A Framework for Implementing Role-based Access Control Using CORBA Security Service

Konstantin Beznosov
beznosov@cs.fiu.edu

Center for Advanced Distributed Systems Engineering
School of Computer Science
Florida International University

May 14, 21
1999

We Will Discuss Today:

- CORBA access control model
- Definition of CORBA protection state configuration
- Framework for implementing RBAC models using CORBA Security Service
- Example configurations of CORBA protection state that support RBAC models

RBAC Introduction

- Role-based access control (RBAC) – a family of reference models:
  - permissions are associated with roles, and
  - roles and users are assigned to appropriate roles
- role can represent competency, authority, responsibility or specific duty assignments
- relations
  - between roles
  - between permissions and roles
  - between users and roles

RBAC Introduction (cont'd.)

- established reference models:

  1. unrelated roles (RBAC_0),
  2. role-hierarchies (RBAC_1),
  3. user and role assignment constraints (RBAC_2), and
  4. both hierarchies and constraints (RBAC_3).

- supports three security principals: least privilege, separation of duties and data abstraction.

- purpose – to facilitate access control administration and review.

Problem Statement

- RBAC is getting popular and recognized by the industry and the government
  - Implementations of RBAC concepts in: Oracle, NetWare, Java, DG/UX, object-oriented systems, object-oriented databases, MS Windows NT, enterprise security management systems.
  - proposed rules on security from the DHHS include RBAC
- significant financial investments in CS in commercial and government organizations
- It is important to foresee if CS will fully support RBAC models
- No work in the research community that has explored the potential of CS for support of RBAC reference models

Solution Overview

- Define a configuration of CORBA protection system
- Re-define RBAC models in the language of CORBA protection system
- Identify what needs to be implemented for support of RBAC_0-RBAC_3 besides CORBA security service
- Provide a check-list for users of CORBA Security Service implementations

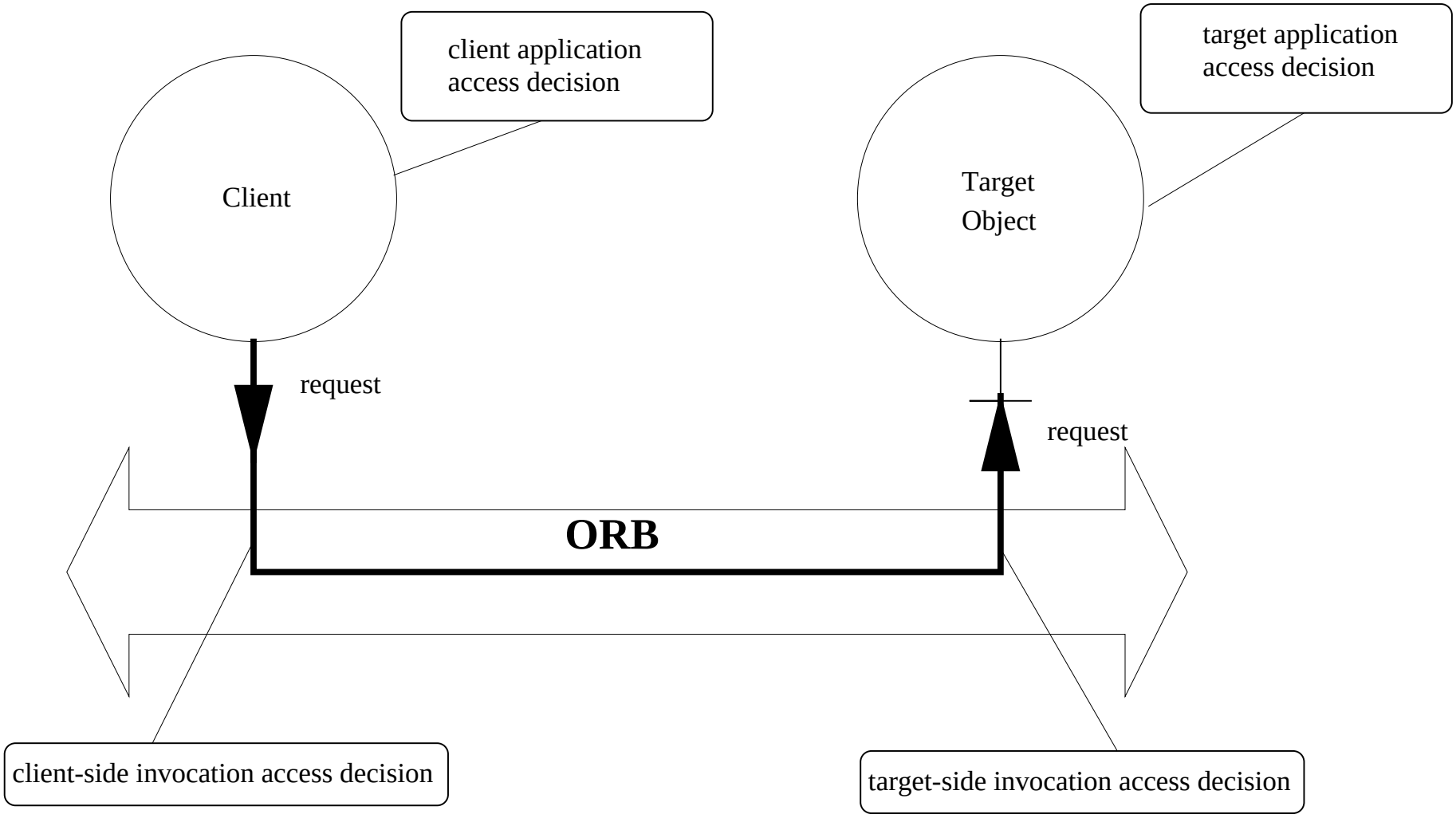
CORBA Security: Basics

- interfaces to a collection of objects for enforcing a range of security policies
- abstraction from an underlying security technology
- not tailored to any particular access control model
- to be adequate for the majority of cases and could be configured to support various access control models

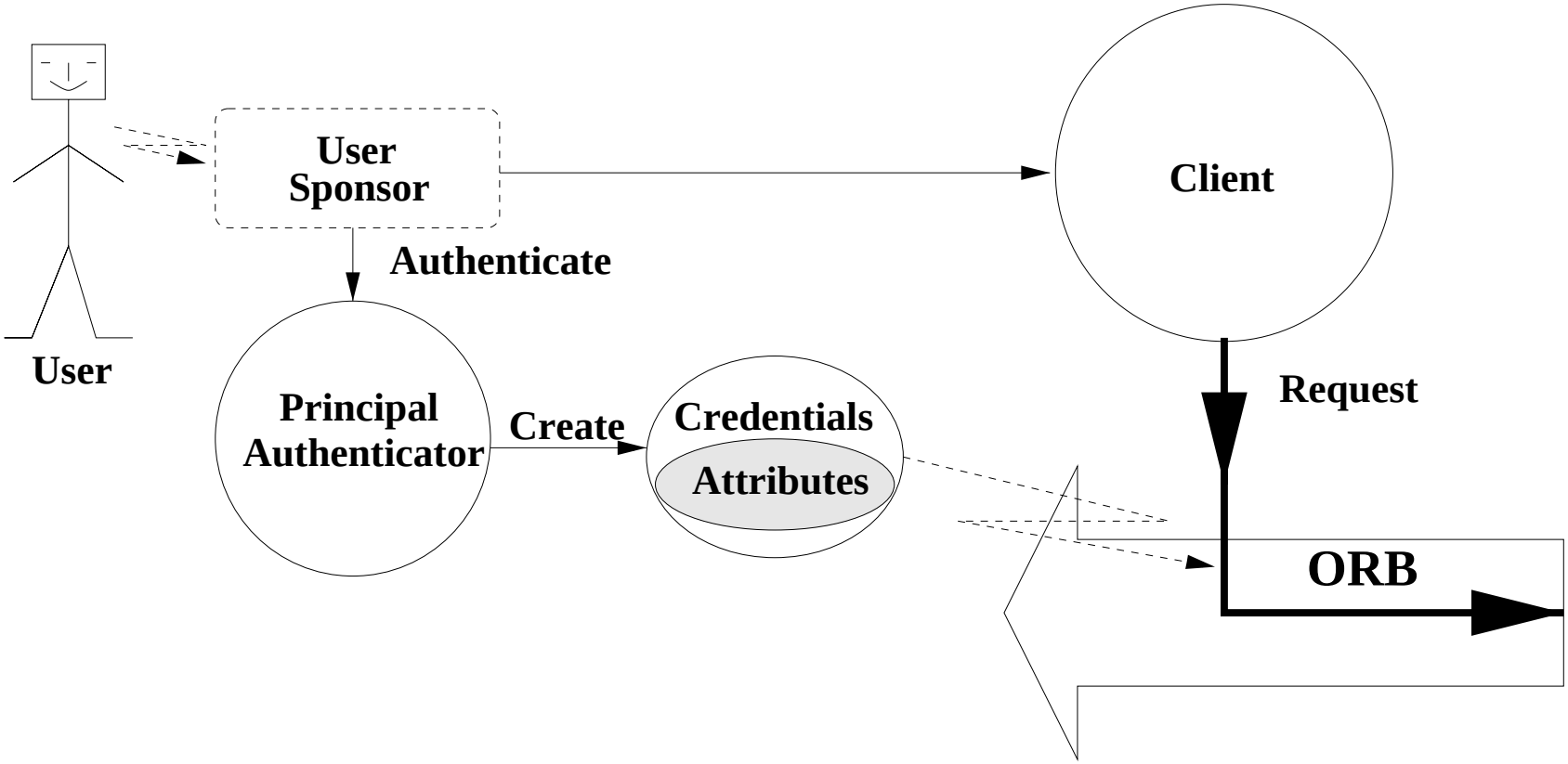
CORBA Security Service (CS): Functions

- identification and authentication
- authorization and access control
- auditing
- integrity and confidentiality protection
- authentication of clients and target objects
- optional non-repudiation
- administration of security policies and related information

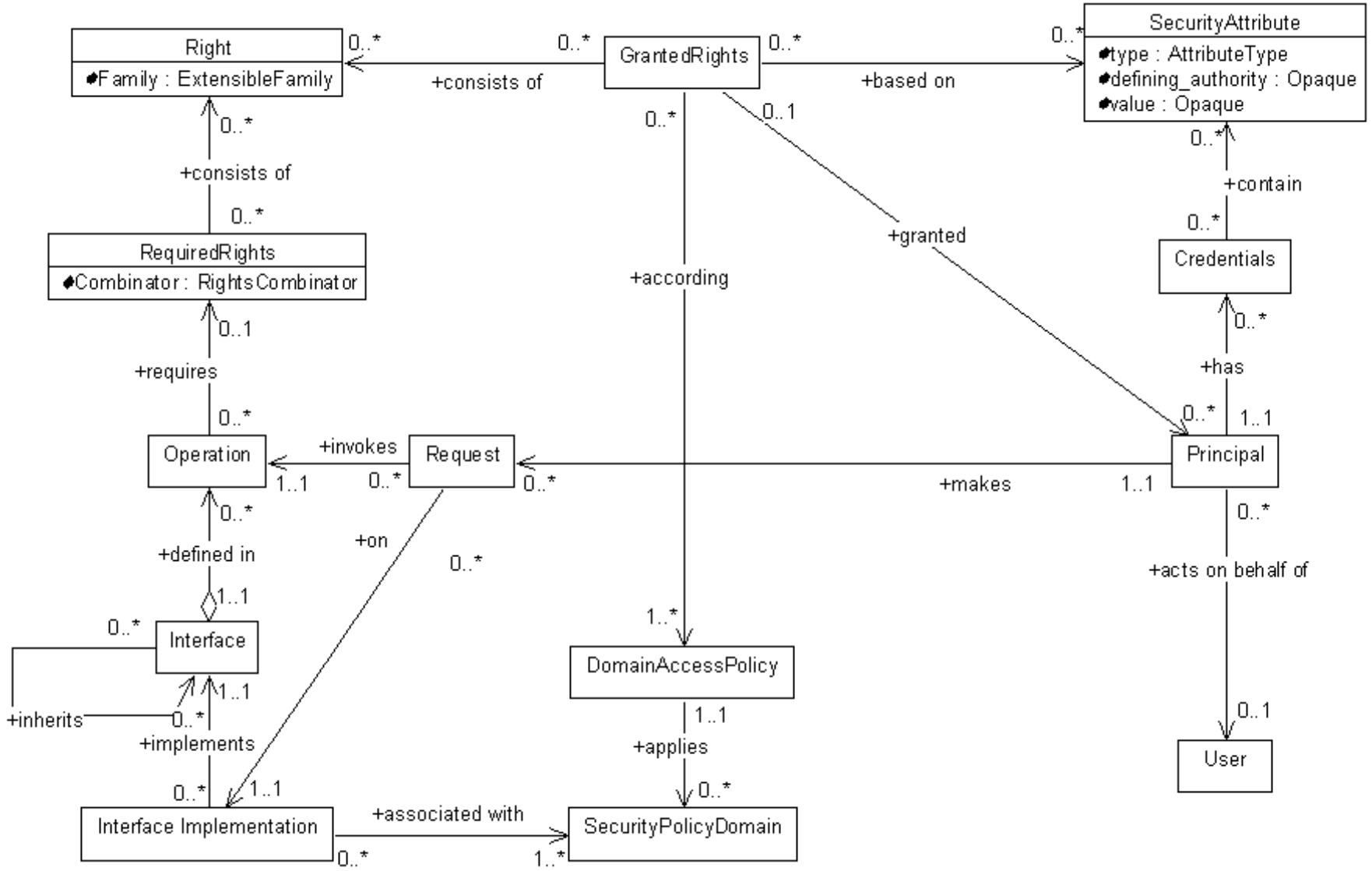
CS: Control Points



CS: User Authentication



CS: Access Control Model



Example for Illustration: Principals

Principal	Attributes
p_1	a_1
p_2	a_2, a_6
p_3	a_2, a_3
p_4	a_4, a_5

Interface and Operations

$\boldsymbol{I} = \{i_0, i_1, ...i_K\}$ a set of interface types in a CORBA-based distributed system S, where

$\boldsymbol{K} = \mid I \mid$ is the size of I

$\boldsymbol{M} = \{m_0, m_1, ..., m_N\}$ a set of all operation names defined in S, where

$\boldsymbol{N} = \mid M \mid$ is the size of M

$\boldsymbol{IM}$ a set of operations from M uniquely identified by interfaces from I that they are defined on

Example: Required Rights Matrix

Operations	Required Rights	Combinator	Allowed
$i_1 m_1$	r_1	all	Only a principal who is granted right r_1
$i_1 m_2$	r_1, r_2	any	Any principal who is granted either r_1 or r_2
$i_2 m_1$	r_2, r_3	all	Only a principal who is granted both $r_2$and r_3
$i_2 m_2$	r_2, r_3, r_4	all	Only a principal who is granted all r_2, r_3, r_4
$i_3 m_1$	r_1, r_2, r_3, r_4	all	Any principal who is granted either of r_1, r_2, r_3, r_4

Example: Granted Rights Matrix

Attributes	Granted Rights	
	Domain	
	d_1	d_2
a_1^i	r_1	r_2
a_2^i	-	r_1
a_3^i	$r_{2,}r_3$	-
a_4^i	r_3	$r_{1,}r_4$
a_5^i	r_1, r_2, r_3	r_2, r_3, r_4
a_6^i	r_6	r_1

Example: Granted Rights per Principal

Principal	Granted Rights	
	Domains	
	d_1	d_2
p_1	r_1	r_2
p_2	r_6	r_1
p_3	$r_{2,}r_3$	r_1
p_4	r_1, r_2, r_3	r_1, r_2, r_3, r_4

CORBA Protection State Configuration

Thirteen-tuple (*A*, *IM*, *O*, *R*, *D*, *C*, *RRM*, *DS*, *IDM*, *GRM*, *effective_rights*, *combine*, *interface_operation*):

$\boldsymbol{A}$ – the set of privilege attributes.

$\boldsymbol{IM}$ – the set of operations uniquely identified by interfaces.

$\boldsymbol{O}$ – the set of distinguishable interface instances.

$\boldsymbol{R}$ – the set of rights.

$\boldsymbol{D}$ – the set of access policy domains.

$\boldsymbol{C = \{all, any\}}$ – the set of rights combinators.

$\boldsymbol{RRM}$ – required rights matrix: [*IM*, Rights] $\subseteq R$, [*IM*, Combinator] $\in C$.

CORBA Protection State Configuration (cont'd.)

$\boldsymbol{DS = \{i, d\}}$ – the set of delegation states.

$\boldsymbol{IDM}$ – the matrix of domain membership for interface instances. [*D*, *O*] $\subseteq\{T,F\}$, $[d, o] == T \Longrightarrow o \in d$.

$\boldsymbol{GRM}$ – granted rights matrix. [*A*, *D*] $\subseteq R$.

effective_rights: $D \times 2^A \longrightarrow 2^R$, a function mapping a set of privilege attributes in a domain to a set of effective rights.

combine: $2^D \times 2^R \longrightarrow 2^R$, a function mapping sets of rights for every domain to a set of effective rights.

interface_operation: $M \times O \longrightarrow IM$, a function mapping an operation name *m* and an interface instance *o* into an interface operation.

Example: Operations Permitted to Principals

Principal	Operations	
	Domains	
	d_1	d_2
p_1	i_1m_1, i_1m_2	i_1m_2
p_2	-	i_1m_1, i_1m_2
p_3	i_1m_2, i_2m_1	i_1m_1, i_1m_2
p_4	i_1m_1, i_1m_2, i_2m_1	i_1m_1, i_1m_2, i_2m_1, i_2m_2, i_3m_1

Example: Access Matrix for Domain d_2

Subjects	Objects		
	i_1	i_2	i_3
p_1	$i_1 m_2$		
p_2	$i_1 m_1$, $i_1 m_2$		
p_3	$i_1 m_1$, $i_1 m_2$		
p_4	$i_1 m_1$, $i_1 m_2$	$i_2 m_1$, $i_2 m_2$	$i_3 m_1$

Some Observations

1. Subjects cannot be objects

2. No operations permitted on one object could be permitted on another object

3. Implementations of the same interface in the same domain are indistinguishable from the access control point of view

$\mathbf{RBAC}_0$: Base Model

Users in RBAC map to users in CS

Roles are represented by set A of privilege attributes of type *role*

Permissions are equivalent to the set of rights R in CS

Sessions are equivalent to principals, which are nothing but sets of security attributes, from CS AC point of view

Correspondence between RBAC and CORBASEC Notations

RBAC		CS	
Meaning	Notation	Meaning	Notation
Users	U	Users	U
Roles	R	Attributes of type “role”	A
Role	r	Attribute of type “role”	a
Permissions	P	Rights	R
permission	p	Right	r
Sessions	S	Principals	P
Session	s	Principal	p

Original $\mathbf{RBAC}_0$ Definition

- *U, R, P, and S* (users, roles, permissions and sessions respectively)
- $PA \subseteq P \times R$, a many-to-many permission to role assignment relation
- $UA \subseteq U \times R$, a many-to-many user to role assignment relation
- *user* : $S \rightarrow$ U, a function mapping each session s_i to the single user *user*(s_i)
- *roles* : $P \rightarrow 2^R$, a function mapping each session s_i to a set of roles *roles*(s_i) $\subseteq$ { *r* | (*user*(s_i), r) $\in$*UA*} and session s_i has the permissions $\bigcup_{r \in roles(s_i)}$ { *p* | (*p*, *r*) $\in$*PA*}

$\mathbf{RBAC}_0$ Definition in the Language of CS

- *U, A, R, P* (users, attributes of type *role*, rights, and principals, respectively)
- $PA \subseteq R \times A$, a many-to-many assignment of granted rights to security attributes of type *role* relation.
- $UA \subseteq U \times A$, a many-to-many user to security attributes of type *role* assignment relation
- $user : P \rightarrow U$, a function mapping each principal p_i to the single user $user(p_i)$, constant for the principal lifetime, and
- $roles : P \rightarrow 2^A$, a function mapping each principal p_i to a set of privilege attributes of type role $roles(p_i) \subseteq \{ a \mid (user(p_i), a) \in A\}$ and principal p_i has the granted rights $\bigcup_{a \in roles(p_i)} \{r \mid (r, a) \in PA\}$

To Support $\mathbf{RBAC}_0$

1. comply with CS standard
2. provide a means to administrate user-to-role assignment relation *UA*
3. provide a means for users to select through *UserSponsor* a set of roles with which they would like to activate the new principal
4. implement *PrincipalAuthenticator* which creates principal credentials containing privilege attributes of type *role* according to relation *UA*
5. implement *PrincipalAuthenticator* which creates principal credentials containing one and only one privilege attribute of type *AccessId*

Original $\mathbf{RBAC}_1$ Definition

- *U*, *R*, *P*, *S*, *PA*, *UA*, and *user* are unchanged from $RBAC_0$
- *RH*$\subseteq R \times R$ is a partial order on R called the role hierarchy or role dominance relation, also written as $\geq$, and
- *roles* : $S \rightarrow 2^R$ is modified from RBAC_0 to require *roles*$(s_i) \subseteq \{$ *r* | $(\exists r' \geq r)$ [(*users*(s_i), r') $\in$*UA*] $\}$ (which can change with time) and session s_i has the permissions $\bigcup_{r \in roles(s_i)} \{$ p | $(\exists r'' \leq r)$[(p, r'') $\in$*PA*] $\}$

RBAC_1 Defintion in CS Language

RBAC_1 is RBAC_0 with role hierarchies. RBAC_1 implemented in CS is formally defined as follows:

- *U*, *A*, *R*, *P*, *PA*, *U*A and *user* are unchanged from RBAC_0.
- $RH \subseteq A \times A$ is a partial order on R called the role hierarchy, written as $\geq$. It is the same as in [**?**].
- *roles* : P $\rightarrow 2^A$ is modified from RBAC_0 to require *roles*$(p_i) \subseteq \{$ *a* | $(\exists\, a' \geq a)$ [(*users*(p_i), a') $\in$*UA*]$\}$ and principal p_i has the granted rights $\bigcup_{a \in roles(p_i)} \{\text{r} \mid (\exists\, a'' \leq a)$ (r, a'') $\in$*PA*$\}$

Implementing RBAC$_1$

- *roles* implemented and enforced by a *Principal Authenticator*
  - A user provides a set of roles to *UserSponsor*
- The *PrincipalAuthenticator* creates new credentials of the principal
  - Credentials have requested by user roles provided that they satisfy the definition of function *roles* for RBAC$_1$
- A valid implementation of RBAC$_1$
  - Allows a user to specify any role junior to those the user is a member of
    - an implementation of *PrincipalAuthenticator* activates all roles which are junior to the specified

To Support RBAC_1

1. Implement RBAC_0

2. Provide a means to administration the role hierarchy relation *RH*

3. Implement *PrincipalAuthenticator* which creates principal credentials containing privilege attributes of type role according to relations *UA*, *RH* as well as function *roles*

$\mathbf{RBAC}_2$: Constraints

Constraints in RBAC are predicates that apply to *UA* and *PA* relations and the *user* and *roles* functions

- Constraints on *UA* – user administrator tools
- Constraints on functions *user* and *roles* – *PrincipalAuthenticator*
- Constraints on *PA* – security administrator tools

To Support RBAC$_2$

1. Implement RBAC$_0$, and

2. Implement support of constraints on *UA* relation user administrator tools, and

3. Implement *PrincipalAuthenticator* with support of constraints on functions *user* and *roles*, and

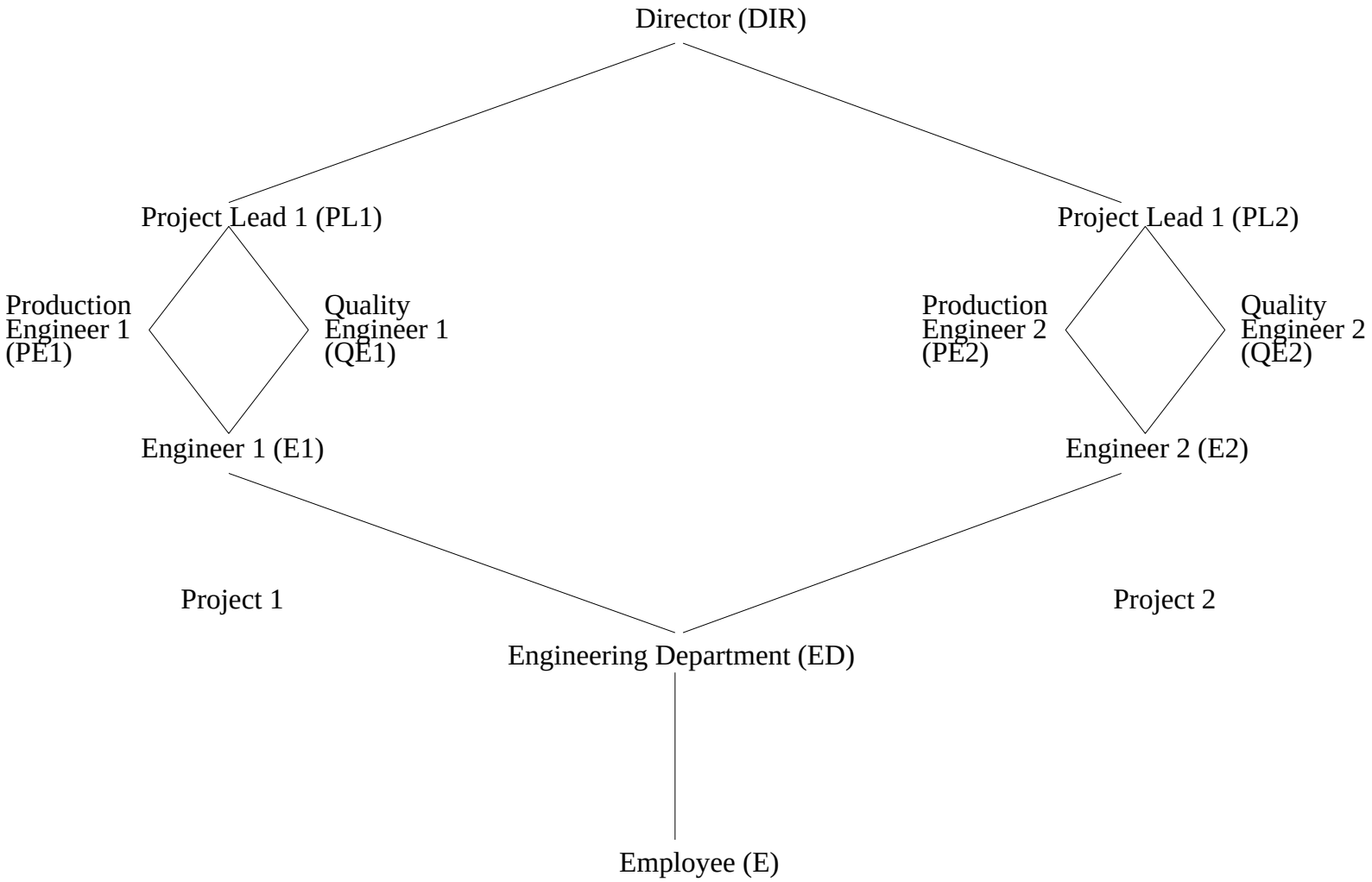
4. Enable enforcement of constraints on *PA* relation by security administration tools.

$\mathbf{RBAC_3}$: $\mathbf{RBAC_1}$ + $\mathbf{RBAC_2}$

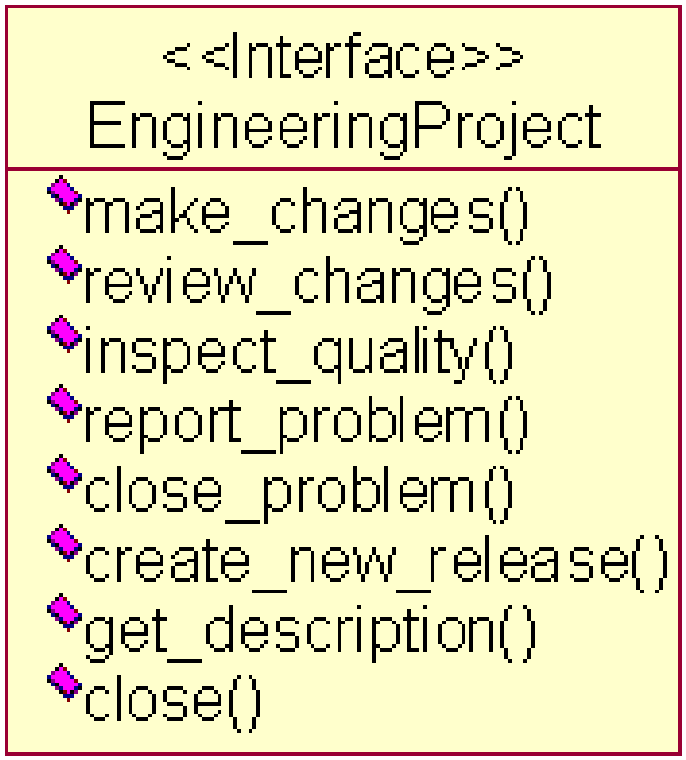
To support $RBAC_3$:

1. Implement $RBAC_1$
2. Implement $RBAC_2$.

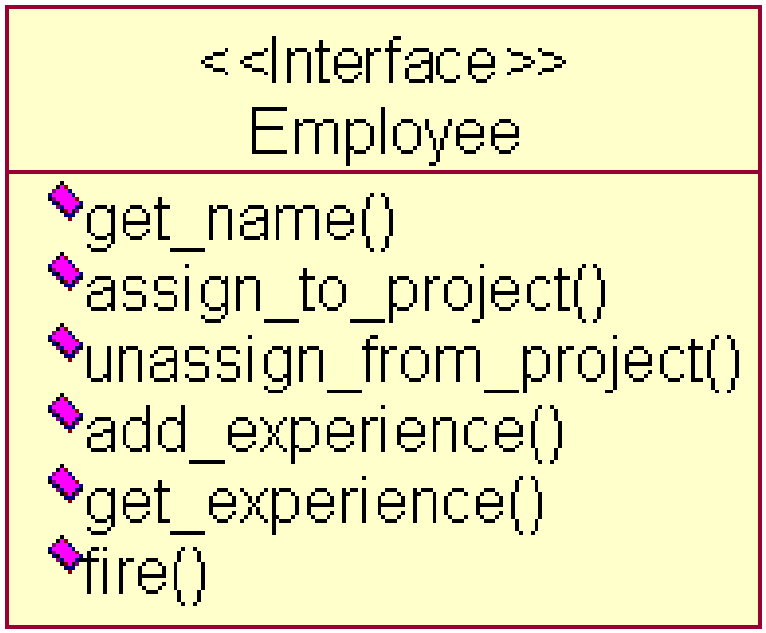
Example Role Hierarchy



Engineering Project Interface



Employee Interface



Hypothetical Access Control Policies

1. Anyone can lookup employee's name and experience.

2. Everyone in the engineering department can get a description of and report problems regarding any project.

3. Engineers, assigned to projects, can make changes and review changes related to their project.

4. Quality engineers can inspect the quality of projects they are assigned to.

5. Production engineers can create new releases.

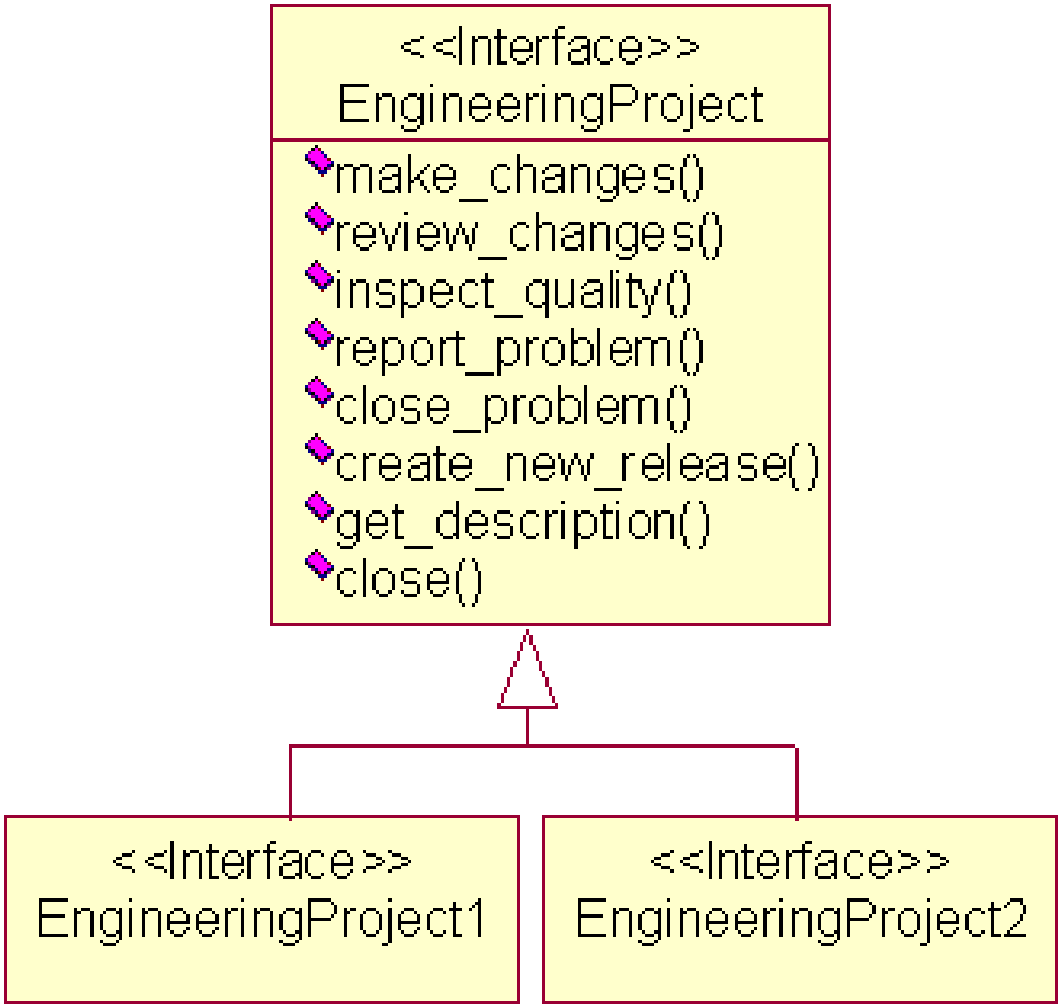
6. Project leaders can close problems.

7. The director can manage employees (assign them to projects, un-assign them from projects, add new records to their experience, and fire) and close engineering projects.

Additional Assumptions

- *effective_rights()* returns a union of granted rights per attribute.
- *combine()* returns a union of rights granted in each domain.

Single Access Policy Domain Solution



Configuration of System Protection State

$\boldsymbol{A}$ = {e, ed, e1, e2, pe1,pe2, qe1, qe2, pl1, pl2, dir}. All these attributes have type *role*.

$\boldsymbol{IM}$ = {Employee::get_name, Employee::assign_to_project, Employee::unassign_fro
Employee::add_experience, Employee::get_experience, Employee::fire,
EngineeringProject1::inspect_quality, EngineeringProject1::make_changes,
EngineeringProject1::report_problem, EngineeringProject1::review_changes,
EngineeringProject1::close, EngineeringProject1::close_problem, EngineeringProj
EngineeringProject1::get_description, EngineeringProject2::inspect_quality,
EngineeringProject2::make_changes, EngineeringProject2::report_problem,
EngineeringProject2::review_changes, EngineeringProject2::close, EngineeringPro
EngineeringProject2::create_new_release, EngineeringProject2::get_description}.

$\boldsymbol{O}$ = {e, ed, e1, e2, pe1,pe2, qe1, qe2, pl1, pl2, dir, prj1, prj2}. *prj1* is an instance of *EngineeringProject1*, and *prj2* is an instances of *EngineeringProject2*. All other elements of *O* are instances of interface *Employee*.

Configuration of System Protection State (cont'd)

$\boldsymbol{R}$ = {gn, atp, ufp, ae, ge, f, mc1, rc1, iq1, rp1, cp1, cnr1, gd1, c1, mc2, rc2, iq2, rp2, cp2, cnr2, gd2, c2}

$\boldsymbol{D}$ = {d1}

$\boldsymbol{C}$ = {*all*} – we use only one combinator.

$\boldsymbol{DS}$ = {*i*, *d*}

IDM – all interface instances are in members of the only access policy domain.

effective_rights (d_j, $a_1, a_2, ...a_l$) $\subseteq \bigcup_{a_i, 1 \leq i \leq l}$ { r | r∈GRM[a_i, d_j] } – union of granted rights per attribute.

combine $(r_{1,d_1}, r_{2,d_1}, ...r_{l,d_1}, ..., \ r_{1,d_p}, r_{2,d_p}, ...r_{m,d_p}) \subseteq \bigcup \{ \ \mathsf{r} \mid r \in \{r_{1,d_1}...r_{m,d_p}\}\}$ – union of rights granted in each domain.

Required Rights Matrix

Operations	Rights
Employee::get_name	gn
Employee::assign_to_project	atp
Employee::unassign_from_project	ufp
Employee::add_experience	ae
Employee::get_experience	ge
Employee::fire	f
EngineeringProject1::get_description	gd1
EngineeringProject1::inspect_quality	iq1
EngineeringProject1::make_changes	mc1
EngineeringProject1::review_changes	rc1
EngineeringProject1::report_problem	rp1
EngineeringProject1::close_problem	cp1
EngineeringProject1::create_new_release	cnr1
EngineeringProject1::close	c1

Required Rights Matrix (cont'd)

Operations	Rights
EngineeringProject2::get_description	gd2
EngineeringProject2::inspect_quality	iq2
EngineeringProject2::make_changes	mc2
EngineeringProject2::review_changes	rc2
EngineeringProject2::report_problem	rp2
EngineeringProject2::close_problem	cp2
EngineeringProject2::create_new_release	cnr2
EngineeringProject2::close	c2

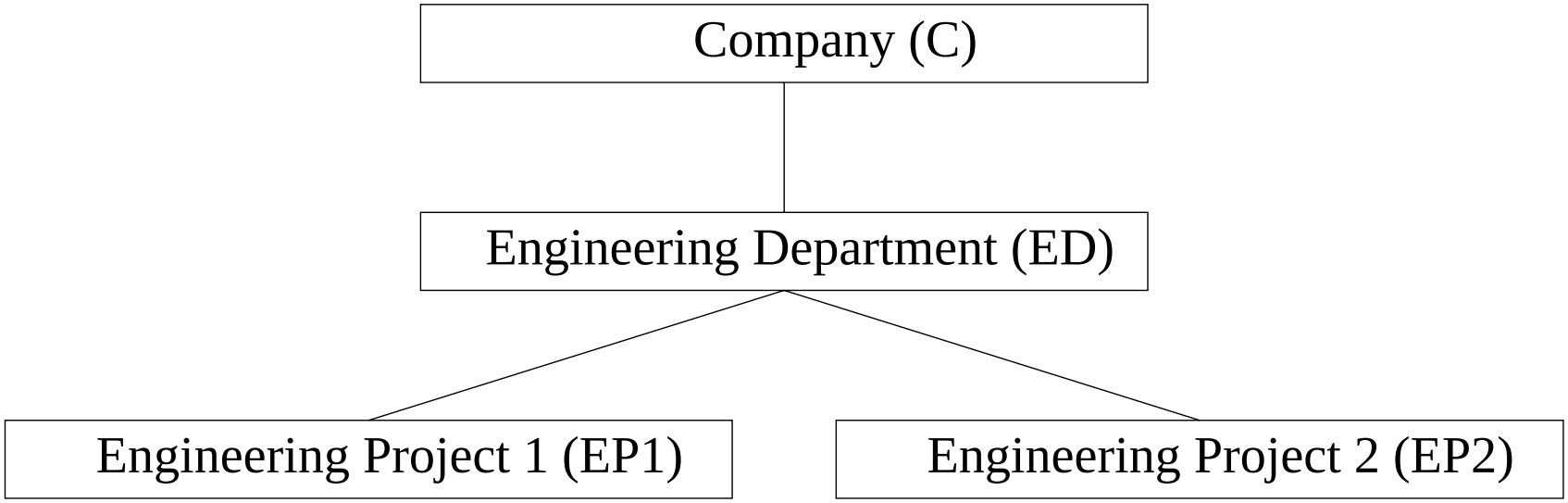
Granted Rights Matrix

Privilege Attribute	Rights
e	gn, ge
ed	gd1, gd2, rp1, rp2
e1	mc1, rc1
pe1	cnr1
qe1	iq1
pl1	cp1
e2	mc2, rc2
pe2	cnr1
qe2	iq1
pl2	cp1
dir	atp, ufp, ae, f, c1, c2

Observations on the Solution

- It works
  - A lead of project 1 with role pl1 activated is able to invoke
    - all instances of *Employee*: *get_name()* and *get_experience()*
    - all instances of *EngineeringProject1*: all but *close()* operations
- Significant Administrative Overhead
  - Gratuitous use of a separate interface (EngineeringProject(1,2)) per project

Multiple Domain Solution



Configuration of a System Protection State

$\boldsymbol{A}$, $\boldsymbol{O}$, $\boldsymbol{C}$, $\boldsymbol{DS}$, $\boldsymbol{effective\_rights}$, $\boldsymbol{combine}$ are the same as in the single domain solution.

$\boldsymbol{IM}$ = {Employee::get_name, Employee::assign_to_project, Employee::unassign_fro
Employee::add_experience, Employee::get_experience, Employee::fire,
EngineeringProject::inspect_quality, EngineeringProject::make_changes,
EngineeringProject::report_problem, EngineeringProject::review_changes,
EngineeringProject::close, EngineeringProject::close_problem, EngineeringProject
EngineeringProject::get_description}.

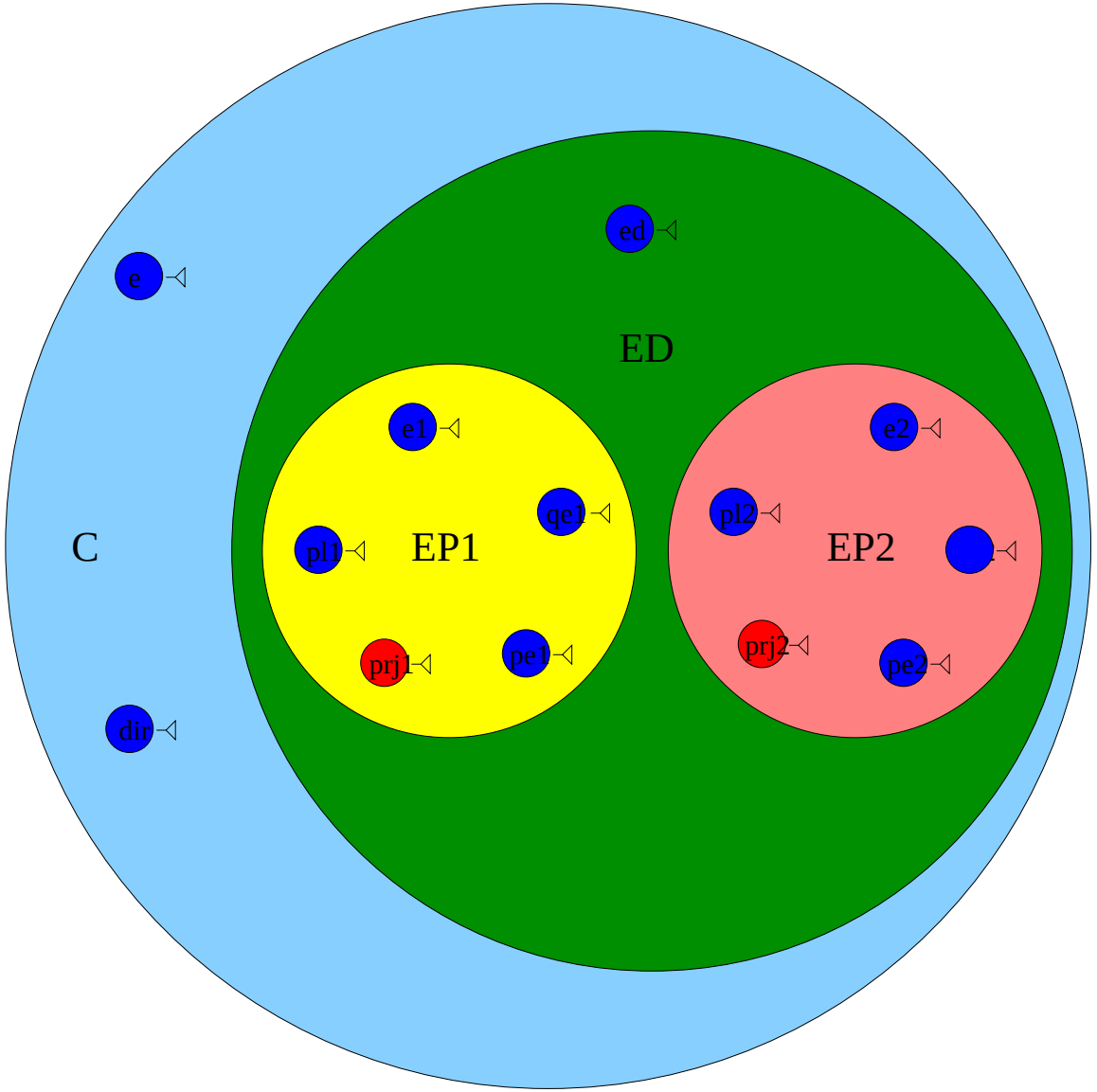
$\boldsymbol{R}$ = {gn, atp, ufp, ae, ge, f, mc, rc, iq, rp, cp, cnr, gd, c}.

$\boldsymbol{D}$ = {C, ED, EP1, EP2}

Required Rights Matrix (RRM)

Operations	Rights
Employee::get_name	gn
Employee::assign_to_project	atp
Employee::unassign_from_project	ufp
Employee::add_experience	ae
Employee::get_experience	ge
Employee::fire	f
EngineeringProject::get_description	gd
EngineeringProject::inspect_quality	iq
EngineeringProject::make_changes	mc
EngineeringProject::review_changes	rc
EngineeringProject::report_problem	rp
EngineeringProject::close_problem	cp
EngineeringProject::create_new_release	cnr
EngineeringProject::close	c

Interface Instance Domain Membership



Interface Instance Domain Membership Matrix (IDM)

Interface	Domains			
Instance	C	ED	EP1	EP2
e	√			
ed	√	√		
e1	√	√	√	
pe1	√	√	√	
qe1	√	√	√	
pl1	√	√	√	
e2	√	√		√
pe2	√	√		√
qe2	√	√		√
pl2	√	√		√
dir	√			
prj1	√	√	√	
prj2	√	√		√

Granted Rights Matrix (GRM)

Attribute	Rights			
	Domains			
	C	ED	EP1	EP2
e	gn	ge	-	-
ed	-	gd, rp	-	-
e1	-	-	mc, rc	-
pe1	-	-	cnr	-
qe1	-	-	iq	-
pl1	-	-	cp, ae	-
e2	-	-	-	mc, rc
pe2	-	-	-	cnr
qe2	-	-	-	iq
pl2	-	-	-	cp, ae
dir	atp, ufp, f, c	-	-	-

Differences

- Allows enforcement of the same policies
- No need in having separate EngineeringProject(1,2) interfaces per project
- No need in having redundant rights.
- RRM and GRM are more comprehensible == easy to manage
- Can also support more flexible policies:
  - Project leaders can add experience to the records of the employees working under supervision of the leaders.
  - Only colleges from the same department can lookup employee experience.

Conclusions

- Implementations compliant with CS specification can support $RBAC_0$–$RBAC_3$.
  - Additional functionality non-specified by CS is required.
    - $RBAC_1$: Implementations of *PrincipalAuthenticator* interface and *UserSponsor* need to be aware of roles and their hierarchies.
    - Support of constraints ($RBAC_2$): a *PrincipalAuthenticator* has to enforce corresponding constraints.
      - Tools to administer user-to-role and role-to-rights relations are also required.

- We set up a framework for implementing as well as for assessing implementations of RBAC models using CS.
  - It provides directions for CS developers to realizing RBAC in their systems.
  - It gives criteria to users for selecting such CS implementations that support models from $RBAC_0$-$RBAC_3$ family.