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

Konstantin Beznosov and **Yi Deng**
{beznosov,deng}@cs.fiu.edu

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

October 28, 1999

Overview

- 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

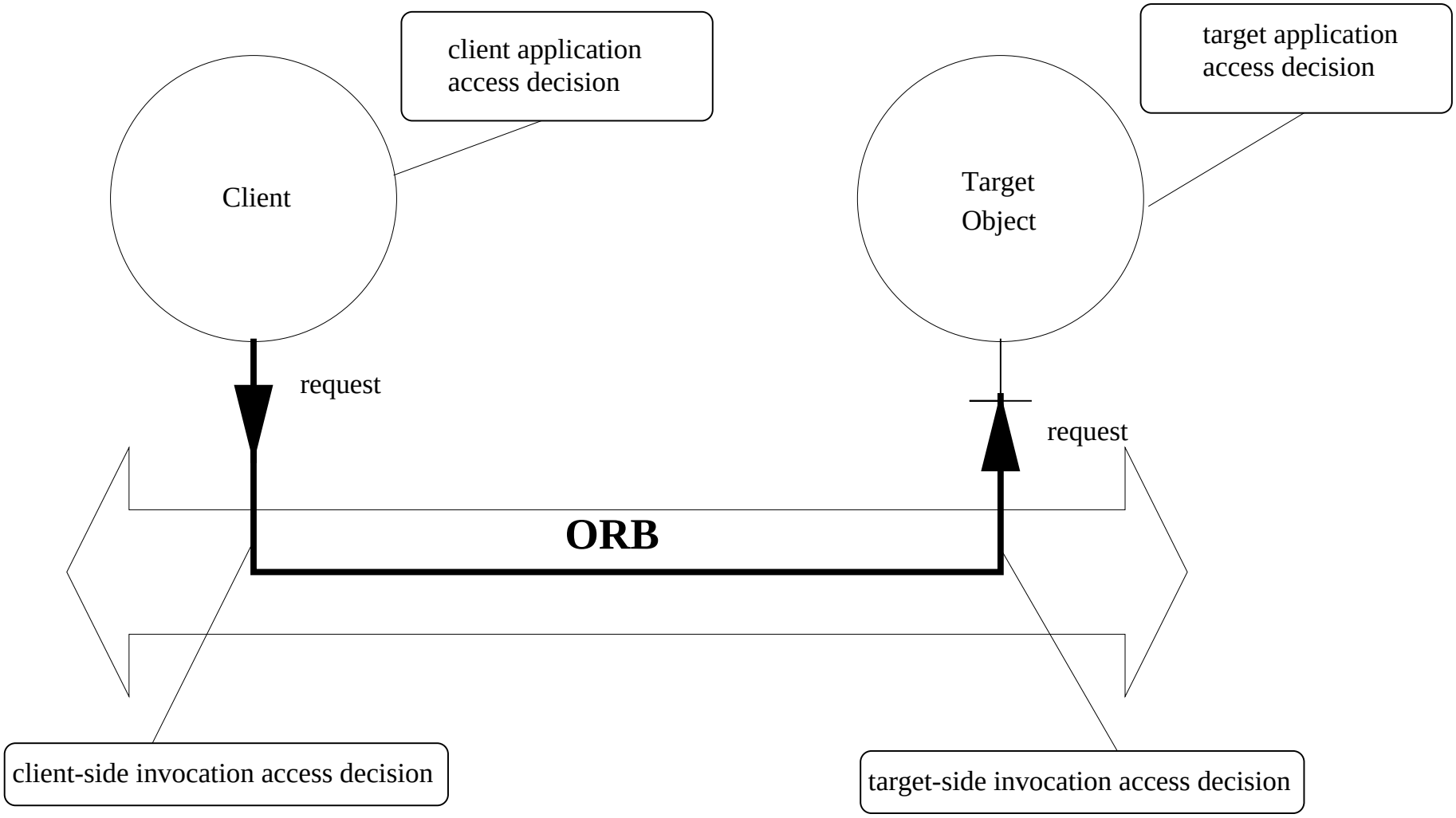
Problem Statement

- 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

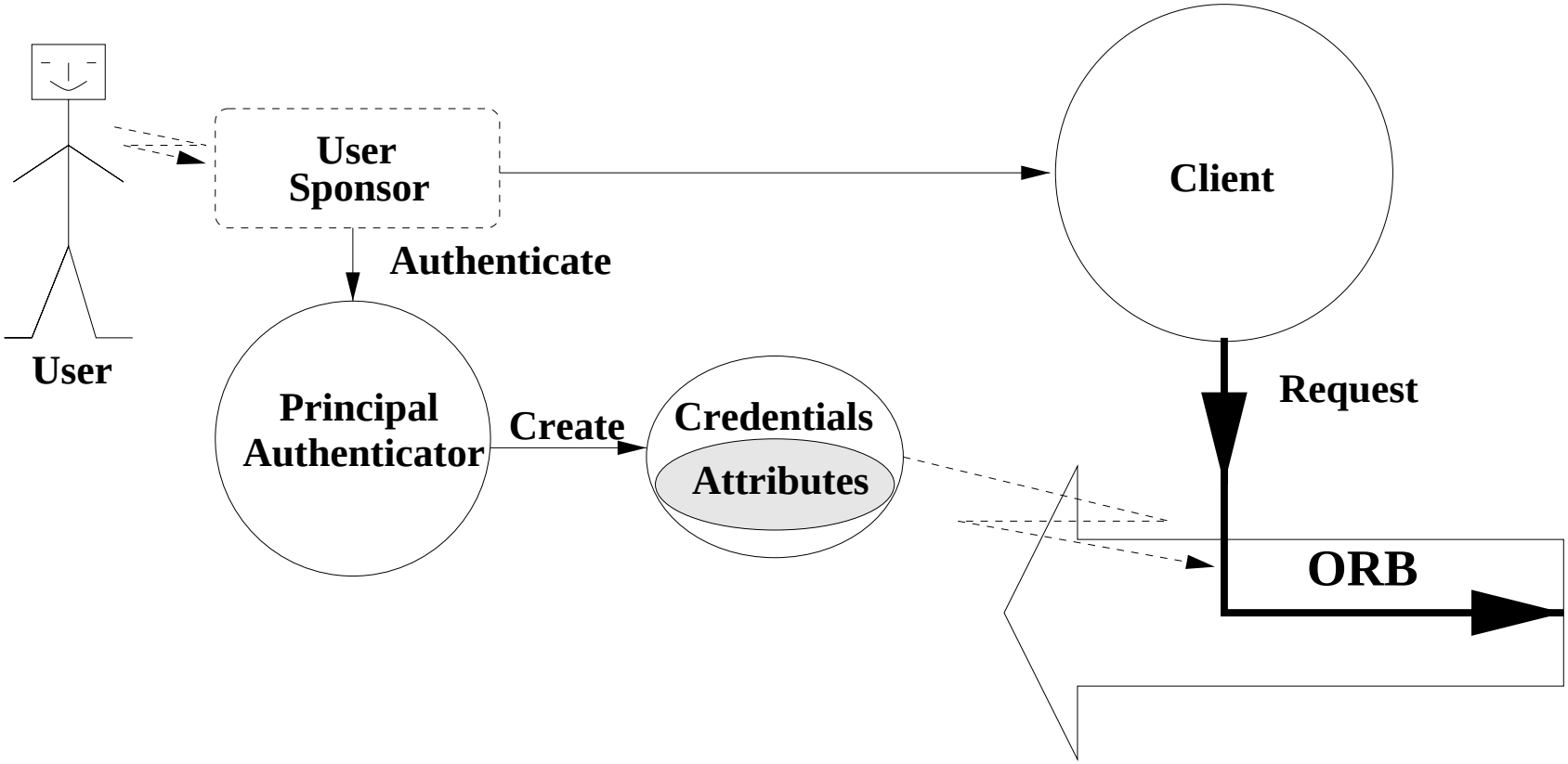
Solution Overview

- 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

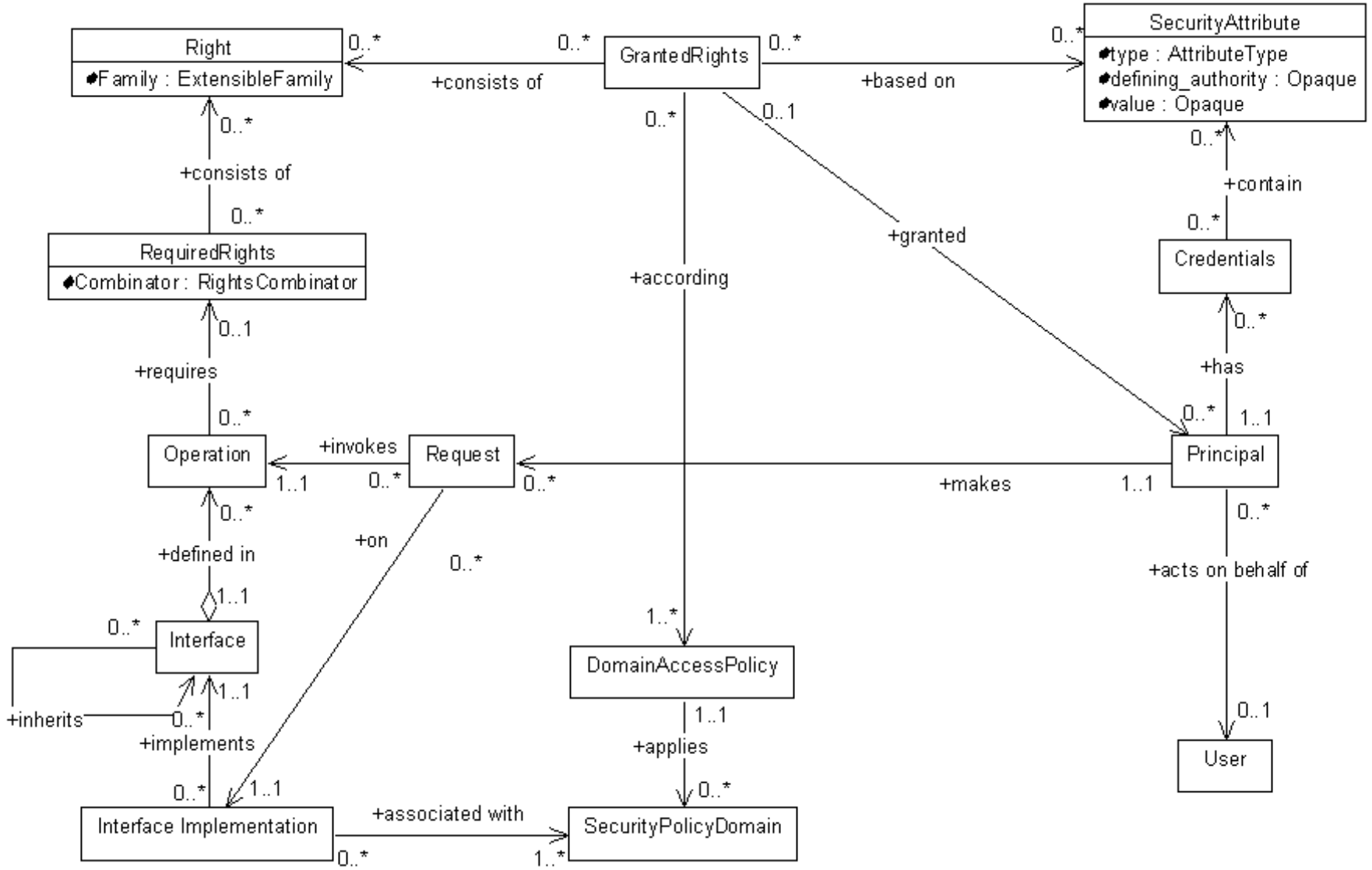
CS: Control Points



CS: User Authentication



CS: Access Control Model



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.

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 $\bigcup_{a \in roles(p_i)} \{r \mid (r, a) \in \text{PA}\}$

To Support $\mathbf{RBAC}_0$

1. comply with CS standard
2. provide a means to administer *UA* relation
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 Definition 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 $\geq$
- *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 roles requested by the user 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

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*

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

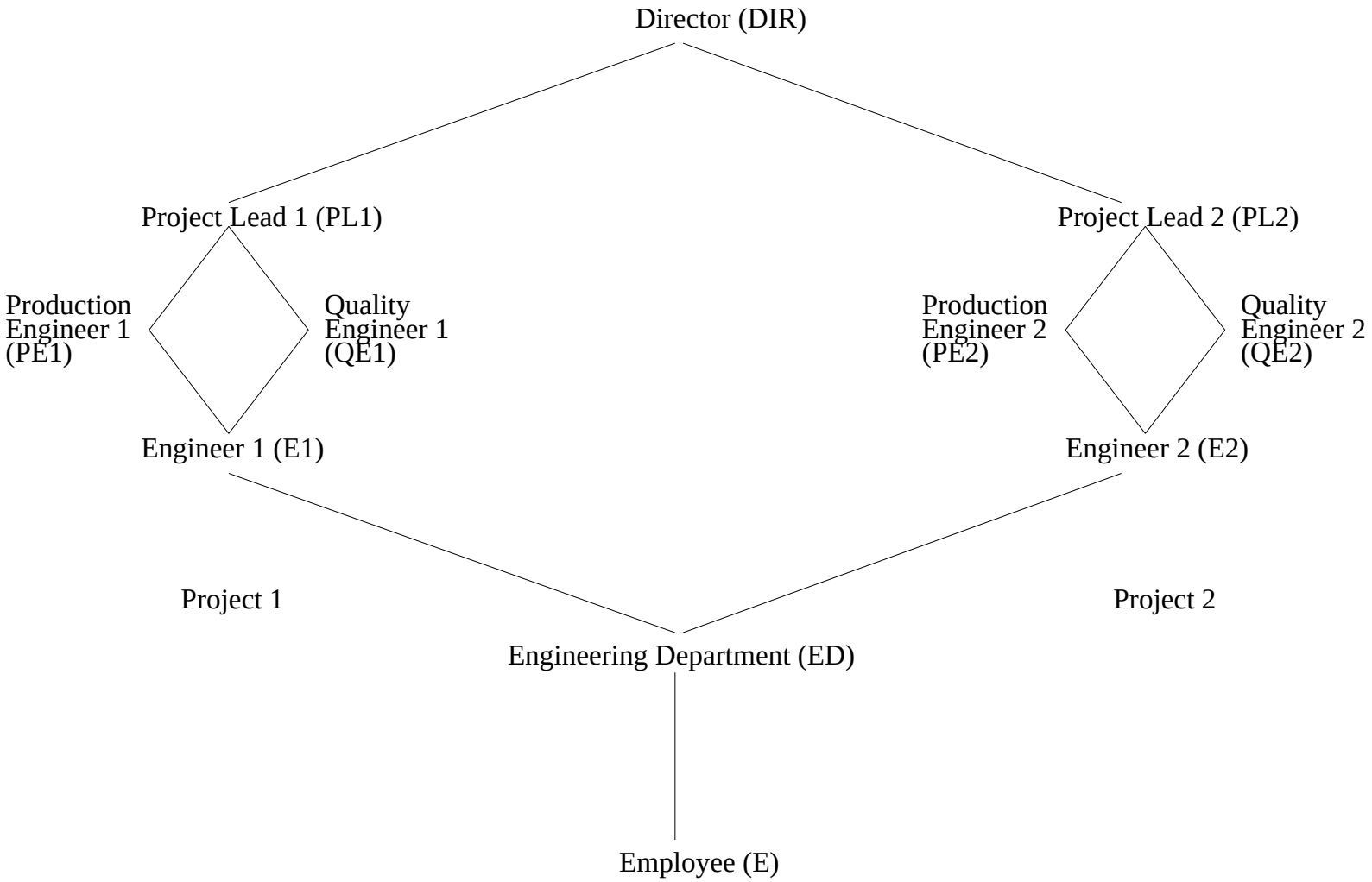
4. Enable enforcement of constraints on *PA* relation by security administration tools.

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

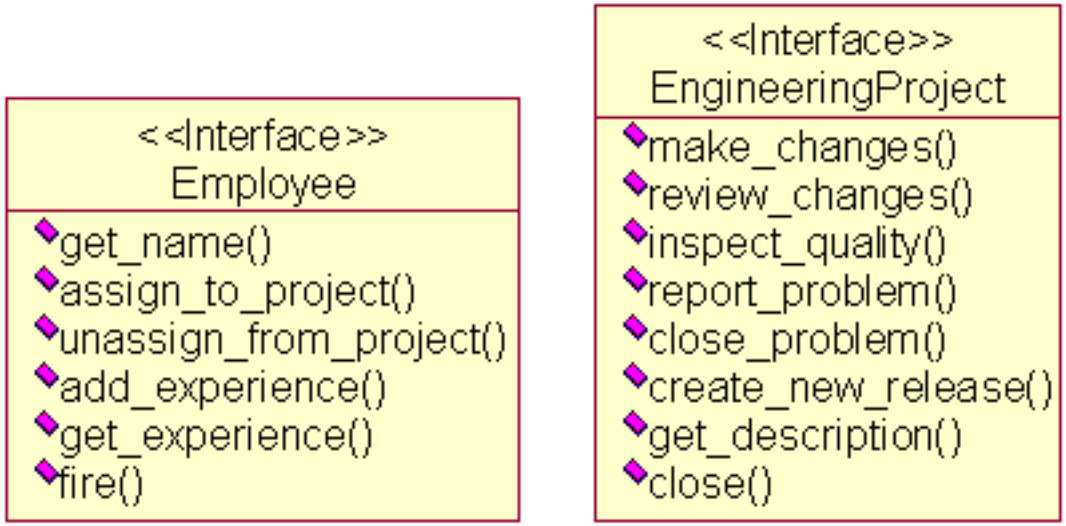
To support RBAC_3:

1. Implement RBAC_1
2. Implement RBAC_2.
3. Implement possible additional constraints on the role hierarchy.

Example Role Hierarchy



Interfaces


<<Interface>>
Employee
get_name()
assign_to_project()
unassign_from_project()
add_experience()
get_experience()
fire()
<<Interface>>
EngineeringProject
make_changes()
review_changes()
inspect_quality()
report_problem()
close_problem()
create_new_release()
get_description()
close()

Hypothetical Access Control Policies

1. Only colleagues can lookup employee experience.

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

3. Engineers working on the projects can make changes and review changes.

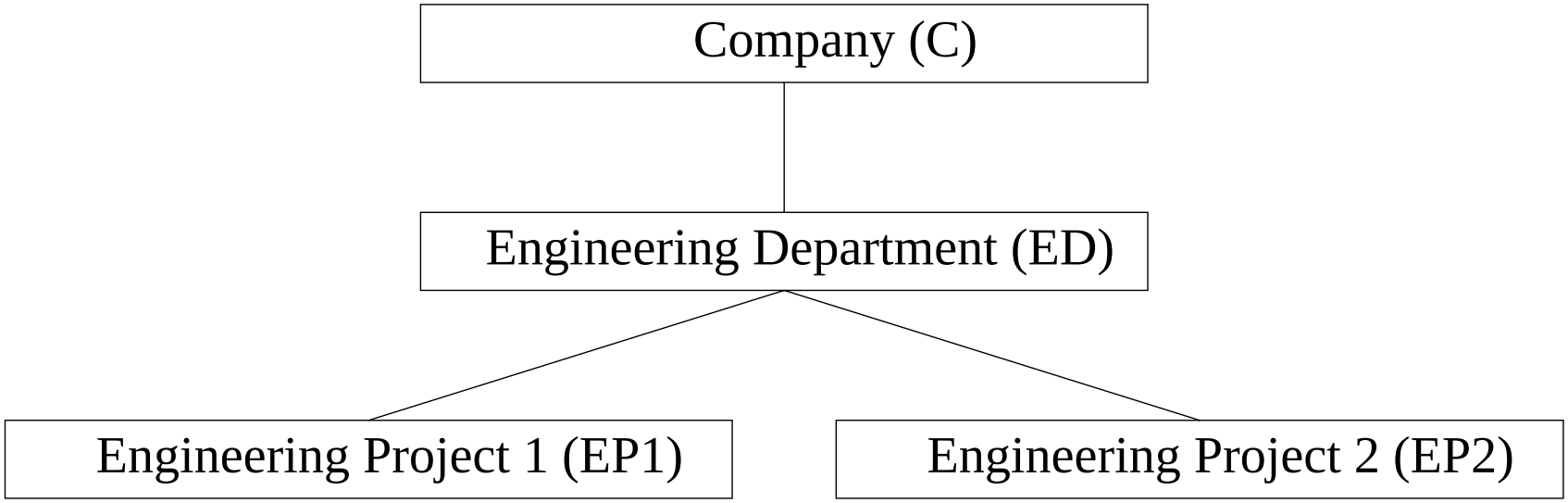
4. Quality engineers can inspect their project quality.

5. Production engineers can create new releases.

6. Project leaders can close problems and add experience to the records of the employees in the project.

7. The director can manage employees ([un]assign from/to projects and fire) and close engineering projects.

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}.

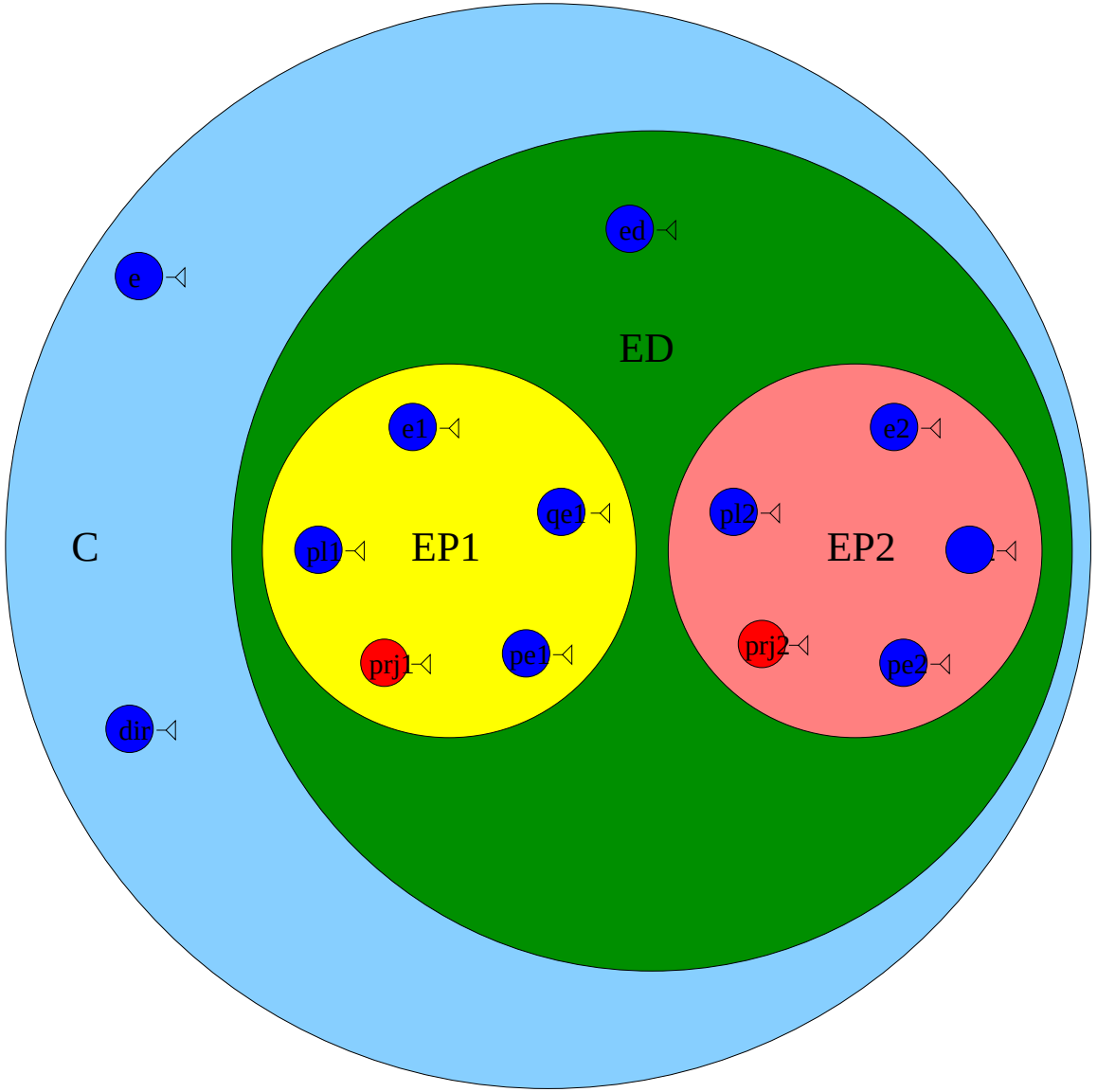
$\boldsymbol{R}$ = {gn, atp, ufp, ae, ge, f, mc, rc, iq, rp, cp, cnr, gd, c}.

$\boldsymbol{D}$ = {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	-	-	-

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.