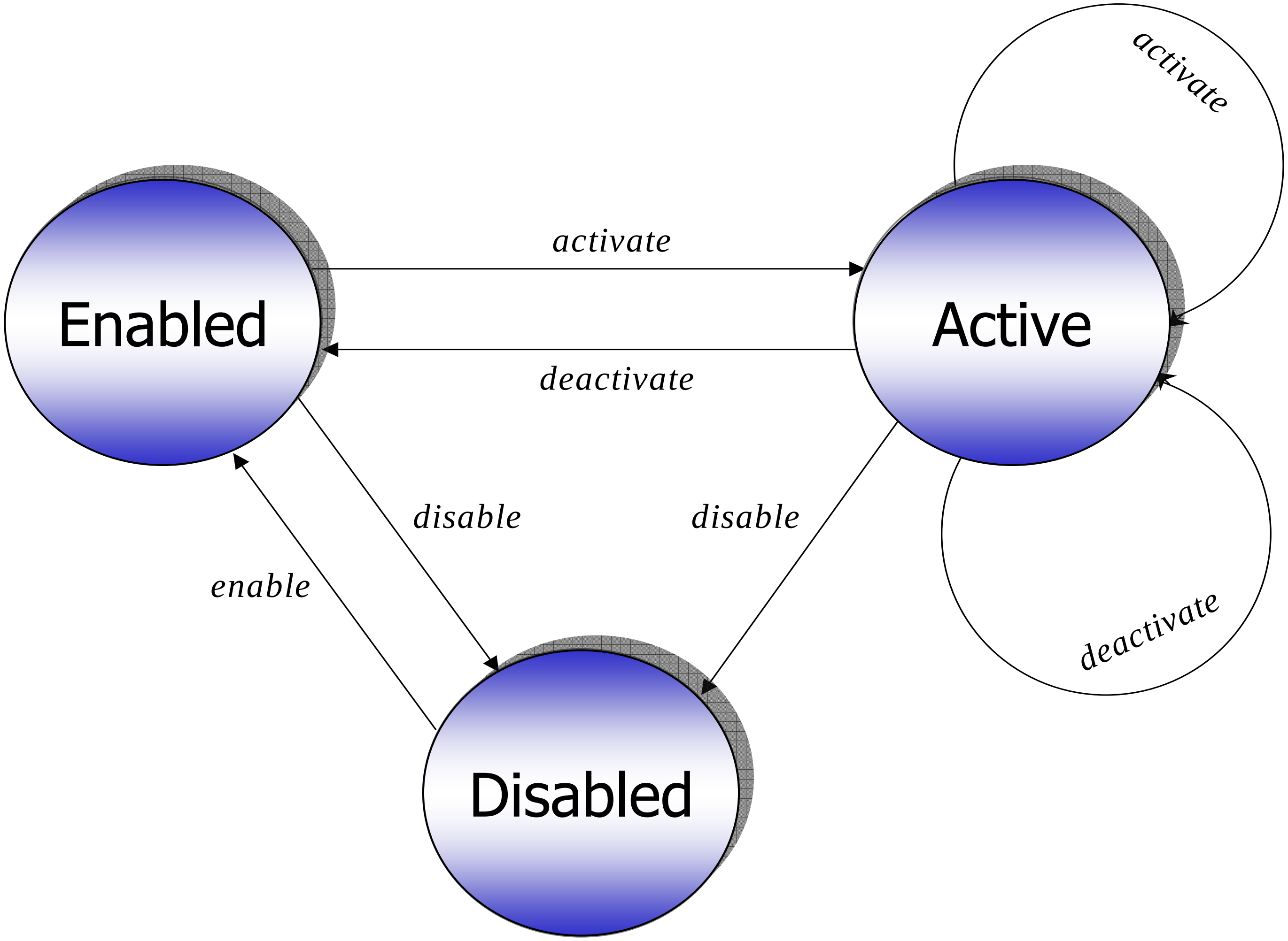
Time-based Access Control Requirement

- Organizational functions and services with temporal requirements
  - A part-time staff is authorized to work only between 9am-2pm on weekdays
  - A day doctor must be able to perform his/her duties between 8am-8pm
  - An external auditor needs access to organizational financial data for a period of three months
  - A video library allows access to a subscriber to view at most three movies every week
  - In an insurance company, an agent needs access to patient history until a claim has been settled

What to model in Generalized Temporal RBAC (GTRBAC)?

- Triggers and Events
- Temporal constraints
  - Roles, user-role and role-permission assignment constraints
  - Activation constraints (cardinality, active duration,..)
- Temporal role hierarchy
- Time-based Separation of duty constraints

States of a Role in GTRBAC



Event and Trigger

- Simple events
  - `enable` r — `disable` r
  - `assign`$_U$ r `to` u — `deassign`$_U$ r `to` u
  - `assign`$_P$ p `to` r — `deassign`$_P$ p to r
  - `activate` r for u — `deactivate` r for u
- Prioritized event *pr:E*, where $pr \in$ `Prios`
- Status
  - Role, assignment status – e.g.. `enabled(`r`)`; `p_assigned(`p, r`)`
- Triggers: $E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow pr{:}E$ `after` Δt, *where* E_i *are events,* C_i *are status expressions*

  Example:

  `enable` DayDoctor $\rightarrow$ `enable` DoctorInTraining after 1 hour
- User/administrator run-time request: *pr:E* `after` Δt

Temporal Constraints: Roles, User-role and Role-permission Assignments

- Periodic time
  - (I, P) : $\langle$`[begin, end]`, `P`$\rangle$ is a set of intervals
  - `P` is an infinite set of recurring intervals
- Calendars:
  - *Hours, Days, Weeks, Months, Years*
- Examples

  $all.Weeks + \{2, \ldots, 6\}.Days + 10.Hours \triangleright 12.hours$

  - Daytime (9am to 9pm) of working days

Temporal Constraints: Roles, User-role and Role-permission Assignments

- Periodicity: (I, P, *pr:E*)
  - ([1/1/2000, $\propto$], Daytime, `enable` DayDoctor)
  - ([1/1/2001, $\propto$], {Mon,Wed}, `assign`$_U$ DayDoctor `to` Smith)
- Duration constraint: (D, *pr:E*)
  - (Five hours, `enable` DoctorInTraining)
  - `activate` DayDoctor `for` Smith $\rightarrow$ `enable` DoctorInTraining after 1 hour

Activation Time Constraints

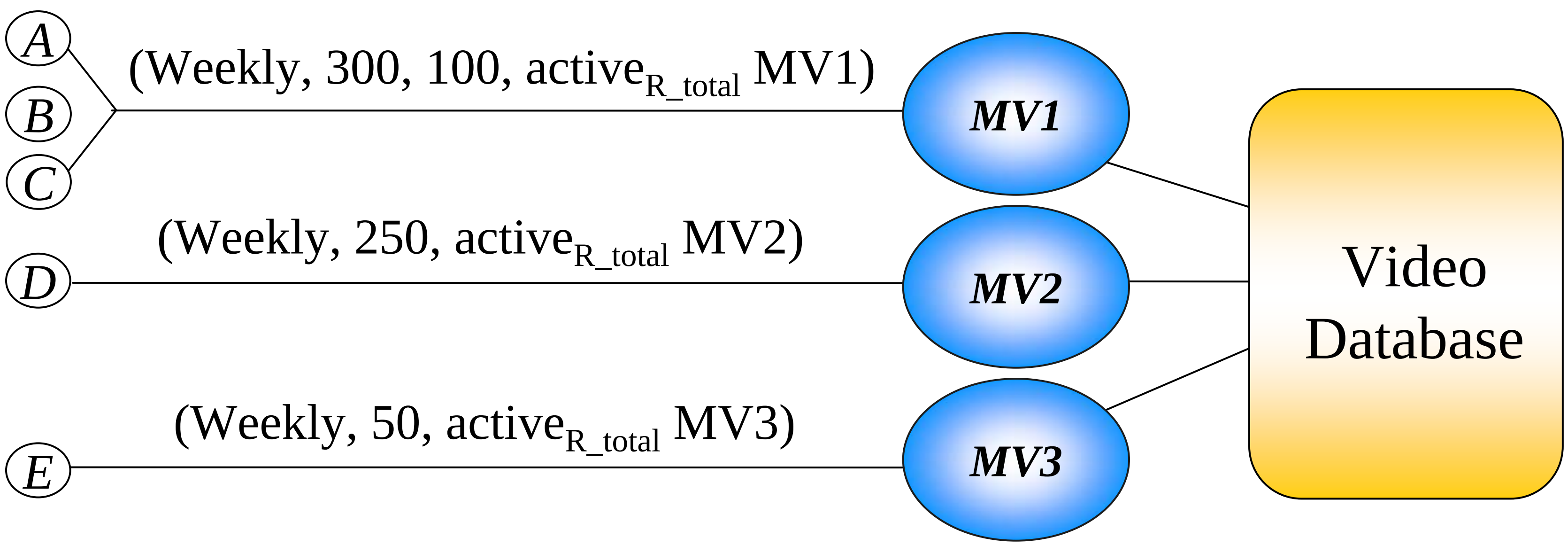
- Active role duration
  - Total duration for role activation
    1. Per role: D_{active}, $[D_{default}]$, $\texttt{active}_{\texttt{R\_total}}$ *r*
    2. Per user role: $D_{uactive}$, *u*, $\texttt{active}_{\texttt{UR\_total}}$ *r*
  - Max active role duration per activation
    1. Per role: D_{max}, $\texttt{active}_{\texttt{R\_max}}$ *r*
    2. Per user role: D_{umax}, *u*, $\texttt{active}_{\texttt{UR\_max}}$ *r*
- Cardinality
  - Total number of role activations
    1. Per role: N_{active}, $[N_{default}]$, $\texttt{active}_{\texttt{R\_n}}$ *r*
    2. Per user role: $N_{uactive}$, *u*, $\texttt{active}_{\texttt{UR\_n}}$ *r*
  - Max number of concurrent activations
    1. Per role: N_{max}, $[N_{default}]$, $\texttt{active}_{\texttt{R\_con}}$ *r*
    2. Per user role: N_{umax}, *u* , $\texttt{active}_{\texttt{UR\_con}}$ *r*

Example GTRBAC access policy for a healthcare system

1	*a.*	(*DayTime*, `enable` DayDoctor), (*NightTime*, `enable` NightDoctor)
	b.	((M, W, F), $\texttt{assign}_{\text{U}}$ *Adams* `to` DayDoctor), ((T, Th, S, Su), $\texttt{assign}_{\text{U}}$ *Bill* `to` DayDoctor); ((M, W, F), $\texttt{assign}_{\text{U}}$ *Alice* `to` NightDoctor), ((T, Th, S, Su), $\texttt{assign}_{\text{U}}$ *Ben* `to` NightDoctor)
	c.	([10am, 3pm], $\texttt{assign}_{\text{U}}$ *Carol* `to` DayDoctor)
2	*a.*	($\texttt{assign}_{\text{U}}$ *Ami* `to` NurseInTraining); ($\texttt{assign}_{\text{U}}$ *Elizabeth* `to` DayNurse)
	b.	$c1$ = (6 *hours*, 2 *hours*, `enable` NurseInTraining)
3	*a.*	(`enable` DayNurse $\rightarrow$ `enable` $c1$)
	b.	(`activate` DayNurse `for` *Elizabeth* $\rightarrow$ `enable` NurseInTraining `after` 10 *min*)

Example of Activation Time Constraint

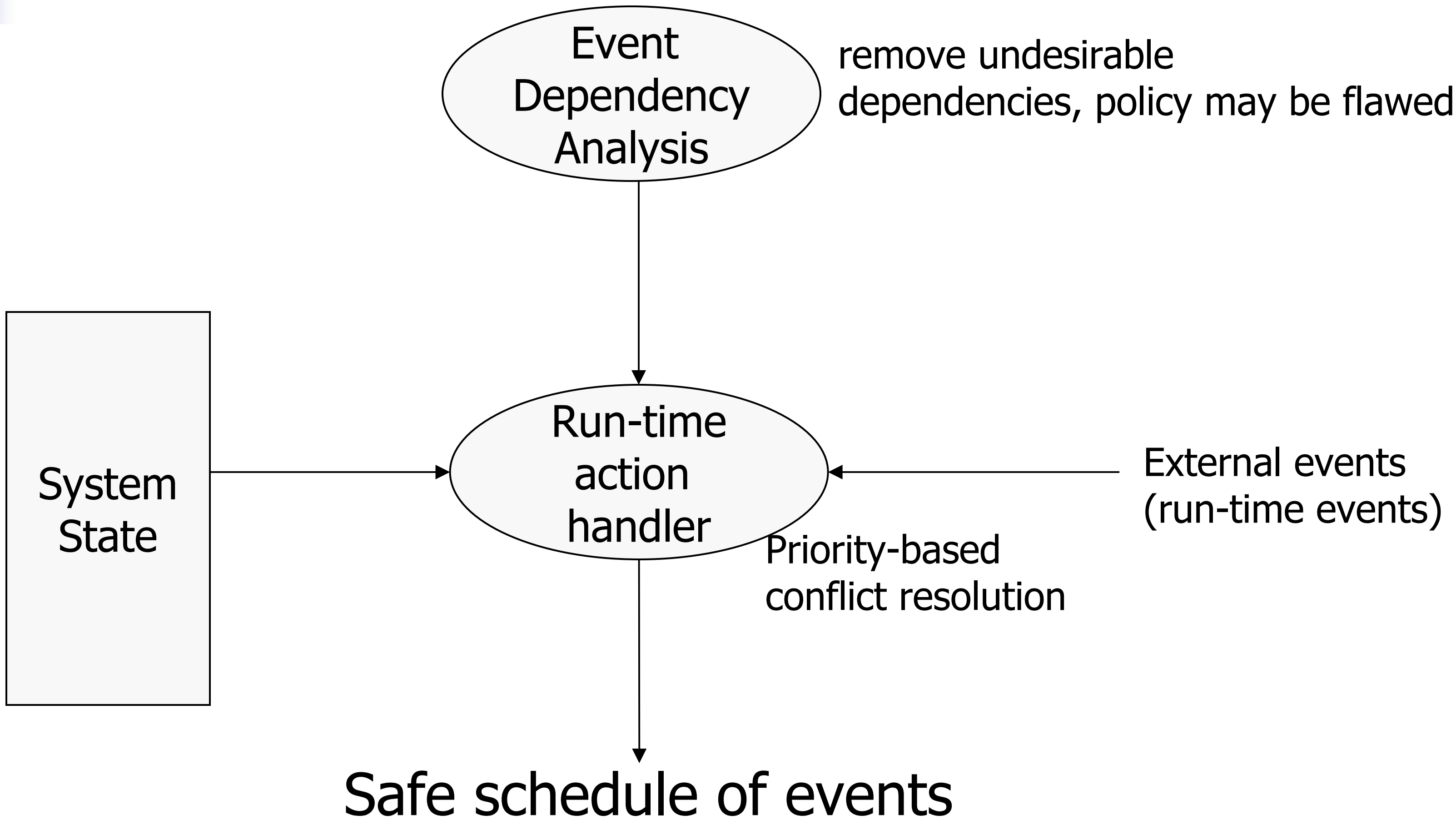
- Video library offers 600 hours of total time per week
- *A, B* and *C* subscribe for 100 hours each
- *D* subscribes for 250 hours
- *E* subscribes for 50 hours



GTRBAC Execution Model for Handling Anomalies

- GTRBAC specification can encounter two possible anomalies
  - Simultaneous occurrence of conflicting events
  - Arbitrary triggering of interdependent triggers can create ambiguity
- Conflict resolution
  - Higher priority takes precedence
  - Disabling event takes precedence when the priorities of the conflicting events are the same
    - `disable` *r* takes precedence over `enable` *r*
  - More specific constraint overrides

GTRBAC Execution Model



Conflicts in GTRBAC

- GTRBAC specification can generate 3 types of conflicts
  - *Type* 1: between events of same type but opposite nature,
    - e.g., `enable` r *vs.* `disable` r
  - *Type* 2: between events of dissimilar types
    - e.g., `activate` r `for` u *vs.* `de-assign` r `to` u OR `disable` r
  - *Type* 3: between constraints

    (a)(*X*, *pr*:*E*) *vs.* (*X*, *q*:*E*) *or* (*X'*,*q*:*Conf*(*E*))

    (b) Per-role *vs.* per-user-role constraints

Handling Conflicts

- *Type* 1 *and Type* 3(a)
  - Higher priority takes precedence
  - Disabling event takes precedence if priorities are the same
    - e.g., `disable` *r* takes precedence over `enable` *r*
- *Type* 2
  - activation event has lower precedence
- *Type* 3(b)
  - per-user-role constraints takes precedence
- Example
  - {`H:enable` r_0, `H:disable` r_0, `VH:enable` r_1, `H:disable` r_1, `VH:(s:activate` r_1 `for` *u*)}
  - After resolution
    {`H:disable` r_0, `VH:enable` r_1, `VH:(s:activate` r_1 `for` *u*)}

Ambiguous Event Dependency

- A set of triggers may give rise to ambiguous semantics
- Example:
  - *tr*1 : `enable R1` → `disable R2`
  - *tr*2 : `enable R2` → `disable R1`
  - Let the runtime requests be: `{enable R1; enable R2}`,

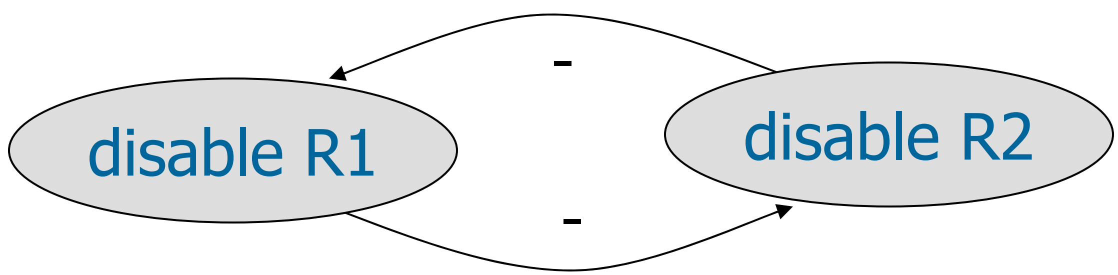
1. *tr*1 fires: `{enable R1; disable R2}`
   (Intuitively, *tr1* blocks *tr*2)
2. *tr*2 fires: `{enable R2; disable R1}`
   (Intuitively, *tr2* blocks *tr1*)

two symmetric possibilities

- Solution: Detect ambiguity using Labeled dependency graph

Dependency Graph Analysis

- Labeled Dependency Graph
  - Directed graph (*N*, *E*)
  - *N*: set of prioritized events that occur in the head of some trigger
  - *E*: set of triples of the form (X, l, Y)
    - For all triggers [$B \rightarrow p$:*E*]
    - For all events *E'* in the body *B*, and for all nodes *q*:*E'* in N
    - <*q*:*E'*, + , *p*:*E*>
    - <*r*:conf(*E'*), -, *p*:*E*> for all [*r*:conf(*E'*)] in *N* such that *q* <= *r*
- Dependency Graph for the Example:



Safe Set of Triggers

- A set of triggers *T* is *safe* if its labeled dependency graph has no cycles with label "-".
- **Theorem:** If a *T* is *safe*, then there exists exactly one execution model.
- **Complexity of** *DAG-based safeness algorithm* : $O(|T|^2)$.

Role Hierarchy in GTRBAC

- Useful for efficient security management of an organization
- No previous work has addressed the effect of temporal constraints on role hierarchies
- GTRBAC-based temporal role hierarchies allow
  - Separation of permission inheritance and role activation semantics that facilitate management of access control
  - Capturing the effects of the presence of temporal constraints on hierarchically related roles

Axioms

- **Axioms:** *For all* $r \in$ `Roles`, $u \in$ `Users`, $p \in$ `Permissions`, $s \in$ `Sessions`, *and time instant* $t \geq 0$, *the following implications hold:*

1. $p\_assigned(p, r, t) \rightarrow can\_be\_acquired(p, r, t)$
2. $u\_assigned(u, r, t) \rightarrow can\_activate(u, r, t)$
3. $can\_activate(u, r, t) \wedge can\_be\_acquired(p, r, t) \rightarrow$

   $can\_acquire(u, p, t)$

Unrestricted Hierarchies
formal definitions

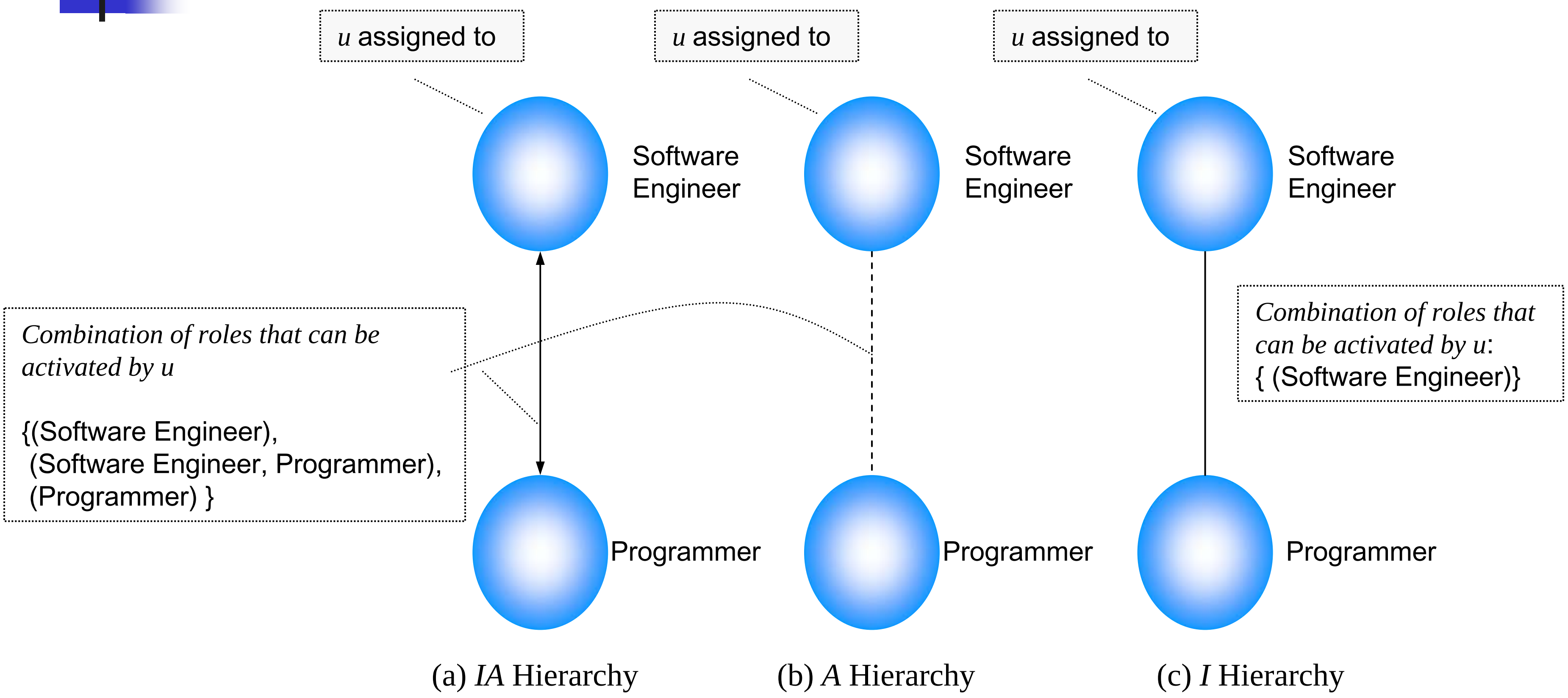
- Unrestricted *I*-hierarchy($x \geq_t y$)
  $\forall p,\ (x \geq_t y) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired(p, x, t)$
- Unrestricted *A*-hierarchy ($x \succeq_t y$)
  $\forall u,\ (x \succeq_t y) \wedge can\_activate(u, x, t) \rightarrow can\_activate\ (u, y, t)$
- Unrestricted *IA*-hierarchy ($x \gtrsim_t y$)
  $(x \gtrsim y) \rightarrow (x \geq_t y) \wedge (x \succeq_t y)$

- Consistency Property:
  Let $<f> \in \{\geq_t, \succeq_t, \gtrsim_t\}$ *and* $<f'> \in \{\geq_t, \succeq_t, \gtrsim_t\}/\{<f>\}$. *Let x and y be distinct roles such that* $x <f> y$; *then the condition* $\neg(y <f'> x)$ *must hold.*

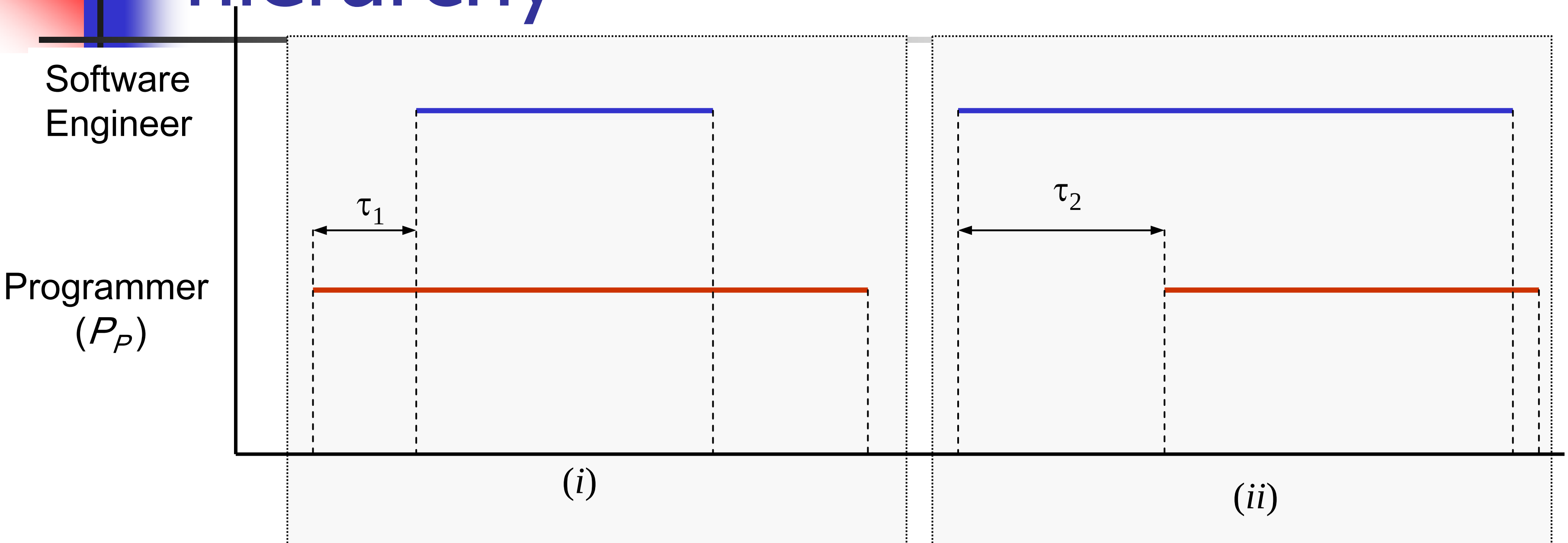
Types of role Hierarchy

- Permission-inheritance hierarchy (I-hierarchy)
  - Senior inherits juniors’ permissions
  - User assigned to senior cannot activate juniors
- Role-Activation hierarchy (A-hierarchy)
  - Senior does not inherit juniors’ permissions
  - User assigned to senior can activate junior
  - Advantage: SOD constraint can be defined on hierarchically related roles
- Activation Inheritance hierarchy (IA-hierarchy)
  - Senior inherits juniors’ permissions
  - User assigned to senior can activate junior

Types of Role Hierarchy



Weakly Restricted temporal role hierarchy



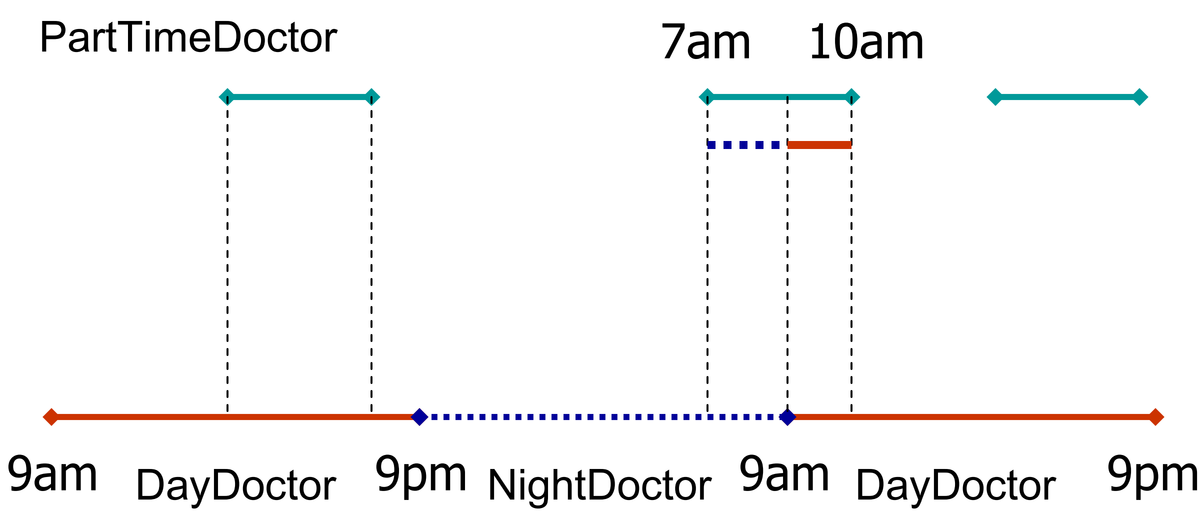
Enabling intervals of Software Engineer and Programmer roles

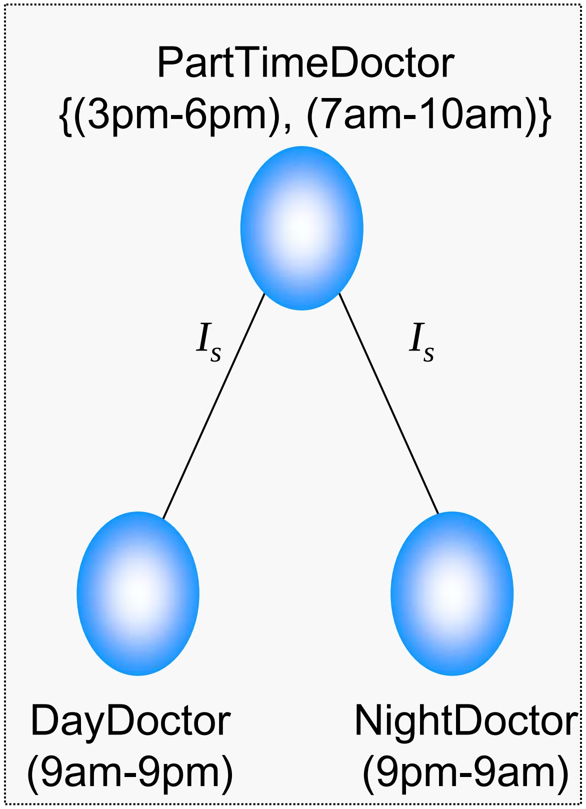
- Weakly Restricted: One role needs to be enabled for the inheritance/activation semantics to apply
  - In I_W, a user assigned to SE can inherit P_P in τ_2.
  - In A_W, a user assigned to SE can activate P in τ_1.

Restricted Hierarchies

Weakly Restricted		
I_w-hierarchy	$(x \geq_{w,t} y)$	$\forall p, (x \geq_{w,t} y) \wedge enabled\ (x, t) \wedge can\_be\_acquired(p, y, t) \rightarrow can\_be\_acquired\ (p, x, t)$
A_w-hierarchy	$(x \geqslant_{w,t} y)$	$\forall p, (x \geqslant_{w,t} y) \wedge enabled(y, t) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t)$
IA_w-ierarchy	$(x \gtrsim_{w,t} y)$	$(x \gtrsim_{w,t} y) \leftrightarrow (x \geq_{w,t} y) \wedge (x \succcurlyeq_{w,t} y)$
Strongly Restricted		
I_s-hierarchy	$(x \geq_{s,t} y)$	$\forall p, (x \geq_{s,t} y) \wedge enabled(x, t) \wedge enabled\ (y, t) \wedge can\_be\_acquired\ (p, y, t) \rightarrow can\_be\_acquired\ (p, x, t)$
A_s–hierarchy	$(x \geqslant_{s,t} y)$	$\forall p, (x \geqslant_{s,t} y) \wedge enabled(x, t) \wedge enabled\ (y, t) \wedge can\_activate(u, x, t) \rightarrow can\_activate(u, y, t)$
IA_s-hierarchy	$(x \gtrsim_{s,t} y)$	$(x \gtrsim_{s,t} y) \leftrightarrow (x \geq_{s,t} y) \wedge (x \succcurlyeq_{s,t} y)$

Temporal Role Hierarchy Example





Hierarchy Constraint Expressions

- Hierarchy $h \in \{I, A, IA\}$:
  - Periodicity: (I, P, enable h),
  - Duration:([I, P| D], D_h, enable h); That is (I, P, D_h, enable h), (D, D_h, enable h) or (D_h, enable h)

Example:

($\geq_t$ is an I-hierarchy and h = (ProjManager $\geq_t$ ProjEngineer))

$$\texttt{enable}\ r \rightarrow \texttt{enable}\ h\ \texttt{after}\ 10\ min$$

Activation Constraints and Temporal Role Hierarchy

- Let P be a permission set for a Software package
  - Only 5 user licenses for the package has been obtained
  - P is assigned to Programmer role,
  - Suppose, we use an activation time cardinality constraint of 5 on Programmer
- Let SE be senior of Programmer
  - With *I* or *IA*-hierarchy: The use of P by more than 5 users at a time can be easily violated
  - With *A*-hierarchy: The use of P by more than 5 users at a time is controlled

Here the cardinality constraint is said to be *permission-oriented*

Activation Constraints and Temporal Role Hierarchy

Requirement: At the most 5 nurses and 3 doctors on active duty; no restriction on permission use

- Apply cardinality of 5 on Nurse role and 3 on Doctor role

- Let Doctor be senior of Nurse
  - With *A*-hierarchy: 3 doctors and 5 nurses can be active at a time but, doctors will not be able to acquire Nurse's permissions.
  - With *I* or *IA*-hierarchy: 3 doctors and 5 nurses can be active at a time, and doctors will also acquire Nurse's permissions.

Here the cardinality constraint is said to be *user-oriented*

Time-based Cardinality, Dependency and Separation of Duty Constraints

- Generic cardinality expression framework
  - Status predicates to capture all the states of GTRBAC (14)
  - Evaluation function and Projection functions
- Control flow dependency constraints
- Time-based SoD constraints
  - Systematic categorization
  - Various time-based semantics

Cardinality Constraints Examples

1	$\lvert\Pi_1 \mathrm{eval}(\mathit{enabled}(r, \text{“}t\text{”})\rvert \geq n$	Number of roles enabled at time “*t*” cannot be less than *n*
2	$\lvert\Pi_1 \mathrm{eval}(\neg \mathit{enabled}(r, \text{“}t\text{”})\rvert \leq n$	Number of roles disabled at time “*t*” cannot be more than *n*.
3	$\lvert\Pi_2 \mathrm{eval}(\mathit{u\_assigned}(\text{“}u\text{”}, r, \text{“}t\text{”}))\rvert \leq n$	Number of roles assigned to “*u*” at time “*t*” cannot be more than *n*
4	$\lvert\Pi_2 \mathrm{eval}(\mathit{can\_activate}(\text{“}u\text{”}, r, \text{"}t\text{”}))\rvert \leq n)$	Set of roles that *u* can activate at time *t* cannot be more than *n*.
5	$(\mathit{Daytime}, \lvert\Pi_1 \mathrm{eval}(\mathit{u\_assignedSet}(u, \text{“Nurse”}, t)\rvert \leq n)$ Number of users assigned to Nurse role in *Daytime* cannot exceed *n*	

Control Flow Dependency Constraints

- Typically used in workflow based systems
- Pre-condition Constraint
  - An event can happen *only if* another has already happened
  - ([*I*, *P*], pre, *Y*, *pr*:*E* after Δ*t* for Δ*d*)
  - E.g.,(*Sat*, pre, `activate` *Manager* `for` *John*, `enable` *Employee*)
- Post-condition Constraint
  - *If* an event happens then another event *must* happen
  - ([*I*, *P*], post, *Y*, *pr*:*E* after Δ*t* for Δ*d*)
  - E.g., (*Sat*, post, `activate` *SysAdmin* `for` *Smith*, `enable` *SysAudit* `after` 30 min)
- Precedence
  - If two events happen then one must always precede another

GTRBAC Separation of Duty Constraints

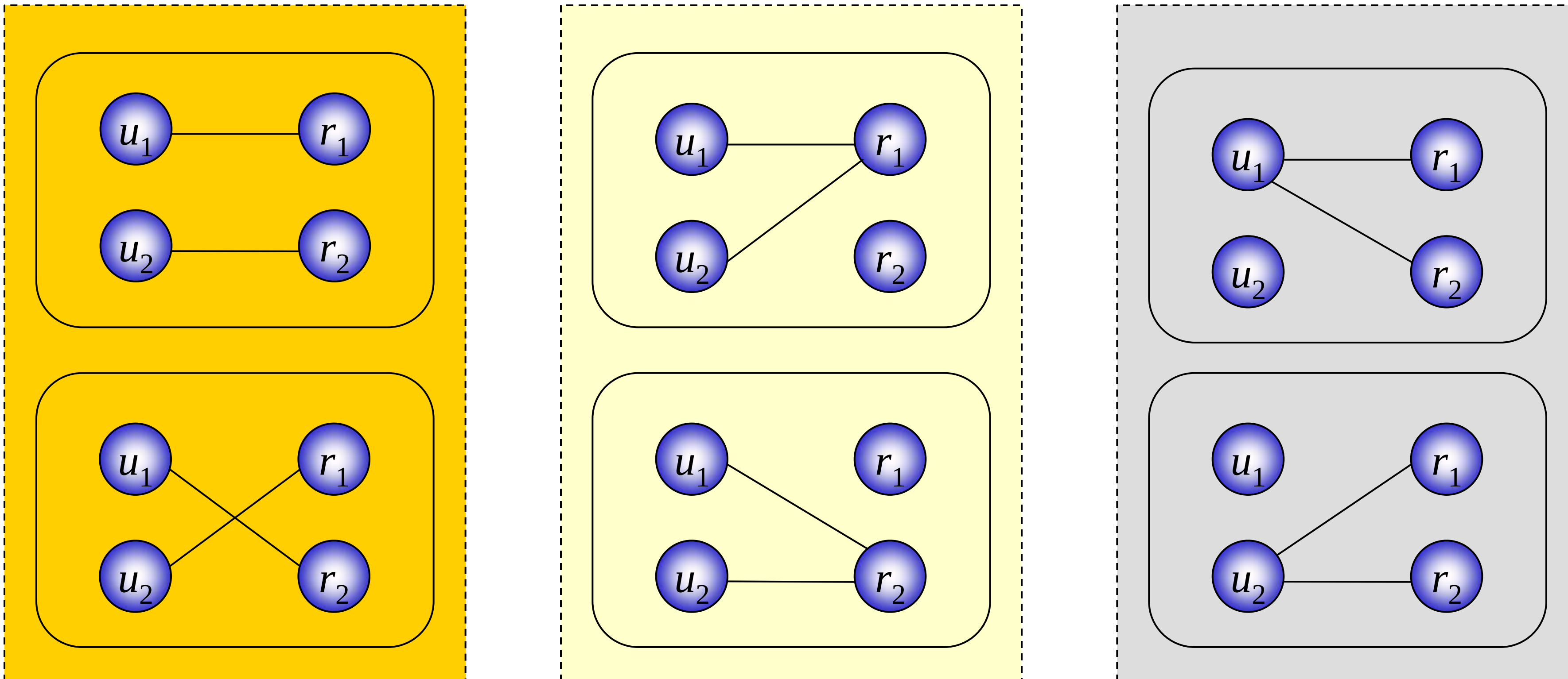
- Important for real world commercial workflow applications and is generally used to prevent fraud
- Categorization of GTRBAC SoDs
  - Role enabling Sods
  - User-role assignment SoDs
  - Role-permission assignment SoDs
  - Activation time SoDs
  - Possibilistic role activation SoDs
  - Possibilistic permission acquisition SoDs

Role Enabling and User-role Assignment SoDs

- Role enabling SoD constraints (*enabled*())
  - (*I*, *P*, EN , R) : No two roles in R can be *enabled* at the same time
    $\forall r_1, r_2 \in \mathrm{R}, SoD \wedge enabled(r_1, t) \rightarrow \neg enabled(r_2, t)$
  - (*I*, *P*, DIS , R) : No two roles in R can be *disabled* at the same time
    $\forall r_1, r_2 \in \mathrm{R}, SoD \wedge \neg enabled(r_1, t) \rightarrow enabled(r_2, t)$
- User assignment SoD constraints (*u_assigned*)
  - UAS-SoD_1 (*I*, *P*, UAS_1, U, R)
    - No two roles in R can be assigned to a user in U at the same time
  - UAS-SoD_2 (*I*, *P*, UAS_2, U, R)
    - No two users in U can be assigned a role in R at the same time
  - UAS-SoD_3(*I*, *P*, UAS_3, U, R)
    - Different users in U cannot be assigned different roles in R at the same time

User-role assignment SoDs

- UAS-SoD$_1$ does not allow *c*; UAS-SoD$_2$ does not allow *b;* UAS-SoD$_3$ does not allow *a*



(a) (b) (c)

User-assignment SoDs (Contd.)

- $\text{UAS-SoD}_4 = \text{UAS-SoD}_2 \wedge \text{UAS-SoD}_3$
  - Roles in R can be assigned to only one user in U
  - Example: one user must complete all the sub tasks
- $\text{UAS-SoD}_5 = \text{UAS-SoD}_1 \wedge \text{UAS-SoD}_3$
  - Users in U can be assigned only one role in R
  - Example: A team should be assigned only one consultancy job (e.g., role ConsultantForBankA)
- $\text{UAS-SoD}_6 = \text{UAS-SoD}_1 \wedge \text{UAS-SoD}_2$
  - A user in U can be assigned to only one role in R (and vice versa)
  - Example: A group of consultants should be assigned to different consultancy jobs (e.g., user A is assigned to role ConsultantForBankA, user B is assigned to role ConsultantForBankB, etc.)

Other GTRBAC SoDs

- Activation Time SoDs (*active*())
  - SoDs involving active roles and sessions
  - Examples
    - No two roles in R can be in *active* state in session(s) of a user in U at the same time
    - No two users in U can have a role in R *active* at the same time
- Possibilistic Activation SoDs (*can_activate*()):
  - Captures implicit /explicit user assignments (A-hierarchy)
  - Similar to user assignment SoDs and
- Possibilistic Acquisition SoDs (*can_be_acquired*())
  - Captures implicit/explicit permission assignment (*I*-hierarchy)
  - Example: A user in U cannot acquire different *permissions* in P at the same time.

Time-based Semantics of SoD Constraints

- Consider $(I, P, \mathrm{UAS}_1, \mathrm{U}, \mathrm{R})$
  - a user in U cannot be assigned to two roles in R
- $(I, P, \mathrm{UAS}_1, \mathrm{U}, \mathrm{R})$ has various forms
  - Weak form: At an instant in (I, P), if a user is assigned to a role in R, at that instant he cannot be assigned to another role in R
  - Strong form: For each interval in (I, P), if there is an instant in which a user is assigned to a role, for no other instant in that interval can he be assigned to another role in R
  - Extended Strong form: At an instant in (I, P), if a user is assigned to a role in R, at no other instant in (I, P) can he be assigned to another role in R

X-GTRBAC
A Policy Specification Language

- An XML conformant specification language for GTRBAC
- Allows identity or credential based dynamic assignment of roles to users
- Allows expressing multidomain policies through role mapping

X-GTRBAC
A Policy Specification Language

```
<!-- Policy Definition--> ::=
<Policy [policy_id = “(value)”]>
        <PolicyName> (name) </PolicyName>
        [<!--XCredType Definition Sheet>]
        [<!--XTemporalConstraint Definition Sheet>]
        <!-- XML User Sheet>
        <!-- XML Role Sheet>
        <!-- XML Permission Sheet>
        <!-- XML User-Role Assignment>
        <!-- XML Role-Permssion Assignment>
        [<!-- XSoD Definition Sheet>]
        [<!-- XHierachy Definition Sheet>]
        [<!-- Local Policy Definitions-->]
        [<!-- Policy Relationship Definitions>]
</Policy>
```

An XML instance of XUS

```
<XUS>
  <Users>
    <User user_id="j1">
         <UserName >John</ UserName >
         <CredType cred_type_id ="C100">
            < type_name >Nurse</type_name>
            <CredExpr>
                  <age> 30 </age>
                  <field> opthalmology </field>
                  <level> 5 </level>
                  <status> single </status>
            </CredExpr>
         </CredType>
         < MaxRoles>2</MaxRoles>
    </User >
    ….
  <Users>
</XUS>
```

An XML instance of XRS

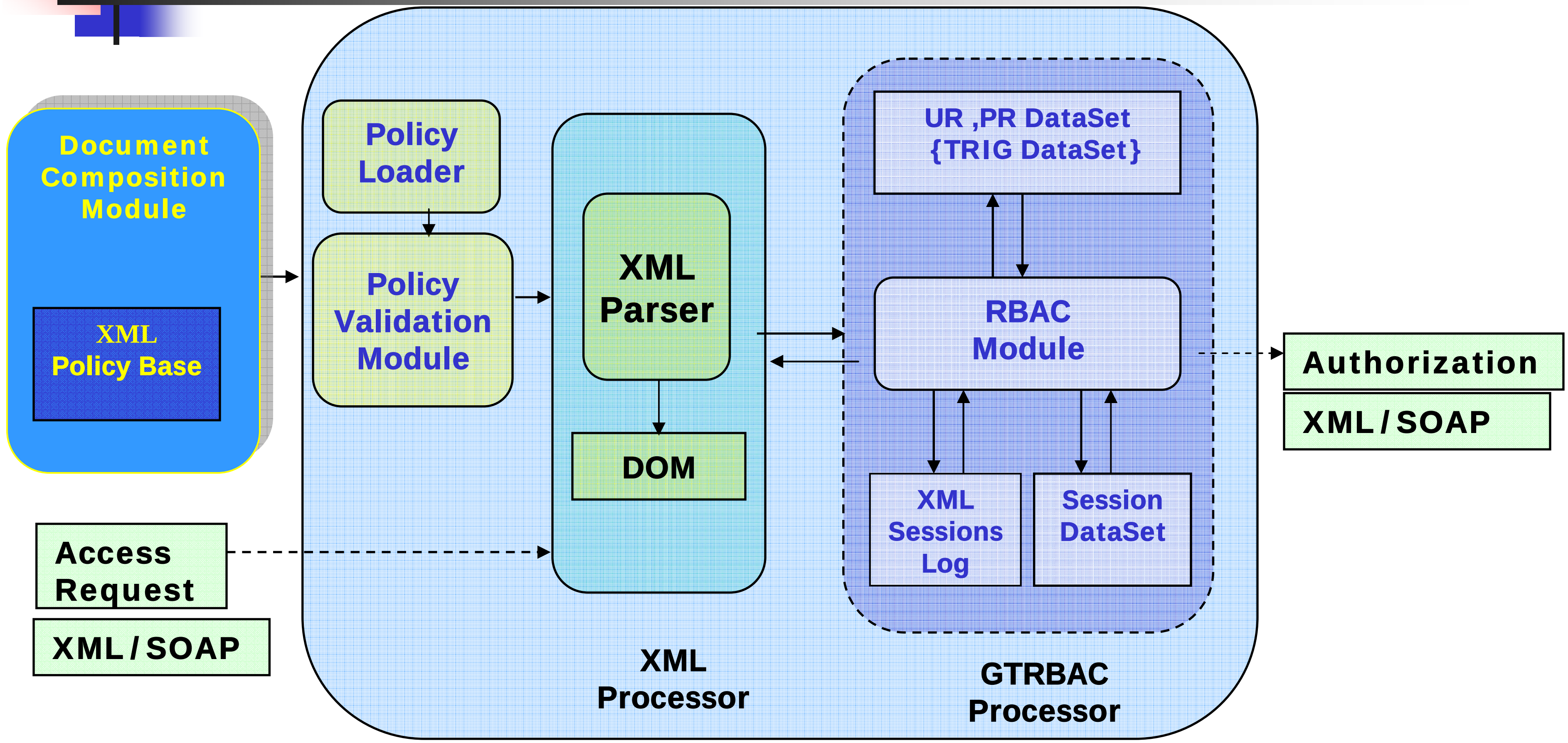
```
<XRS>
   <Role role_id ="R100">
       <RoleName> Nurse </ RoleName >
       <Senior HType = “IA”> Eye_Doctor
       </ Senior>
       <Cardinality> 8 </ Cardinality >
   </Role>
   <Role role_id ="R200">
       <RoleName> Eye_Doctor </RoleName>
       < Junior HType = “IA”> Nurse
       </ Junior>
       <Senior HType = “A”> Eye_Surgeon
       </Senior>
       <Cardinality> 6 </Cardinality>
   </Role>
</XRS >
```

Periodicity and Duration Expressions

```
<XTempConstDef>
  <PeriodicTimeExpr pt_expr_id="PTQuarterWeekOne"
       i_expr_id="Year2003">
    <StartTimeExpr>
        <Year>all</Year>
        <MonthSet>
            <Month>1</Month>
            <Month>4</Month>
            <Month>7</Month>
            <Month>10</Month>
        </MonthSet>
        <WeekSet>
            <Week>1</Week>
        </WeekSet>
    </StartTimeExpr>
  </PeriodicTimeExpr>
 .............
</XTempConstDef>
```

```
<DurationExpr
       d_expr_id="SixWeeks">
        <cal>Weeks</cal>
        <len>6</len>
</DurationExpr>
```

X-GTRBAC Architecture



Validation support is provided by Apache Xalan XSLT engine built into JAXP

Policy Design Issue

- GTRBAC constraint set is not minimal
- Constraint design considerations
  - Usability: Clarity of Semantics, Manageability
  - Complexity of specification

Complexity parameter	**Description**
$n.R$	*n roles*
$n.S$	*n default assignments*
$n.T_r$	*n temporal constraints on (n) roles*
$n.T_{ur}\ (n.T_{rp})$	*n temporal constraints on user- assignment*
$n.A_{ur}\ (n.A_r)$	*n per-user-role (per-role) time constraint*
$n.H$	*n.H indicates n hierarchical relations*

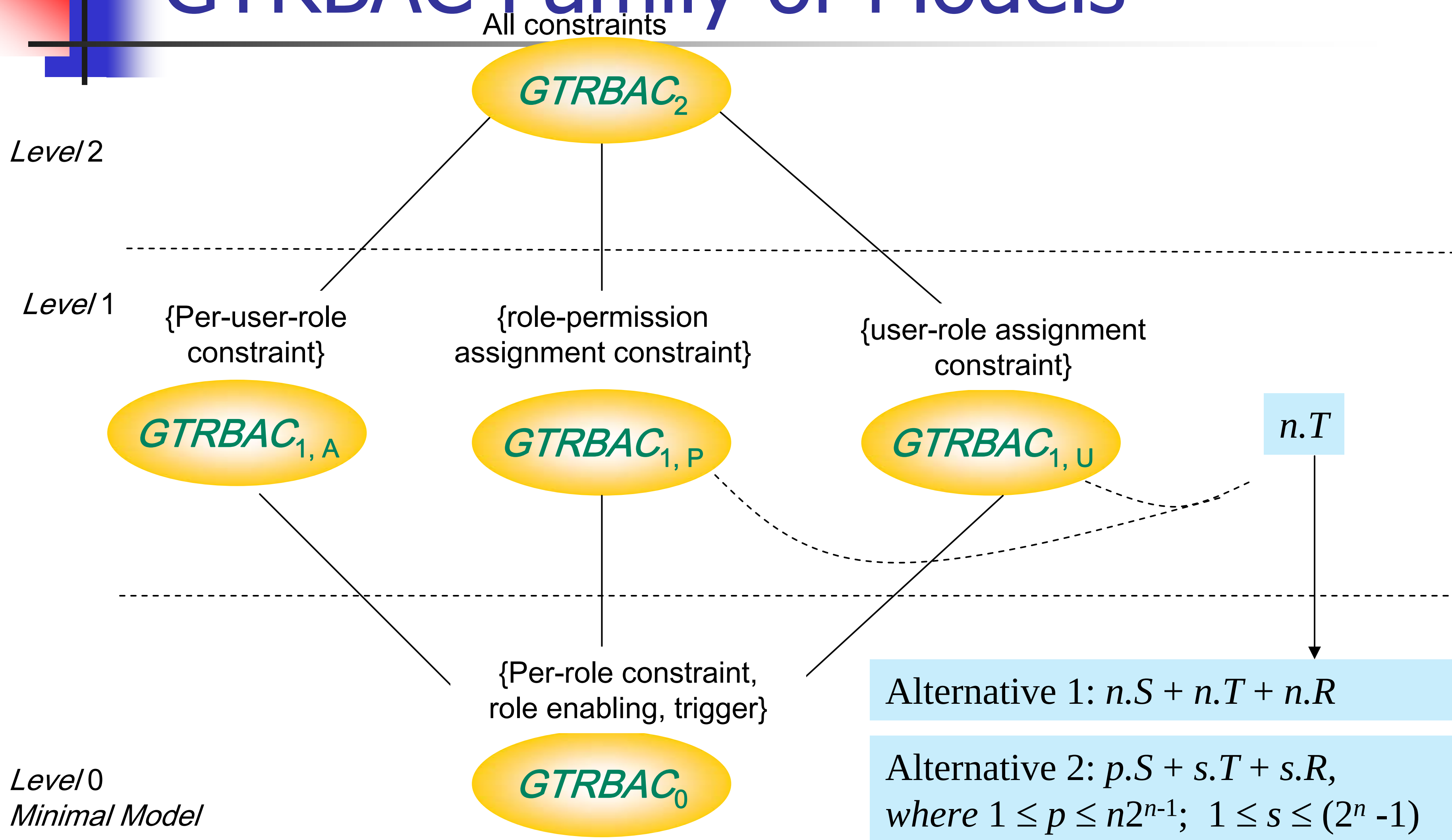
Activity-equivalence

- GTRBAC Configuration *C*:
  - (T, Users, Roles, Permissions, RH), where T is the set of constraints, RH is a *A*-hierarchy.
- Activity-equivalent configurations
  - $C_1 \approx C_2$:

$$(u \overset{C_1}{\underset{t}{\Rightarrow}} p) \Leftrightarrow (u \overset{C_2}{\underset{t}{\Rightarrow}} p)$$

  - u *can acquire* p *at* t *in* C_1
- There exists a minimum set of constraint types

GTRBAC Family of Models

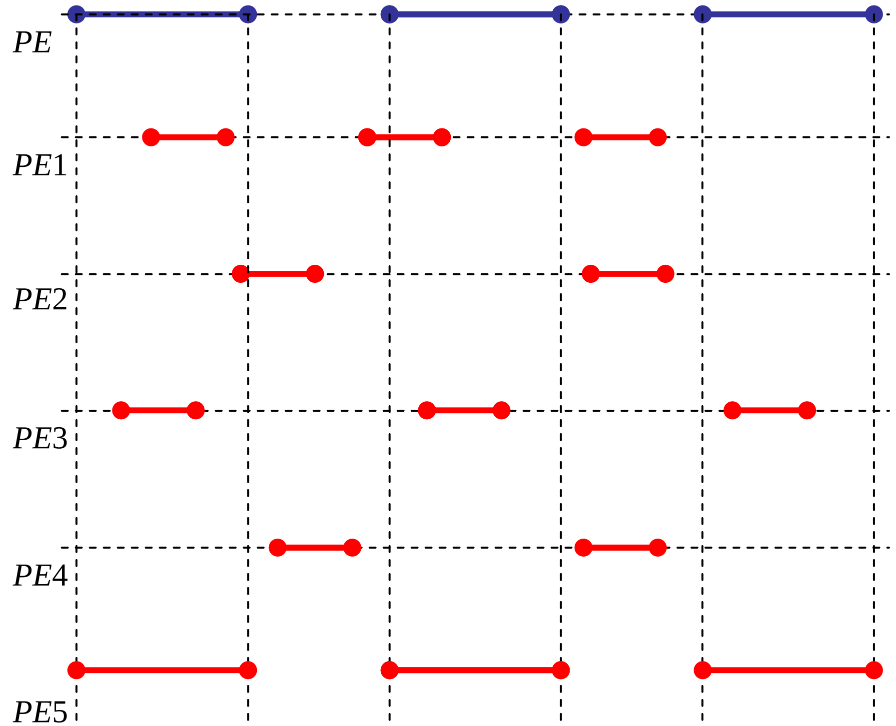


Approach to transformation

- Temporal constraint on user-role assignment is changed to role enabling constraint
- Two approaches
  - Simple approach:
    - PE_u of each u-r assignment becomes enabling constraint for r
  - Minimal Disjoint Set Approach:
    - PE_u of each u-r assignment becomes enabling time constraint for a set of new roles
    - First compute Minimal Disjoint Set (MDS) of all PE_is
    - Create new roles associated with each element of MDS
      - User u is assigned to new roles that have elements of MDS that correspond to PE_u
- Same for the permission role assignment

MDS Approach:Relations on periodic expressions

- Containment: PE1 is contained in PE2
  - All instants in PE1 is in PE2
- Equivalence: PE1 and PE2 have same time instants
- Overlapping: At least one time instant is common to both PE1 and PE2
- Disjoint: No common time instants in PE1 and PE2



PE1, PE2 overlaps with PE
PE3 is contained in PE
PE4 is disjoint from PE
PE5 is equivalent to PE

Minimal Disjoint Set (MDS) & Minimal Subset (MS) of PE over MDS

- Let $PE = \{PE_1, PE_2,.., PE_n\}$
  - $MDS_{PE} = \min_{m\{}\{PE'_i \mid 1 \leq i \leq m\}$ such that
    - $PE'_1, PE'_2, \ldots, PE'_m$ are pair-wise disjoint
    - $PE'_1 \cup PE'_2 \cup \ldots \cup PE'_m = PE_1 \cup PE_2 \cup \ldots \cup PE_n$,
    - If PE'_i contains a time instant of PE_j then it does not contain a time instant that is not in PE_j
  - Bounds: $1 \leq | MDS_{PE} | \leq (2^n - 1)$
- MS of $PE_j \in PE$ over the MDS_{PE}
  - subset of MDS_{PE} that collectively contains all the time instants of PE_j
  - Bounds: $n \leq \sum |MS_{PEi}| \leq n2^{n-1}$

Example

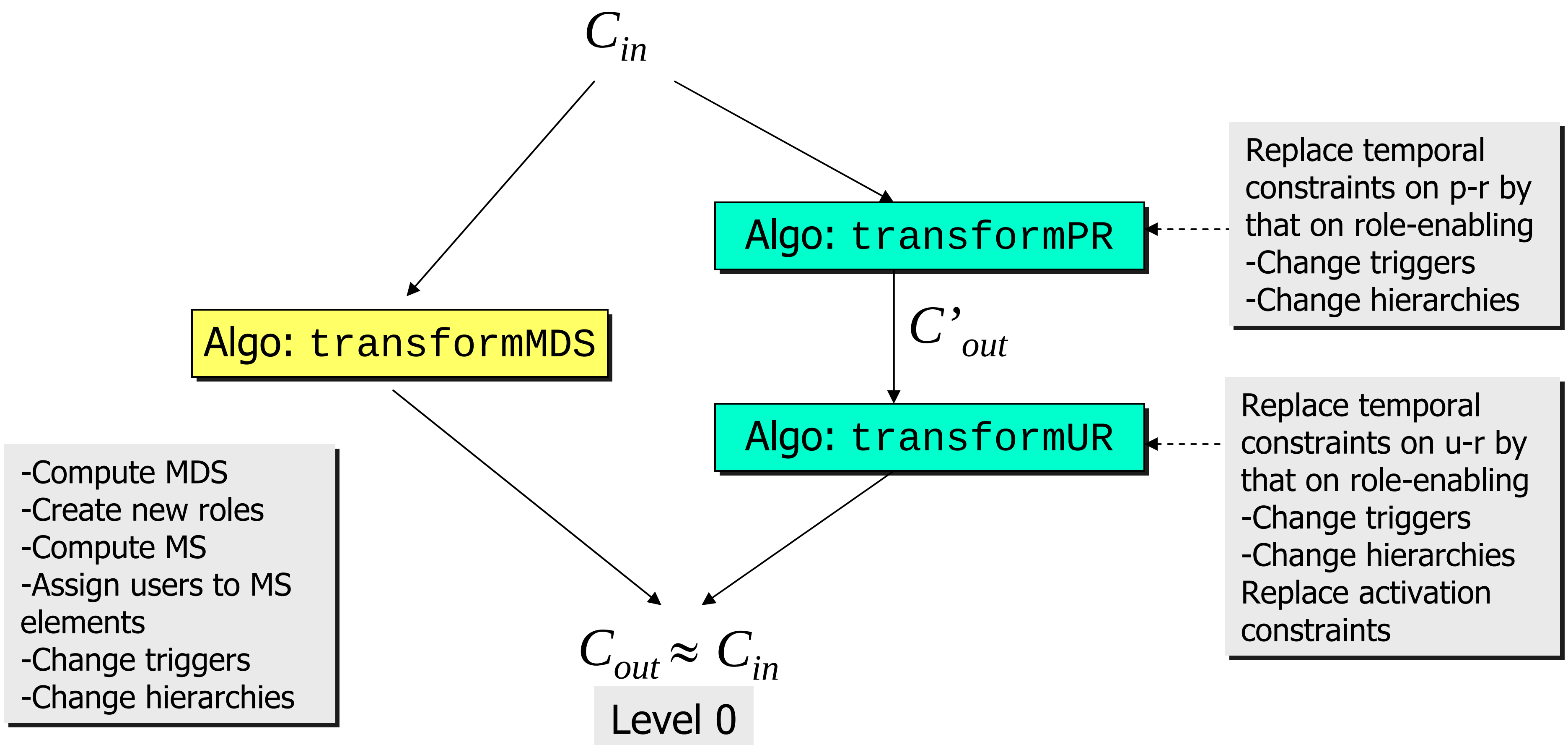
- Let $PE = \{PE_A, PE_B, PE_C, PE_D, PE_E\}$, where
  - PE_A = {Sun, Mon, Tue, Wed, Thu, Fri};
  - PE_B = {Sun, Tue},
  - PE_C = {Sun, Tue, Thu, Fri},
  - PE_D = {Sun, Mon, Tue, Wed, Sat},
  - PE_E = {Thu, Fri}.
- *MDS of PE* is
  - $\{PE_1, PE_2, PE_3, PE_4\}$ = {{Sun, Tue}, {Thu, Fri}, {Mon, Wed}, {Sat}}
- MS values are as follows:
  - *MS of* PE_A : $\{PE_1, PE_2, PE_3\}$;
  - *MS of* PE_B : $\{PE_1\}$
  - *MS of* PE_C : $\{PE_1, PE_2\}$;
  - PE_D : $\{PE_1, PE_3, PE_4\}$
  - PE_E: $\{PE_2\}$

Algorithm `TransformMDS`- Replacing temporal constraint on user-role assignment by temporal roles

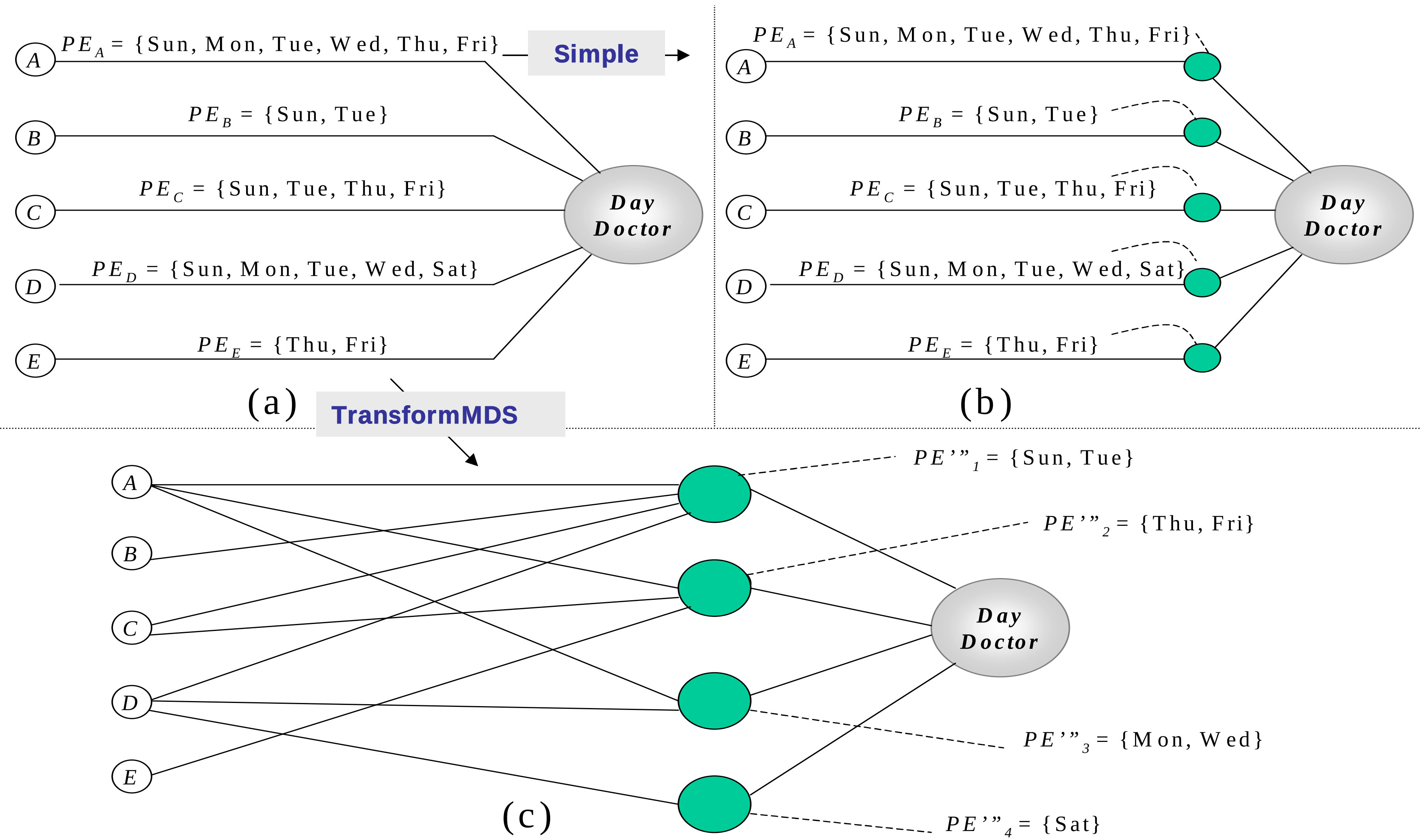
Input: C_{in} **Output**: C_{out}

1. $C_{\text{out}} = \{T', \texttt{Roles}', RH'\} = C_{\text{in}} = \{T, \texttt{Roles}, RH\}$;
2. **FOR** *each* $r \in \texttt{Roles}$ **DO**
3. Let $PE = \{PE_1, PE_2\ldots, PE_n\}$ & $U = \{u_1, u_2\ldots, u_n\}$ be s.t. (PE_i, `assign` r `to` u_i) $\in T'$;
4. Compute MDS of PE; Let the computed $MDS = \{PE'_1, PE'_2\ldots, PE'_n\}$;
5. **FOR** $i = 1$ to n **DO**
6. Compute MS_{PEi} for PE_i
8. **FOR** *each* $PE'_i \in MDS$ **DO**
9. Create a unique role r_i;
10. **FOR** all $u_k \in U$ such that $PE'_i \in MS_{PEk}$ for PE_k **DO**
11. Add *default* assignment (`assign` r_i `to` u_k) in T'.
12. Add constraint (PE_i, `enable` r_i) in T'.
13. Remove constraint (PE_i, `assign` r `to` u_i) from T';
14. $\texttt{Roles}' = \texttt{Roles}' \cup \{r_i\}$;
15. $RH' = RH' \cup \{r \prec r_i\}$;

Simple & MDS approaches



Example



Alternative (c): $p.S + s.T + s.R + s.H$, *where* $n \leq p \leq n2^{n-1}$; $1 \leq s \leq (2^n - 1)$

Alternative (b): $n.S + n.T + n.R + n.H$

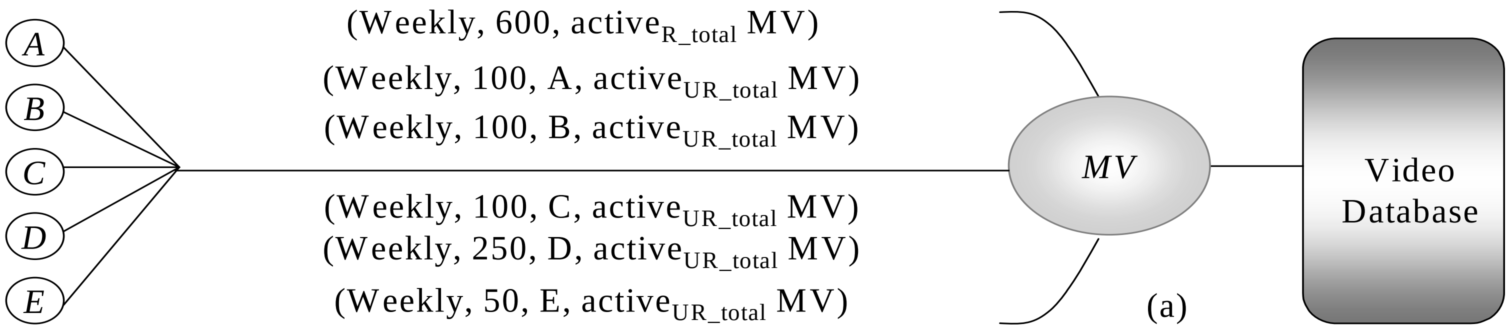
Design Guidelines

- $GTRBAC_{1,U}$ is better than *alternative b* as the policy representation is less complex in terms of number of roles and hierarchies
- $GTRBAC_{1,U}$ is flexible – one can schedule role enabling and user assignments separately
- when p_n and s_n are close to *n* and 1, *alternative c* may be better than $GTRBAC_{1,U}$ representation

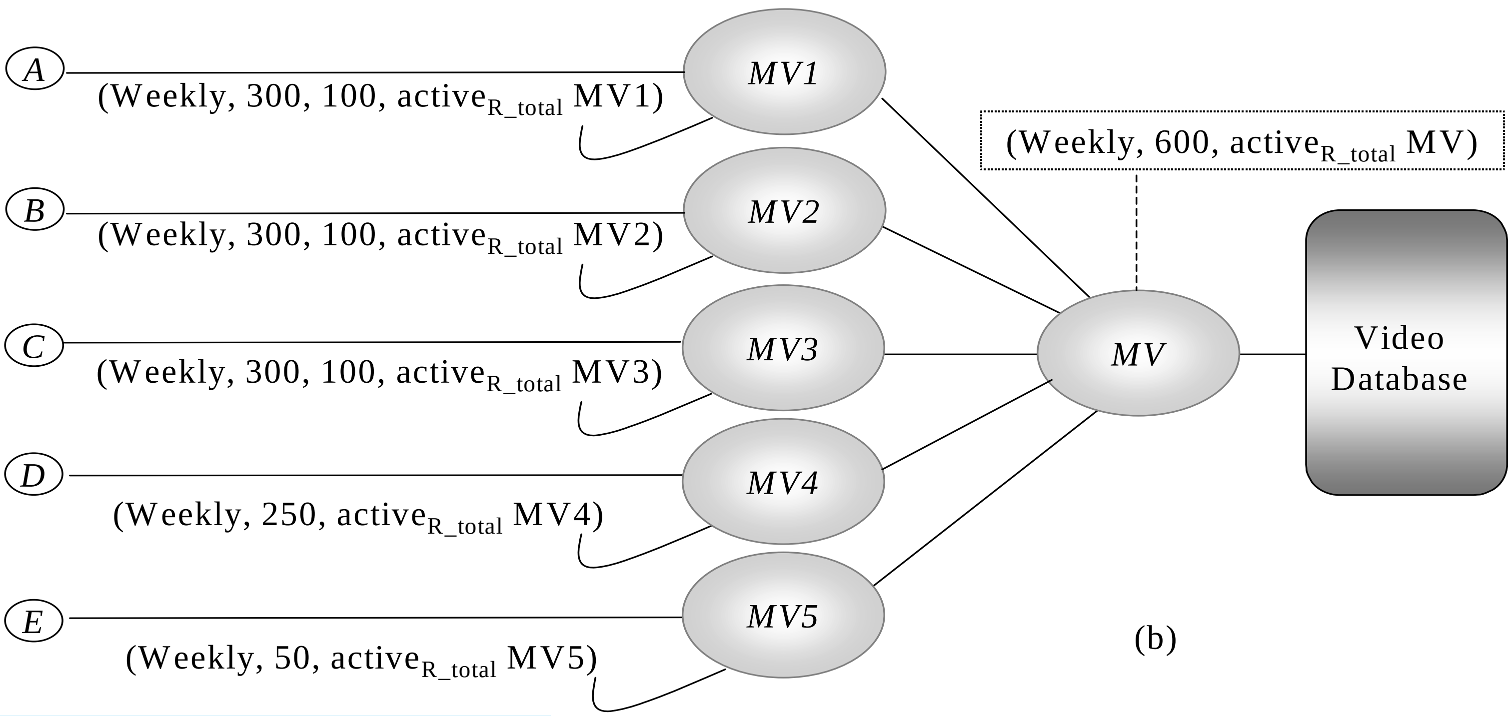
Alternative (b):
$n.S + n.T_R + n.R + n.H$

Alternative (c):
$p.S + s.T_R + s.R + s.H$
where
$n \leq p_n \leq n2^{n-1}$;
$1 \leq s_n \leq (2^n - 1)$

Replacing *per-user-role* by *per-role*

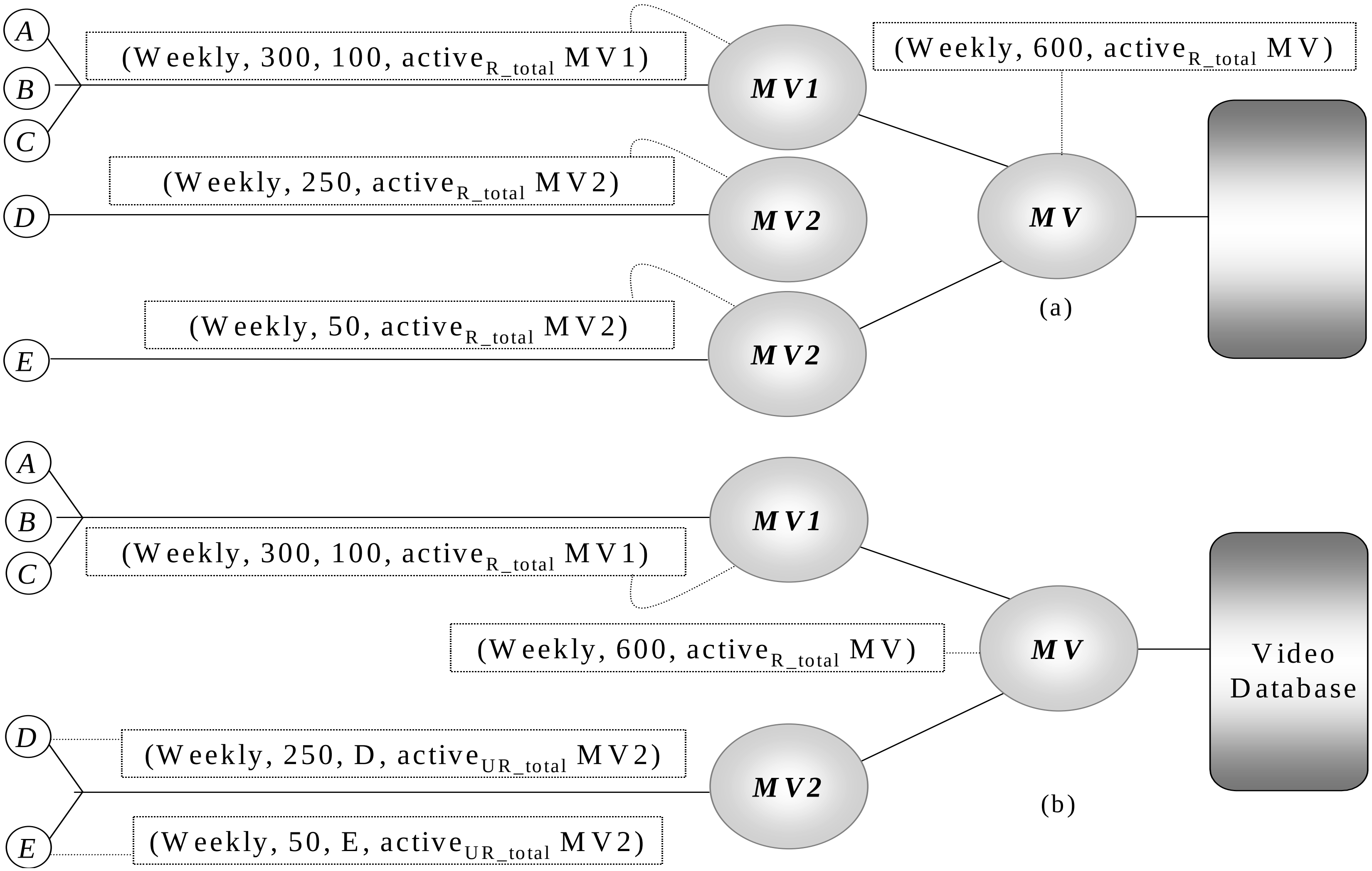


Alternative (a): $n.A_{ur}$, *where* A_{ur} *user-role activation*



Alternative (a): $n.A_r$, $n.R + n.H$

Replacing *per-user-role* by *per-role*



GTRBAC$_{1,A}$ representation: $(n_x - n_y).A_{UR} + n_y.A_R + c.(b.n_y + 1).(R + H)$;
where, (1) $n_x = |D_m|$ and $n_y = |D'|$, *such that* (1) $D' \subseteq D_m$, and (2) *if* $d \in D'$ *then* $C_m(d) > 1$; (2) $b = 1$ *if* $(n > n_x)$; $b = 0$ *otherwise*; (3) $c = 1$ *if* $(n > n_x > 0)$; $c = 0$ *otherwise*.

Design Guidelines

- Per-role constraint with default value
  - If there are many common durations,
- per-user-role constraint
  - Per-user requirements vary significantly
  - More flexibility (e.g., requirements vary constantly)
- Hybrid approach (b in previous slide) can give balanced representations

Related Work

- OASIS RBAC Model [Bacon, 02]
  - Precondition on role activation to support active security
  - No triggers
- RSL2000 Constraint Specification Language [Ahn et. al., 2000],
- Need for activation hierarchy [Sandhu, 1999]
  - Identified its usefulness in expressing MAC
- Separation Duty Constraints
  - Listing of useful set of SoD Constraints
    - [Simon et. al., 1997],[Gligor et. al., 1998]
- None address timing issues
- GTRBAC SoDs subsume all the SoD constraints identified
- GTRBAC Triggers and SoDs provide a technical foundation for enforcing history based SoD constraints

Conclusion

- Role based access control can be used to support diverse set of access control requirements
- Time-based access is a crucial requirement in emerging applications
- GTRBAC’s constraint set is not minimal– however, they are beneficial for practical use

Current and Future work

- Secure Interoperation
  - Integer Programming approach (for tightly coupled environments)
  - Trust-based access management (loosely coupled environment)
    - Issues related to Grid, P2P
- GTRBAC - extended to LoT-RBAC
  - Implementation in mobile environment (near completion)
- Policy evolution and hybrid hierarchy management
  - Administrative tools and techniques
- Extension of the policy design work to generate tools for efficient RBAC policy administration

Relevant References

James B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model," *IEEE Transactions on Knowledge and Data Engineering Vol 7, No. 1*, Jan, 2005.

James B. D. Joshi, Elisa Bertino, Arif Ghafoor, "Analysis of Expressiveness and Design Issues for the Generalized Temporal Role Based Access Control Model," *Transactions on Dependable and Secure Computing*, April-June 2005.

James B. D. Joshi, Rafae Bhatti, Elisa Bertino, Arif Ghafoor, "An Access Control Language for Multidomain Environments", *IEEE Internet Computing*, Nov-Dec, 2004

Rafae Bhatti, James B. D. Joshi, Elisa Bertino, Arif Ghafoor, "XML-Based Specification for Web Services Document Security", *IEEE Computer*, Vol. 37, Number 4, April, 2004, pp 41-49.

Rafae Bhatti, James B. D. Joshi, Elisa Bertino, Arif Ghafoor, "X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control", *ACM Transactions on Information and System Security*, Vol. 8, No. 2, Pages 187-227, May 2005.

Basit Shafiq, James B. D. Joshi, Elisa Bertino, Arif Ghafoor, "Secure Interoperation in a Multi-Domain Environment Employing RBAC Policies," *IEEE Transactions on Knowledge and Data Engineering.*

Smithi Piromruen, James B. D. Joshi, "An RBAC Framework for Time Constrained Secure Interoperation in Multi-domain Environment," IEEE Workshop on Object-oriented Real-time Databases (WORDS-2005), 2005.

Suroop M. Chandran, James B. D. Joshi, "LoT-RBAC: A Location and Time-based RBAC Model", accepted the 6th International Conference on Web Information Systems Engineering, November 20-22, 2005, New York City, New York