Relationship-Based Access Control (ReBAC)

Prof. Ravi Sandhu
Executive Director and Endowed Chair

Lecture 6

ravi.utsa@gmail.com
www.profsandhu.com

Access Control

Fixed policy

Discretionary Access Control (DAC), 1970

Mandatory Access Control (MAC), 1970

Role Based Access Control (RBAC), 1995

Attribute Based Access Control (ABAC), ????

Flexible policy

Access Control

Fixed policy

Discretionary Access Control (DAC), 1970

Mandatory Access Control (MAC), 1970

Relationship Based Access Control (ReBAC), 2008

Role Based Access Control (RBAC), 1995

Attribute Based Access Control (ABAC), ????

Flexible policy

Access Control

Fixed policy

Discretionary Access Control (DAC), 1970

Mandatory Access Control (MAC), 1970

Relationship Based Access Control (ReBAC), 2008

Role Based Access Control (RBAC), 1995

Attribute Based Access Control (ABAC), ????

Flexible policy

ReBAC Models

Online Social Networks (OSNs)

- Social graph is modeled as a directed labeled simple graph $G=<U, E, \Sigma>$
  - Nodes U as users
  - Edges E as relationships
  - $\Sigma=\{\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_n^{-1}\}$

  as relationship types supported

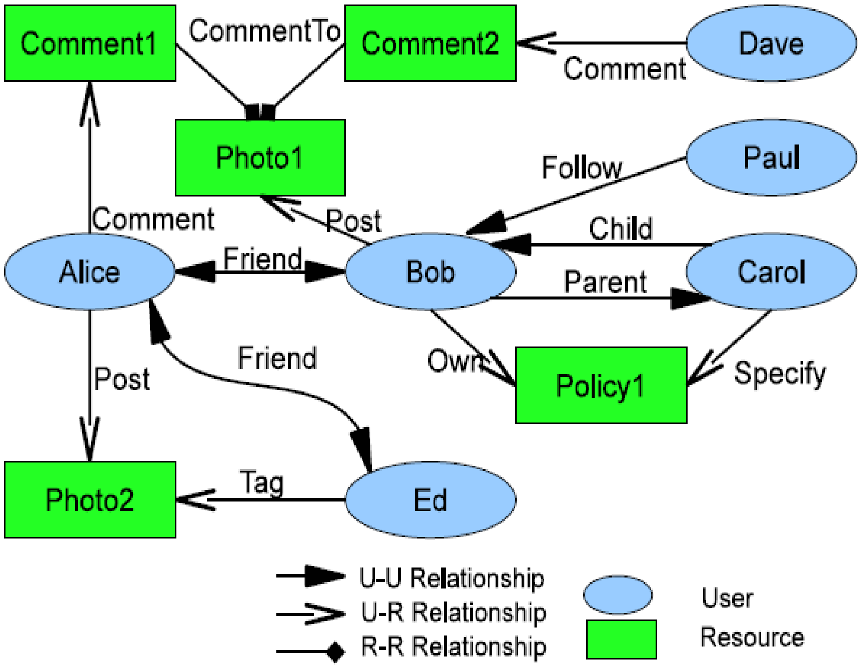
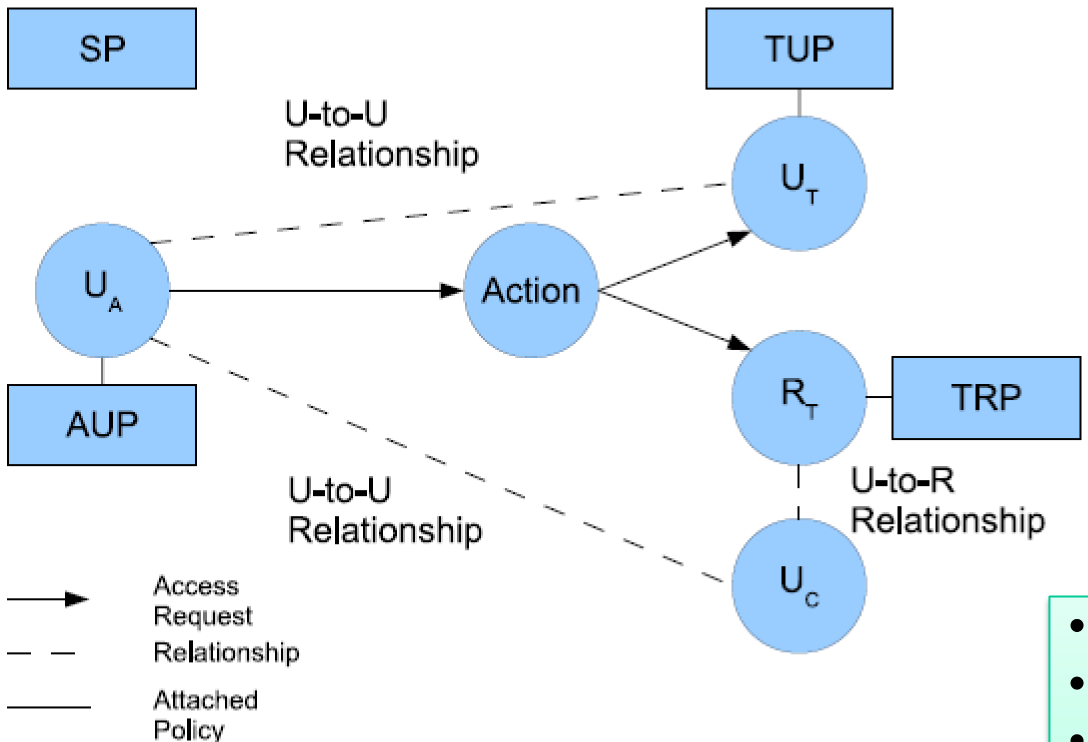


Fig. 3. A Sample Social Graph

Access Control in OSNs

- Policy Individualization
  - Users define their own privacy and activity preferences
  - Related users can configure policies too
  - Collectively used by the system for control decision
- User and Resource as a Target
  - e.g., poke, messaging, friendship invitation
- User Policies for Outgoing and Incoming Actions
  - User can be either requester or target of activity
  - Allows control on 1) activities w/o knowing a particular resource and 2) activities against the user w/o knowing a particular access requestor
  - e.g., block notification of friend's activities; restrict from viewing violent contents

U2U ReBAC (UURAC) Model



U_A: Accessing User
U_T: Target User
U_C: Controlling User
R_T: Target Resource
AUP: Accessing User Policy
TUP: Target User Policy
TRP: Target Resource Policy
SP: System Policy

- Policy Individualization
- User and Resource as a Target
- Separation of user policies for incoming and outgoing actions
- Regular Expression based path pattern w/ max hopcounts (e.g., <u_a, (f*c,3)>)

Access Request and Evaluation

- Access Request <u_a, *action*, *target*>
  - u_a tries to perform *action* on *target*
  - Target can be either user u_t or resource r_t

- Policies and Relationships used for Access Evaluation
  - When u_a requests to access a user u_t
    - u_a's AUP, u_t's TUP, SP
    - U2U relationships between u_a and u_t
  - When u_a requests to access a resource r_t
    - u_a's AUP, r_t's TRP, SP
    - U2U relationships between u_a and u_c

Policy Representations

Accessing User Policy	$< action, (start, path\ rule) >$
Target User Policy	$< action^{-1}, (start, path\ rule) >$
Target Resource Policy	$< action^{-1}, u_c, (start, path\ rule) >$
System Policy for User	$< action, (start, path\ rule) >$
System Policy for Resource	$< action, (r.typename, r.typevalue), (start, path\ rule) >$

- *$action^{-1}$* in TUP and TRP is the passive form since it applies to the recipient of action
- TRP has an extra parameter u_c to specify the controlling user
  - U2U relationships between u_a and u_c
- SP does not differentiate the active and passive forms
- SP for resource needs *r.typename, r.typevalue* to refine the scope of the resource

Example

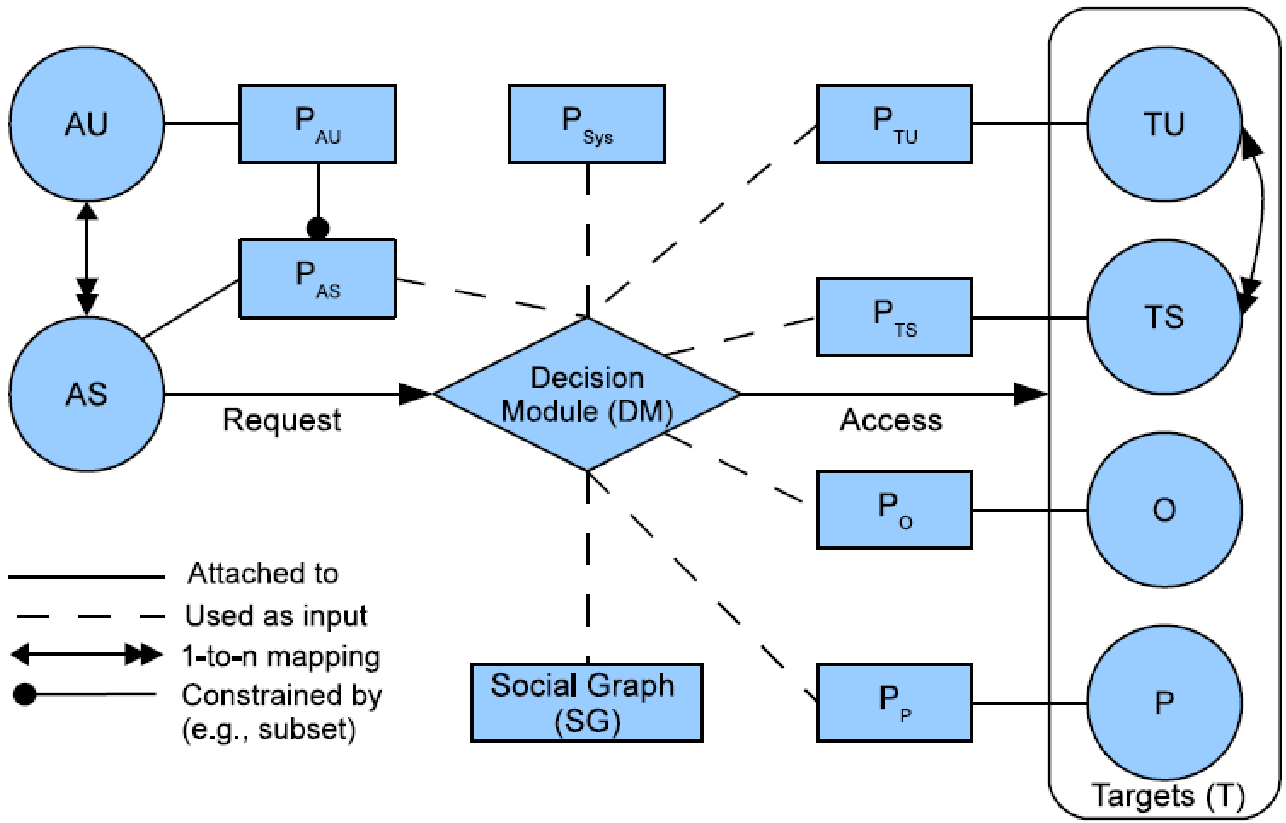
- Alice's policy P_{Alice}:
  - $< poke, \left(u_a, (f*,3)\right) >, < poke^{-1}, \left(u_t, (f,1)\right) >,$
  - $< read, \left(u_a, (\Sigma*,5)\right) >$
- Harry's policy P_{Harry}:
  - $< poke, \left(u_a, (cf*,5) \vee (f*,5)\right) >, < poke^{-1}, \left(u_t, (f*,2)\right) >$
- Policy of file2 P_{file2}:
  - $< read^{-1}, Harry, (uc, \neg(p+,2) >$
- System's policy P_{Sys}:
  - $< poke, \left(u_a, (\Sigma*,5)\right) >$
  - $< read, (filetype, photo), \left(u_a, (\Sigma*,5)\right) >$

- "Only Me"
  - $< poke, \left(u_a, (\emptyset, 0)\right) >$ says that ua can only poke herself
  - $< poke^{-1}, \left(u_t, (\emptyset, 0)\right) >$ specifies that ut can only be poked by herself
- The Use of Negation Notation
  - $(fffc \wedge \neg fc)$ allows the coworkers of the user's distant friends to see, while keeping away the coworkers of the user's direct friends

Beyond U2U Relationships

- There are various types of relationships between users and resources in addition to U2U relationships and ownership
  - e.g., share, like, comment, tag, etc
- U2U, U2R and R2R
- U2R further enables relationship and policy administration

U2U, U2R & R2R ReBAC (URRAC) Model



AU: Accessing User
AS: Accessing Session
TU: Target User
TS: Target Session
O: Object
P: Policy
P_{AU}: Accessing User Policy
P_{AS}: Accessing Session Policy
P_{TU}: Target User Policy
P_{TS}: Target Session Policy
P_{O}: Object Policy
P_{P}: Policy for Policy
P_{Sys}: System Policy

Differences with UURAC

- Access Request
  - (s, act, T) where T may contain multiple objects
- Policy Administration
- User-session Distinction
- Hopcount Skipping
  - Local hopcount stated inside “[[]]” will not be counted in global hopcount.
  - E.g., “([f*,3][[c*, 2]],3)”, the local hopcount 2 for c* does not apply to the global hopcount 3, thus allowing f* to have up to 3 hops.

Policy Conflict Resolution

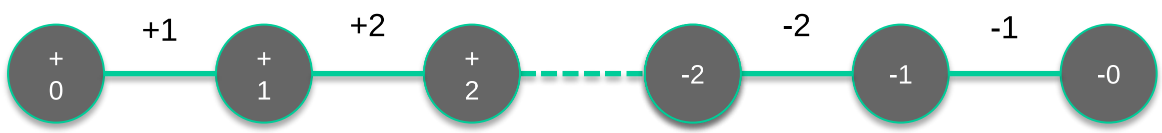
- System-defined conflict resolution for potential conflicts among user-specified policies
- Disjunctive, conjunctive and prioritized order between relationship types
  - <share-1, (own ∨ tag ∨ share)>
  - <read-1, (own ∧ tag)>
  - <friend_request, (parent > @)>

Beyond Relationships

- ReBAC usually relies on type, depth, or strength of relationships, but cannot express more complicated topological information
- ReBAC lacks support for attributes of users, resources, and relationships
- Useful examples include common friends, duration of friendship, minimum age, etc.

Attribute-based Policy

- $<quantifier, f(ATTR(N), ATTR(E)), count \geq i>$

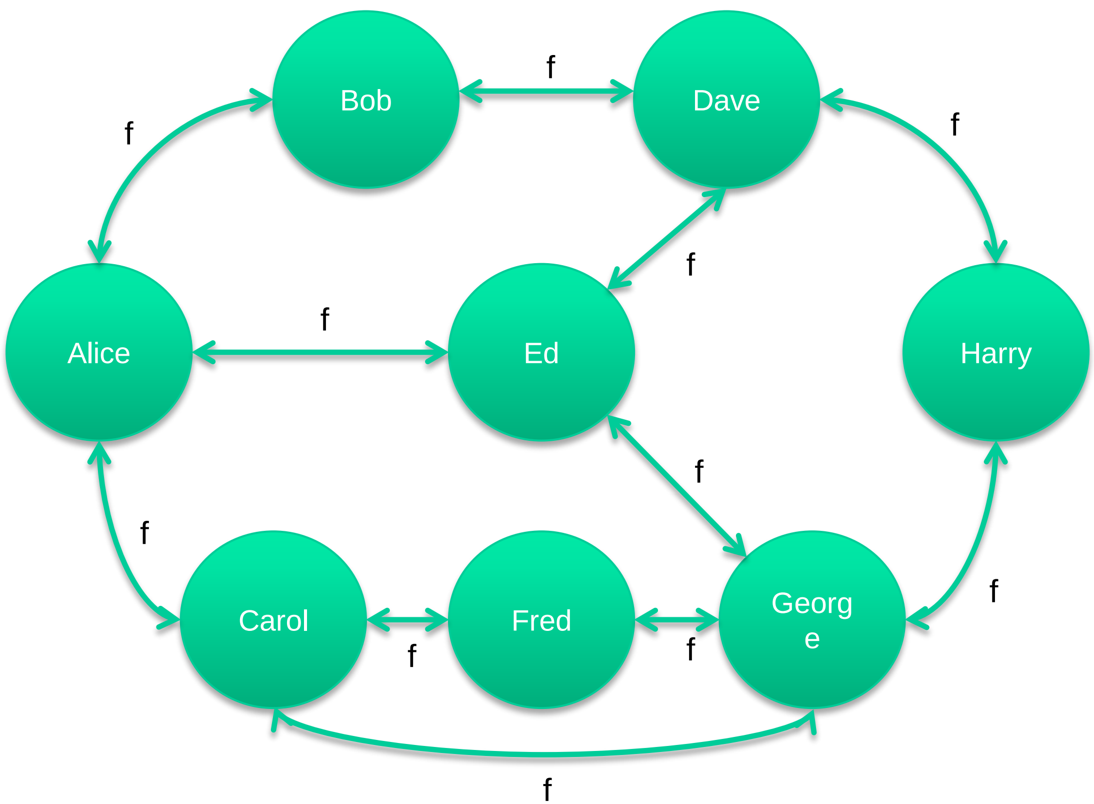


∀[+1, -2], age(u) > 18
∃[+1, -1], weight(e) > 0.5
∃{+1, +2, -1}, gender(u) = “male”

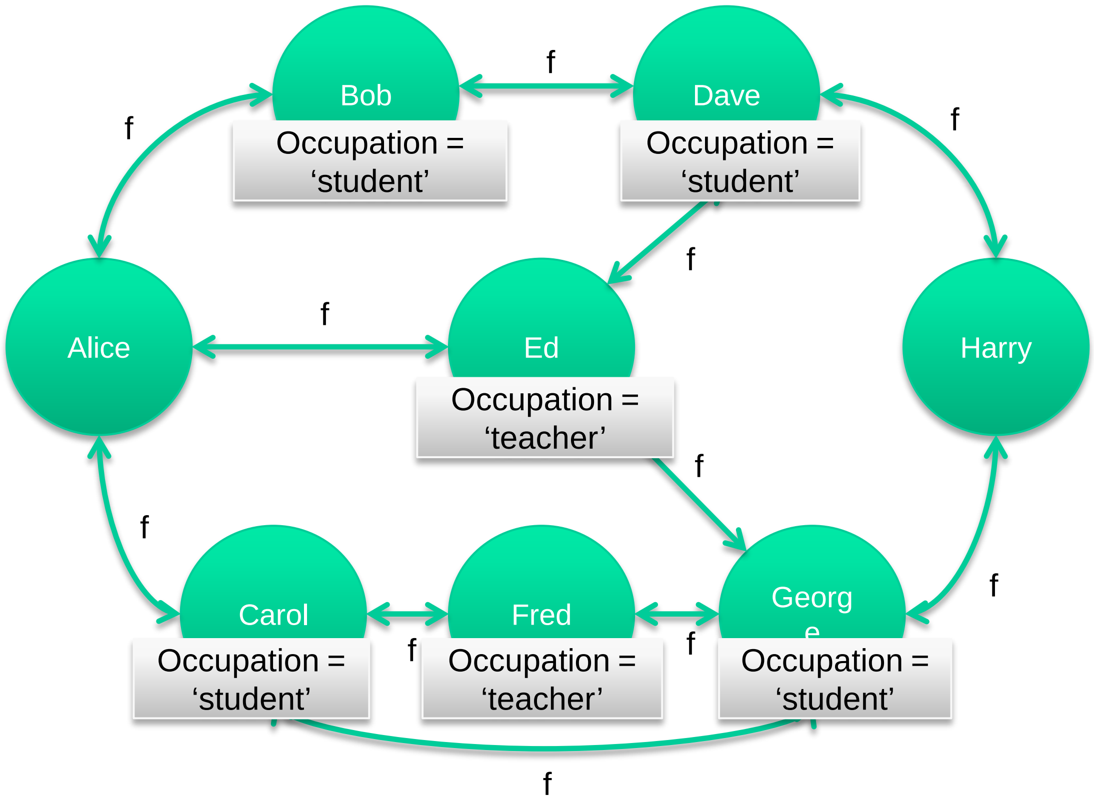
Attribute-based Policy

- Node attributes
  - Define user's identity and characteristics: e.g., name, age, gender, etc.
- Edge attributes
  - Describe the characteristics of the relationship: e.g., weight, type, duration, etc.
- Count attributes
  - Occurrence requirements for the attribute-based path specification, specifying the minimum

Example: No Attributes

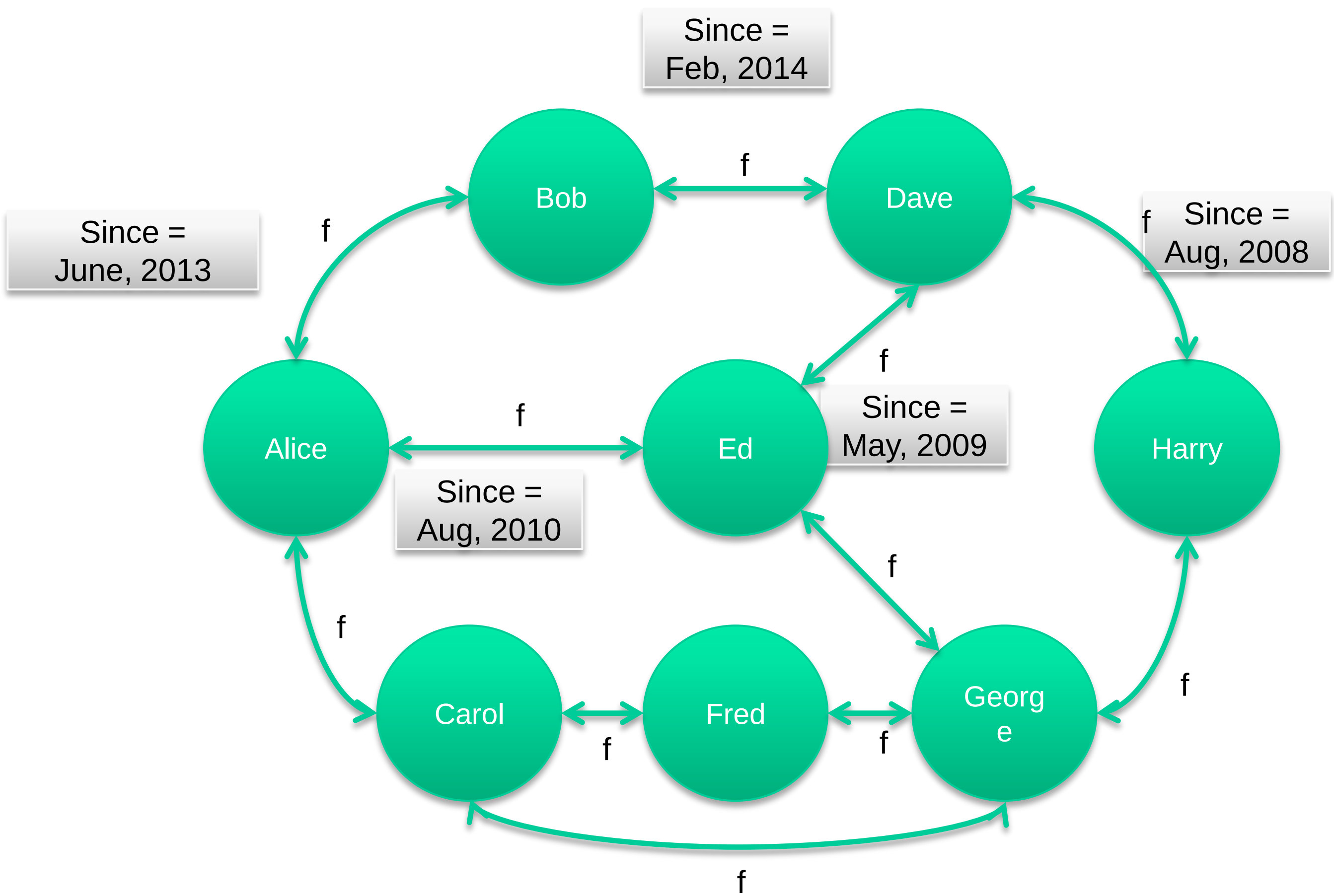


Example: Node Attributes



$<$access, (u_a, ((f*, 4): ∃[+1, -1], occupation = 'student', count ≥ 3)))$>$

Example: Edge Attributes

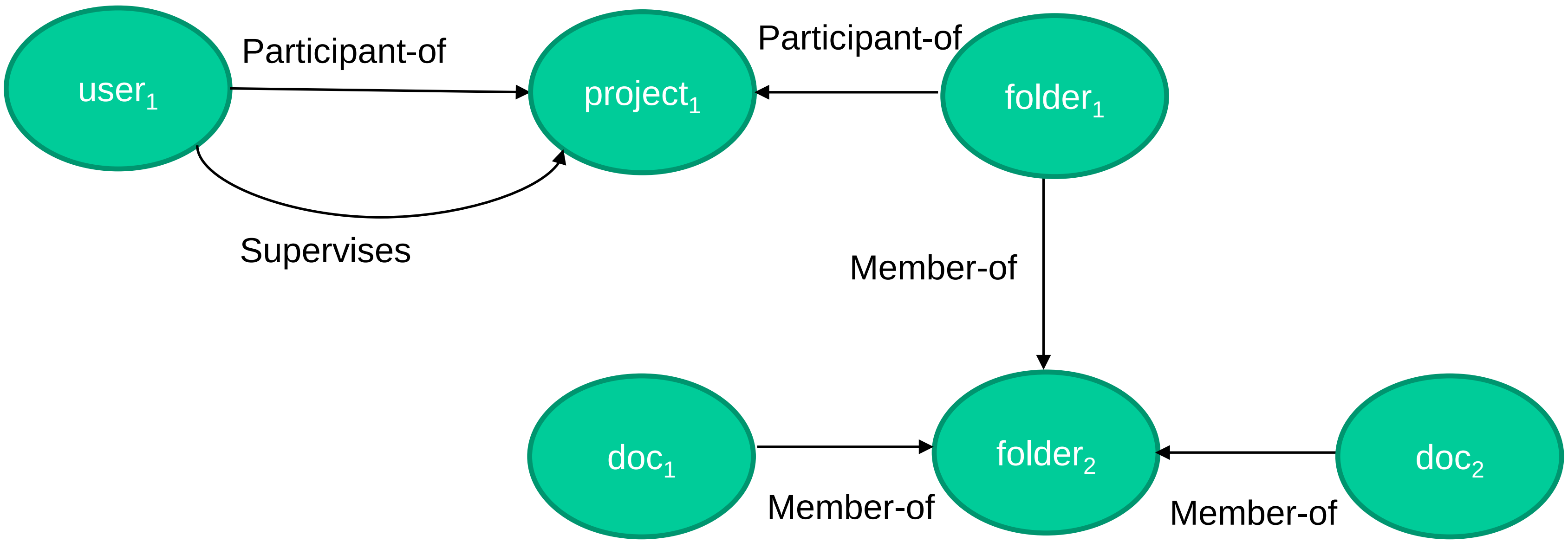


<read, Photo1, (u_a, ((f*, 3): $\forall$[+1, -1], duration $\geq$ 3 month, _)))>

ReBAC Models
Object-to-Object

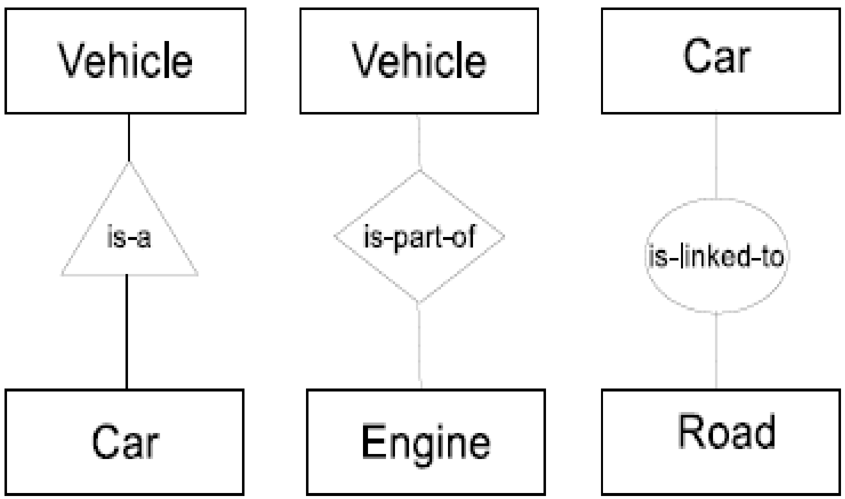
Object Relationships in ReBAC

- ReBAC for OSN generally considers only user to user relationship
- OSN has very specific types of resources – photos, notes, comments, which are strongly tied to users.
- Even though some ReBAC models consider general computing systems beyond OSNs they still need users/subjects existence in relationship graph.

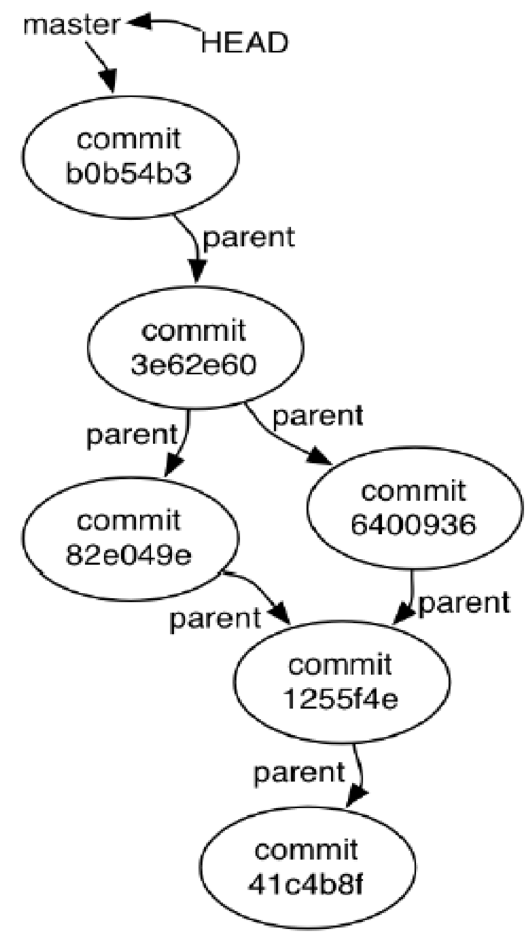


A sample Relationship Graph for Organizational Environment
[RPPM, Crampton et al. ,2014]

Existence of Object Relationship Independent of User



Object Relationship in Object –Oriented System (Inheritance, Composition and Association)



History of a Git Project (Version Control System) is a DAG

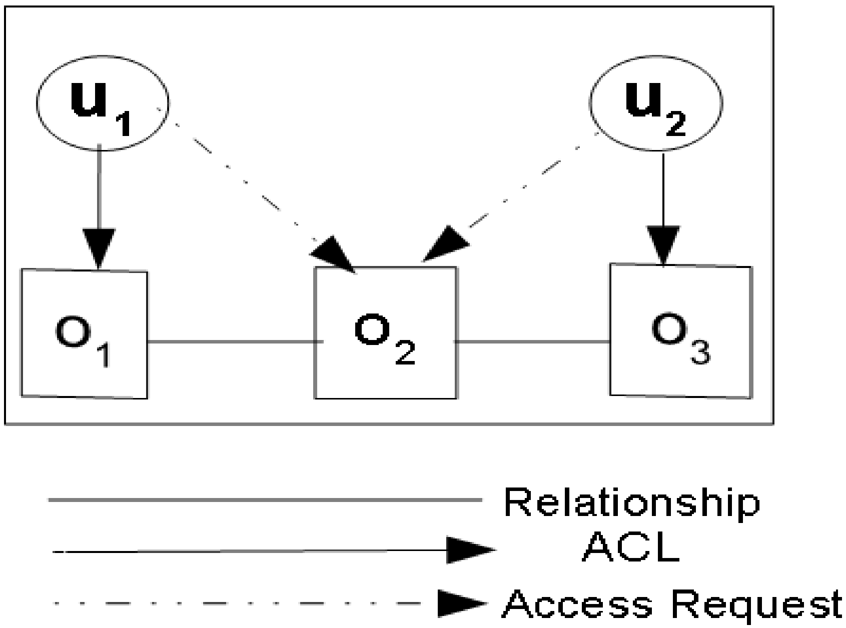
Limitations of Existing ReBAC Models

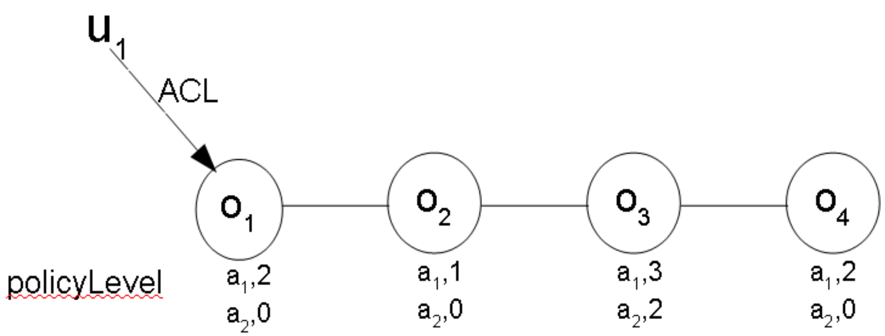
- Cannot configure relationship between objects independent of user.
- Cannot express authorization policy solely considering object relationship.

How the model would look like?

Object to Object Relationship Based Access Control

Policy Level Example

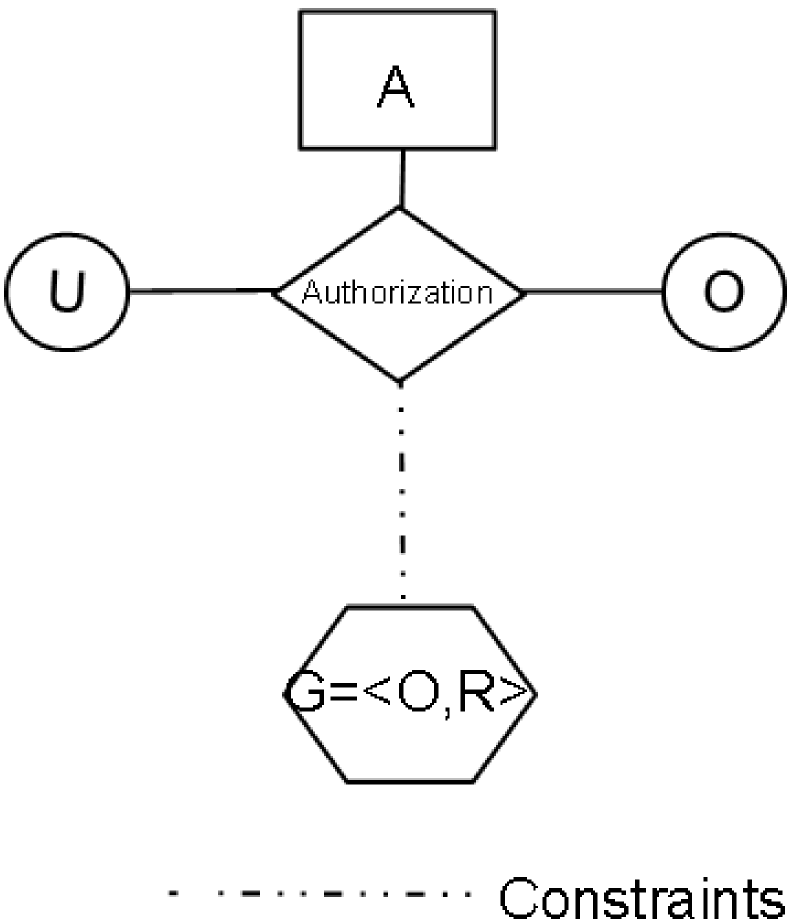




$ACL(o_1) = \{u_1\}$
$ACL(o_2) = \{\}$
$ACL(o_3) = \{u_2\}$

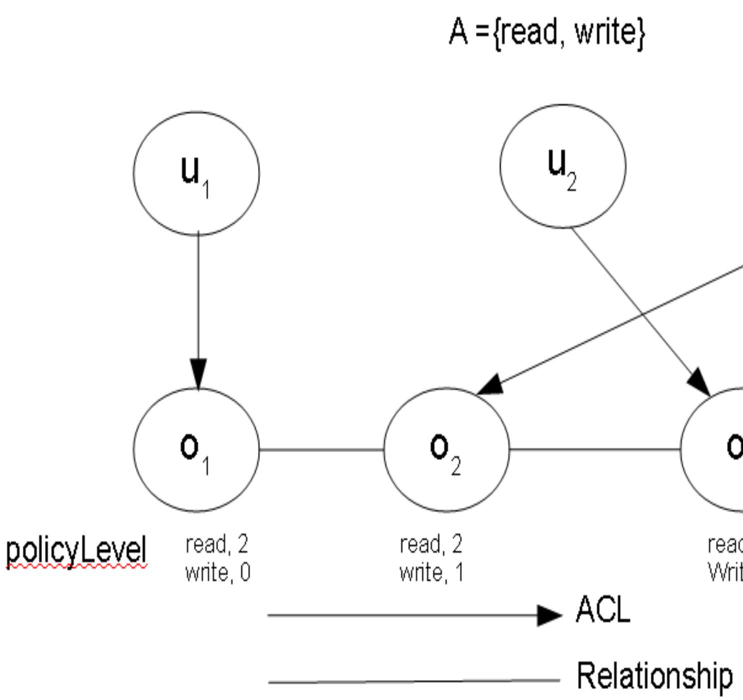
$policyLevel(a_1, o_1) = 2$
$policyLevel(a_2, o_1) = 0$
$policyLevel(a_1, o_2) = 1$
$policyLevel(a_2, o_2) = 0$
$policyLevel(a_1, o_3) = 3$
$policyLevel(a_2, o_3) = 2$
$policyLevel(a_1, o_4) = 2$
$policyLevel(a_2, o_4) = 0$

OOReBAC: Model Components and Definition



- U is a set of users
- O is a set of objects
- $R \subseteq \{z \mid z \subset O \wedge |z| = 2\}$
- $G = \langle O, R\rangle$ is an undirected relationship graph with vertices O and edges R
- A is a set of actions
- $P^i(o_1) = \{ o_2 \mid$ there exists a simple path of length p in graph G from o_1 to $o_2 \}$
- policyLevel: $O \times A \rightarrow \mathbb{N}$
- ACL: $O \rightarrow 2^U$ which returns the Access control List of a particular object.
- There is a single policy configuration point. Authorization Policy. for each action $a \in A$, $Authz_a(u{:}U, o{:}O)$ is a boolean function which returns true or false and u and o are formal parameters.
- Authorization Policy Language:
  Each action "a" has a single authorization policy $Authz_a(u{:}U, o{:}O)$ specified using the following language.
  $\phi ::= u \in PATH_i$
  $PATH_i ::= ACL(P^0(o)) \cup \ldots \cup ACL(P^i(o))$ where $i = \min(|O| - 1, \text{policyLevel}(a, o))$
  where for any set X, $ACL(X) = \bigcup_{x \in X} ACL(x)$

OOReBAC: An Example



Sequence of operations and its outcome:

- $U = \{u_1, u_2, u_3\}$
- $O = \{o_1, o_2, o_3, o_4\}$
- $R = \{\{o_1, o_2\}, \{o_2, o_3\}, \{o_3, o_4\}\}$
- $ACL(o_1) = \{u_1\}$
  $ACL(o_2) = \{u_3\}$
  $ACL(o_3) = \{u_2\}$
  $ACL(o_4) = \{u_3\}$
- policyLevel(read, o_1) = 2
  policyLevel(write, o_1) = 0
  policyLevel(read, o_2)= 2
  policyLevel(write,o_2) = 1
  policyLevel(read,o_3) = 0
  policyLevel(write,o_3) = 0
  policyLevel(read,o_4) = 2
  policyLevel(write,o_4) = 1

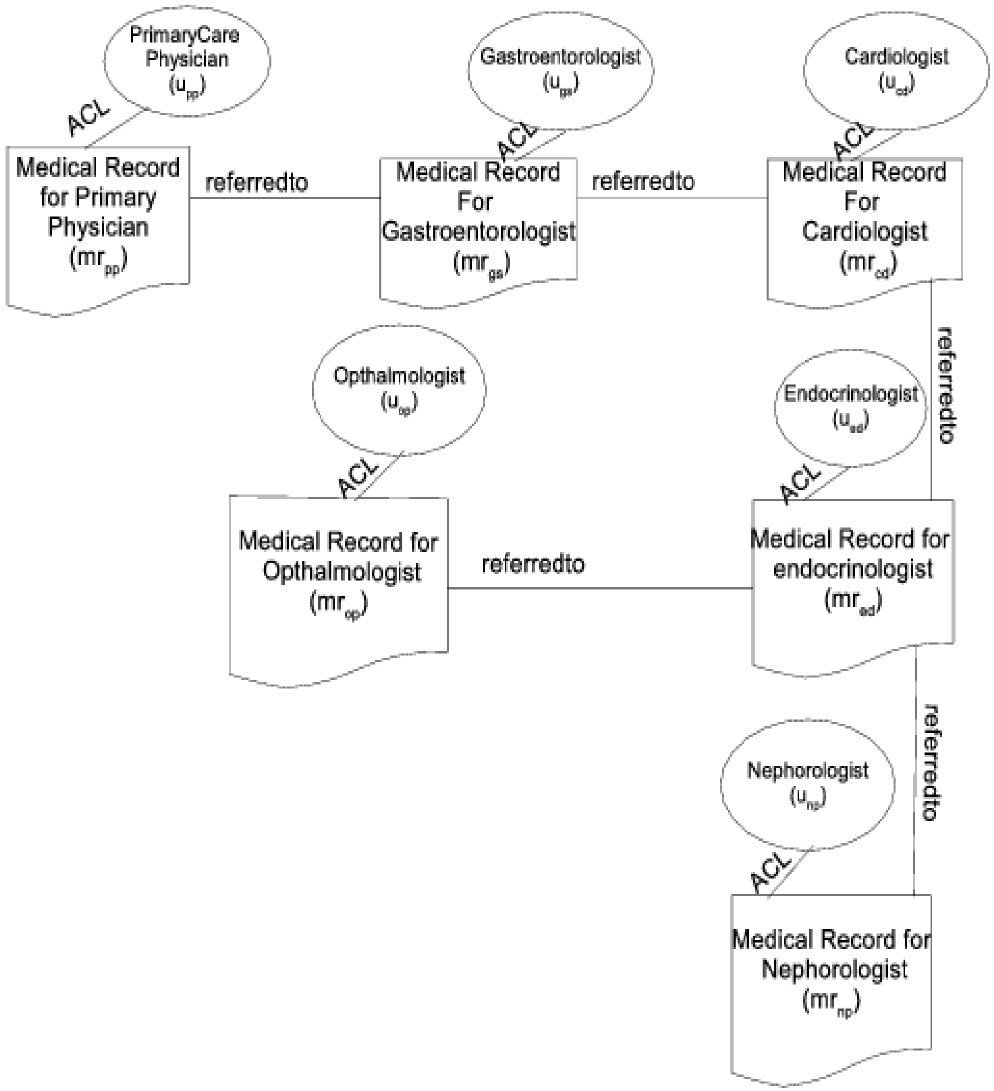
Configuration:

- $A = \{read, write\}$
- $Authz_{read}(u{:}U, o{:}O) \equiv u \in P^{policyLevel(read,o)}$
- $Authz_{write}(u{:}U, o{:}O) \equiv u \in P^{policyLevel(write,o)}$

Sequence of operations and its outcome:

- read(u_1,o_3), write(u_1,o_3) are denied
- read(u_2, o_1) is allowed, write(u_2, o_1) is denied
- read(u_1,o_4, write(u_1,o_4) are denied

OOReBAC: Application



An OOReBAC Instantiation

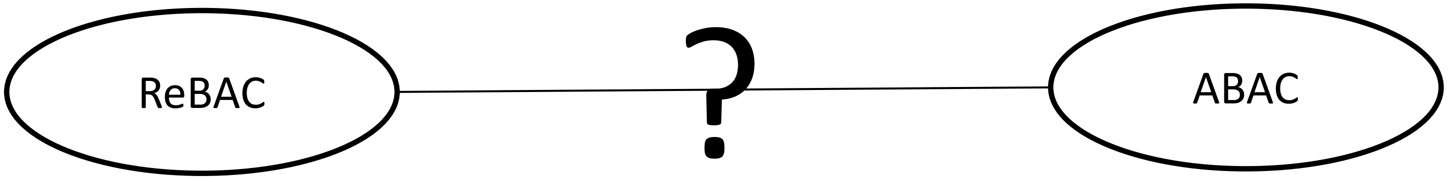
- $U = \{ u_{pp}, u_{gs}, u_{cd}, u_{op}, u_{ed}, u_{np} \}$
- $O = \{ mr_{pp}, mr_{gs}, mr_{cd}, mr_{op}, mr_{ed}, mr_{np} \}$
- $R = \{\{mr_{pp}, mr_{gs}\}, \{mr_{gs}, mr_{cd}\}, \{mr_{cd}, mr_{ed}\}, \{mr_{op}\ mr_{ed}\},\{mr_{np}, mr_{ed}\}\}$
- $ACL(mr_{pp}) = \{u_{pp}\}$,
  $ACL(mr_{gs}) = \{u_{gs}\}$,
  $ACL(mr_{cd}) = \{u_{cd}\}$,
  $ACL(mr_{op}) = \{u_{op}\}$,
  $ACL(mr_{ed}) = \{u_{ed}\}$,
  $ACL(mr_{np}) = \{u_{np}\}$
- Action $=\{$read, write$\}$
- $policyLevel(read,mr_{pp})=\infty$, $policyLevel(write,mr_{pp})=0$,
  $policyLevel(read,mr_{gs})=\infty$, $policyLevel(write,mr_{gs})=0$,
  $policyLevel(read,mr_{cd})=\infty$, $policyLevel(write,mr_{cd})=0$,
  $policyLevel(read,mr_{op})=\infty$, $policyLevel(write,mr_{op})=0$,
  $policyLevel(read,mr_{ed})=\infty$, $policyLevel(write,mr_{ed})=0$,
  $policyLevel(read,mr_{np})=\infty$, $policyLevel(write,mr_{np})=0$
- Authorization policy:
  $Authz_{read}(u,o) \equiv u \in P^{policyLevel(read,o)}$
  $Authz_{write}(u,o) \equiv u \in P^{policyLevel(write,o)}$

Sequence of Operations and Outcomes

1) $read(u_{np}, mr_{pp})$: authorized
2) $read(u_{cd}, mr_{np})$: authorized
3) $write(u_{np}, mr_{np})$: authorized
4) $write(u_{np}, mr_{pp})$: denied
5) $write(u_{np}, mr_{pp})$: denied

ABAC-ReBAC Comparison

ReBAC Vs. ABAC



- Are they Comparable ?
- Can Attributes Express Relationships?
- Can ReBAC Configure ABAC? Vice versa?
- Do they have equal expressive power?

If not

- Which one is more expressive?

Attribute Types

1. Attribute Value Structure
   - Atomic-valued or Single-valued Attribute (e.g. gender)
   - Set-valued or Multi-valued Attribute (e.g. phoneNumber)
   - Structured Attribute (e.g person-Info (name, age, phoneNumber))
2. Attribute Value Scope
   - Entity Attribute (e.g. friend)
   - Non-entity Attribute (e.g. age)
3. Boundedness of attribute range
   - Finite Domain Attribute (e.g. gender)
   - Infinite Domain Attribute (e.g. time)
4. Attribute association
   - Contextual or Environmental Attribute (e.g. currentTime)
   - Meta Attribute (e.g. role(user) = manager , task(manager) = supervise)
5. Attribute mutability
   - Mutable Attribute
   - Immutable Attribute

Attribute Function Composition

$$f : X \rightarrow Y$$

$$g : Y \rightarrow Z$$

$$x \in X \text{ , } g\big(f(x)\big) \in Z$$

Assumptions

- All non entity attribute are finite domain
- Entity attribute functions are partial functions defined on existing entities only
- Inner attribute function in an attribute function composition should always be entity attributes
- Structured attribute is a multivalued tuple of atomic or set-valued attributes. So it is more expressive than atomic or set-valued attribute.

ReBAC Classification

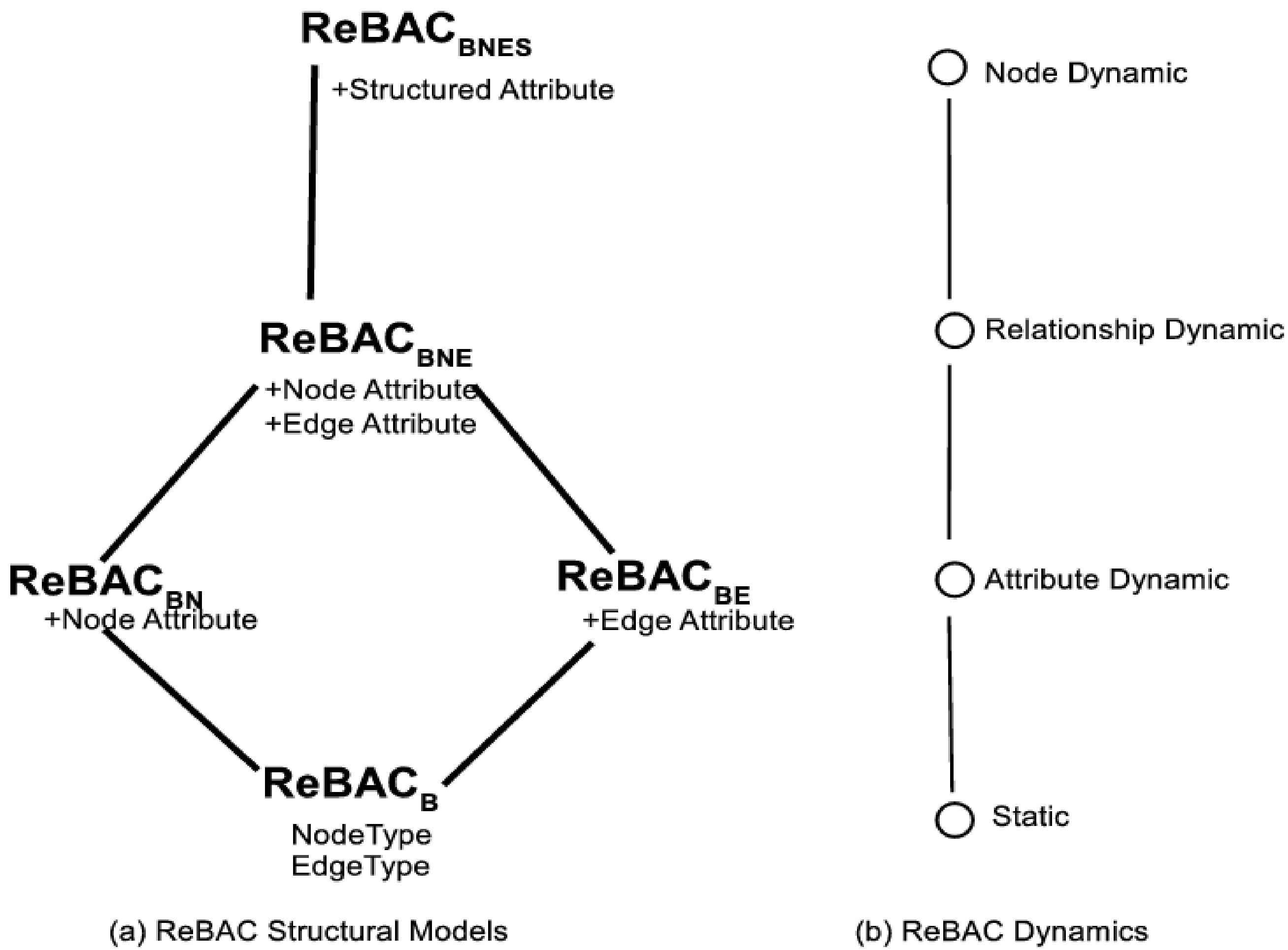


Figure 3.: ReBAC Framework

Example

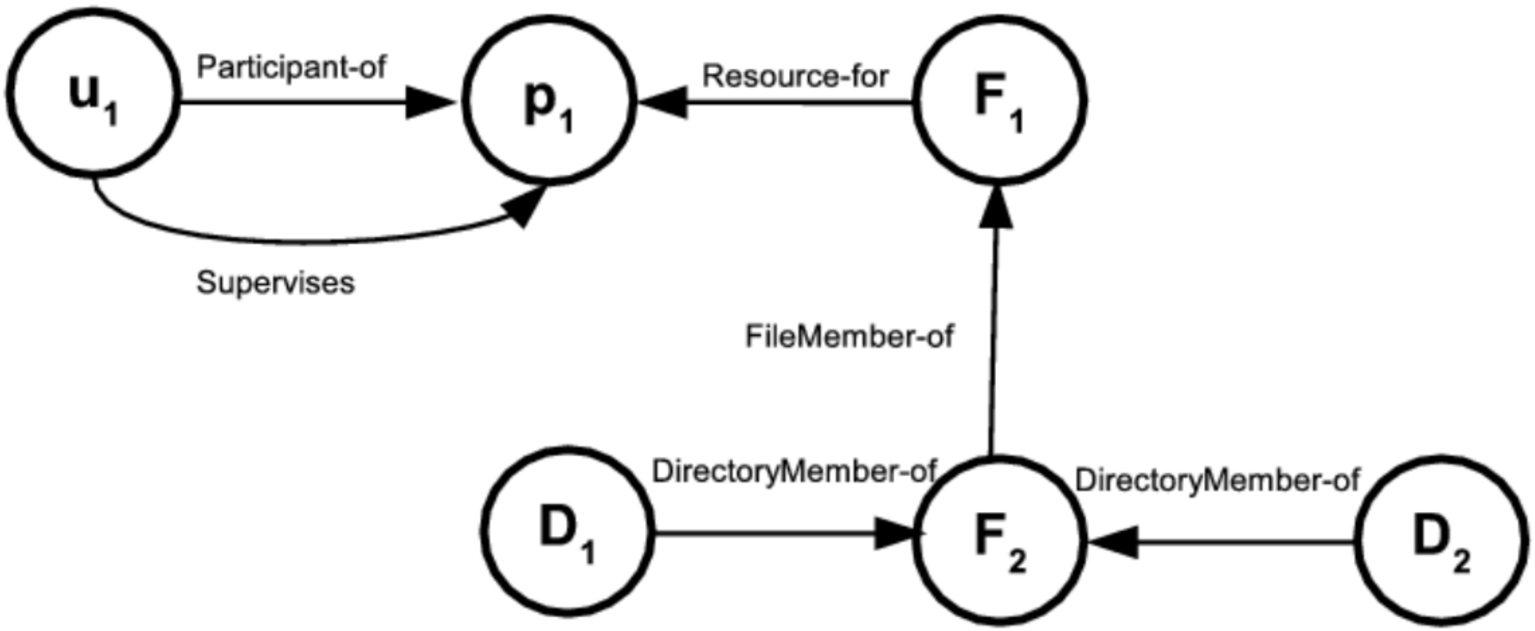


Figure 4.: A Simple Relationship Graph Expressible in $ReBAC_B$ [Crampton et al. 2014]

Example (Continued…)

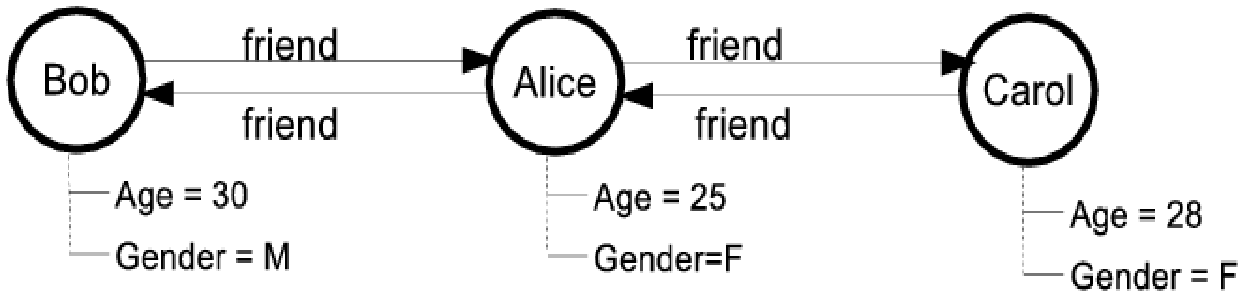


Figure 5: An Example of Node Attributes in Relationship Graph Expressible in $ReBAC_{BN}$

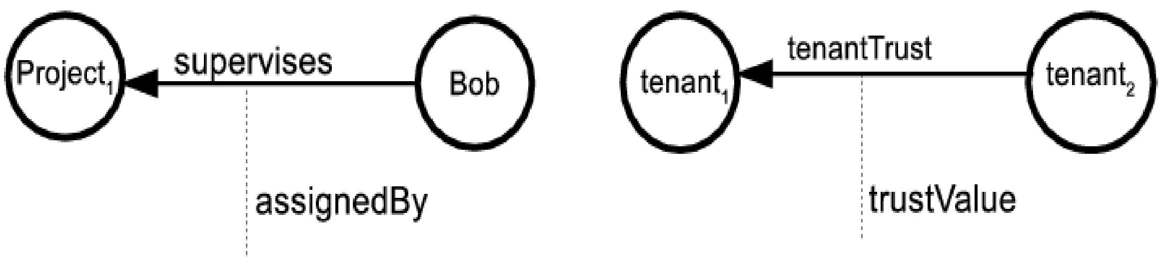
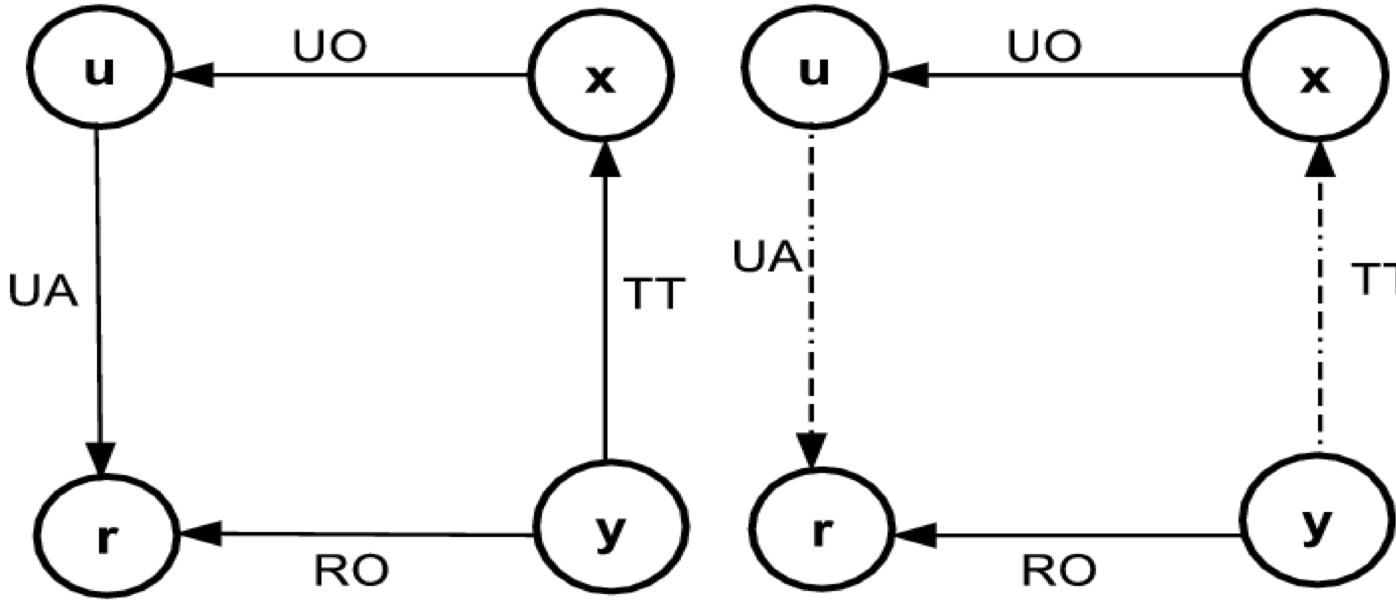


Figure 6: An Example of Edge Attributes in Relationship Graph Expressible in $ReBAC_{BE}$

Example (Continued…)



Structure Edge Attribute:
dependsOn

Sub Attributes of dependsON
Source Node
Target Node
RelationshipType

dependsOn (u,r,UA) = (y,x,TT)

Figure 7: An Example of Node Attributes in Relationship Graph Expressible in $\text{ReBAC}_{\text{BNES}}$ [Cheng et al. 2016]

ABAC Classification

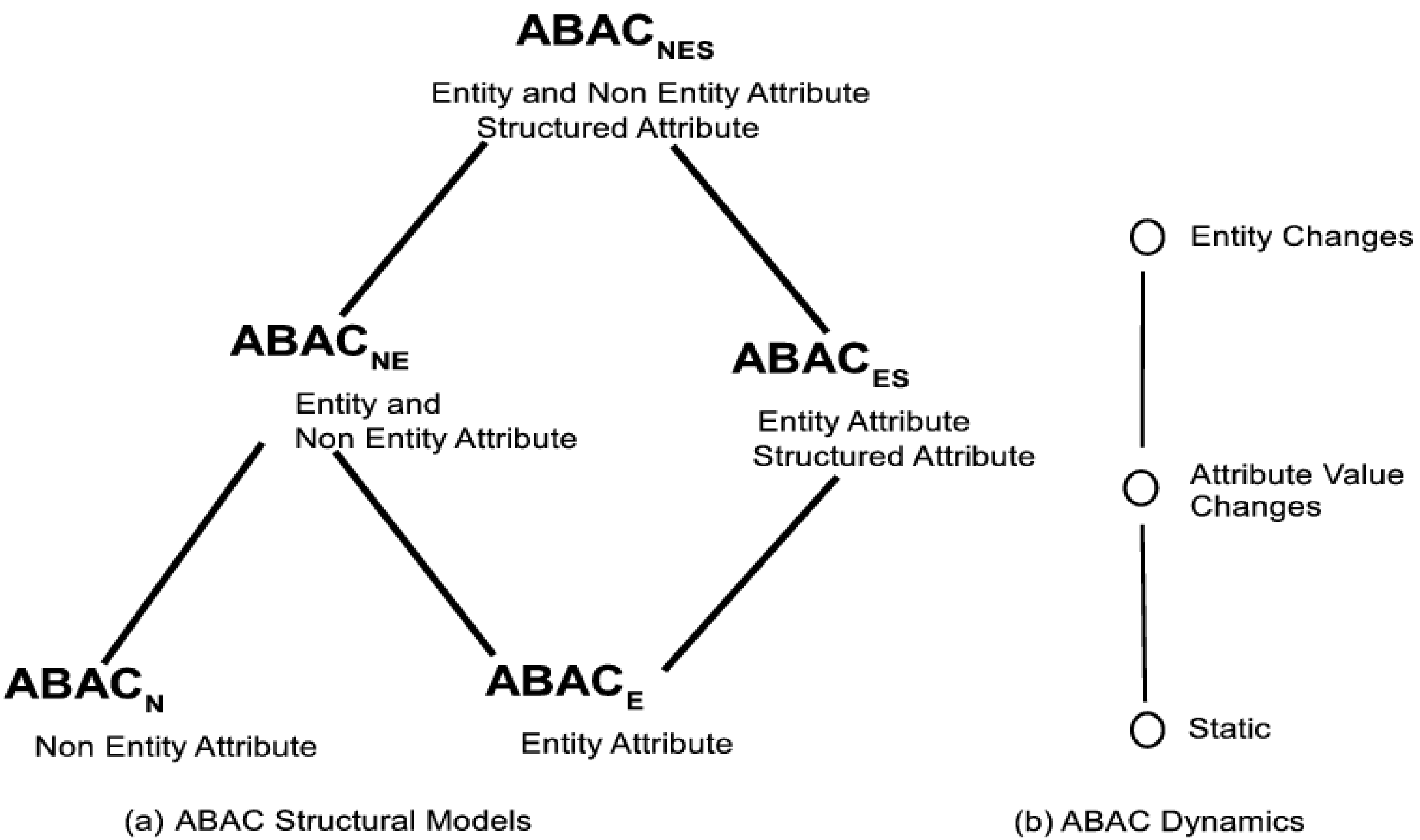
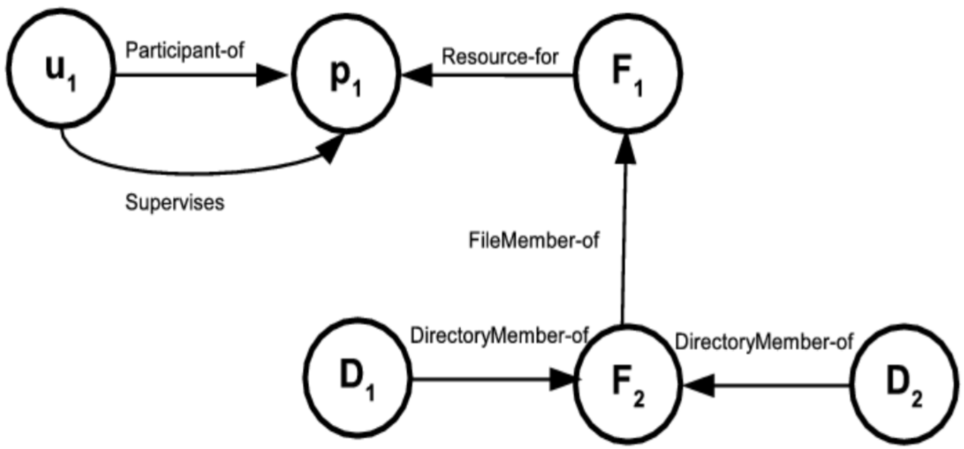


Figure 8: ABAC Framework

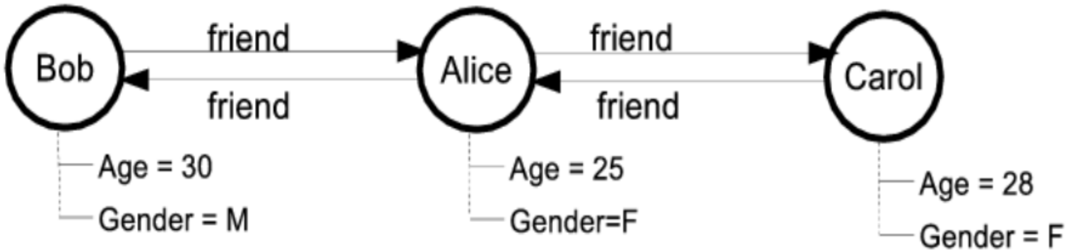
Expressing Relationship Graph with Attributes



- Entity types = {user, project, file , directory}
- Attributes:
  - User attributes ={Participant-of, Supervises}
  - File attributes = {Resource-for, FileMember-of}
  - Project attributes = {}
  - Directory attributes ={DirectoryMember-of}

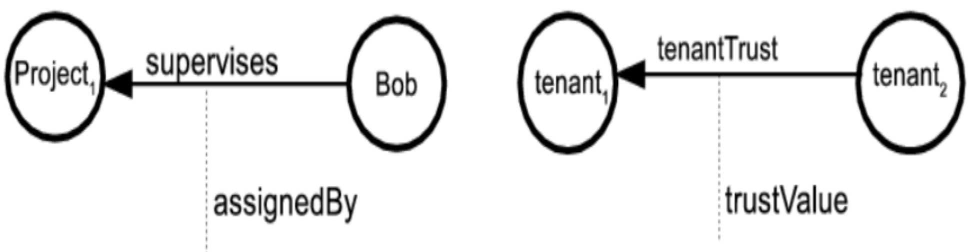
Relationship Graph in Figure 4 is Expressible with $ABAC_E$

Expressing Relationship Graph with Attributes (Continued…)



- entityType = {user}
- Attribute:
  - ❑ user's entity attribute ={friend}
  - ❑ User's Non Entity Attribute ={Name, Age, Gender}

Relationship Graph in Figure 5 is Expressible with $ABAC_E$

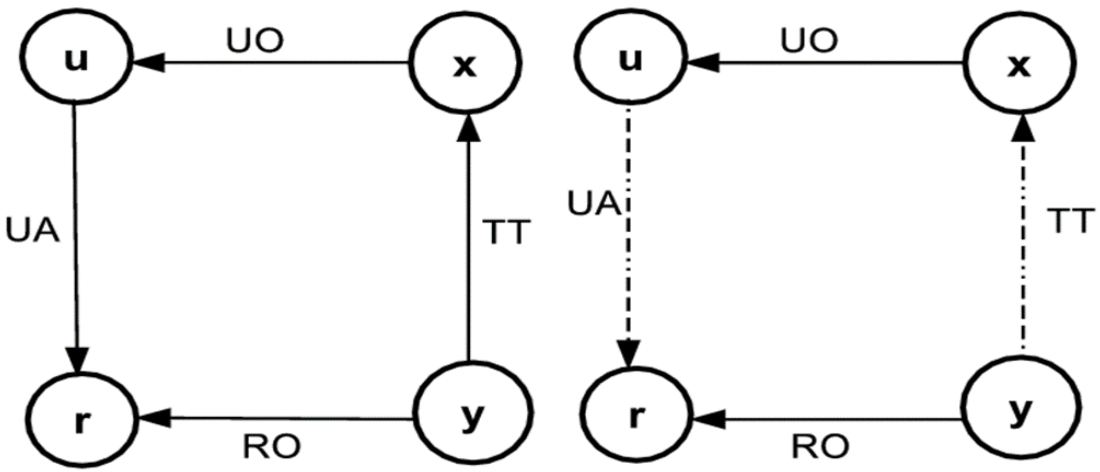


- entityType = {user, project, tenant}
- Attribute:
  - ❑ user's atomic entity attribute ={supervises}
  - ❑ User's structured entity Attribute ={assignedBy}

  e.g. assignedBy(Bob) = ("Project1", "supervises", "Alice")

Relationship Graph in Figure 6 is Expressible with $ABAC_{ES}$

Expressing Relationship Graph with Attributes (Continued…)



- Entity types: {user, tenant, role}
- Attribute:
  - User's atomic entity attribute: {UO,UA}
  - Users Structured Entity Attribute: {dependentEdge}

  dependentEdge(u) = ("r","UA", {(y,x,TT)})

Relationship Graph in Figure 7 is Expressible with $ABAC_{ES}$

Expressing Multilevel Relationship With Attributes

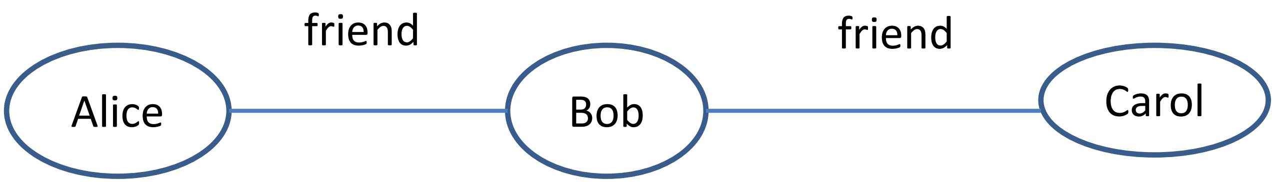


Figure 9. A simple Relationship Graph

Attribute Composition

- ❑ Needs one attribute: friend
- ❑ Policy Expression uses Attribute composition

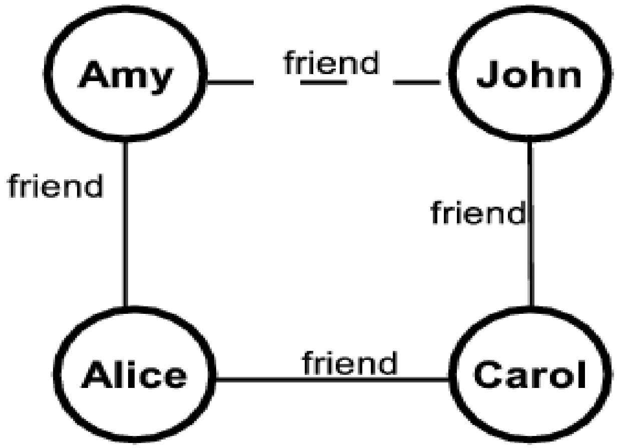
friend(Alice)={Bob}
friend(friend(Alice))={Carol}

Composite Attribute

- ❑ Needs two attribute
  1. friend
  2. friendoffriend
- ❑ Policy Expression uses direct attributes

friend(Alice) ={Bob}
friendoffriend(Alice)={Carol}

Example:



friend(Alice) = {Amy, Carol}
friendoffriend(Alice) = {John}

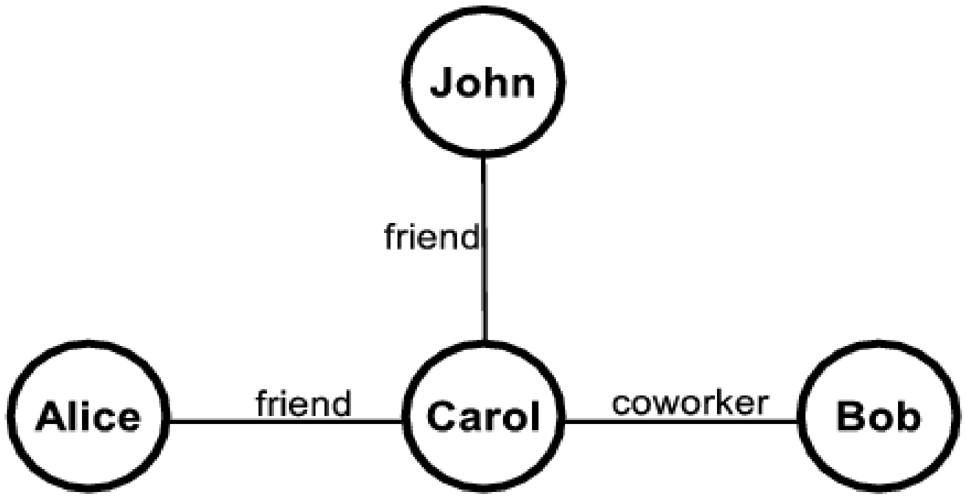
Figure 10. A simple Relationship Graph

If the friend relationship between Amy and John deleted

friendoffriend(Alice) = ?

Instead of keeping the end user as attribute value we have to keep the exact path information.

Example



Attribute Composition:

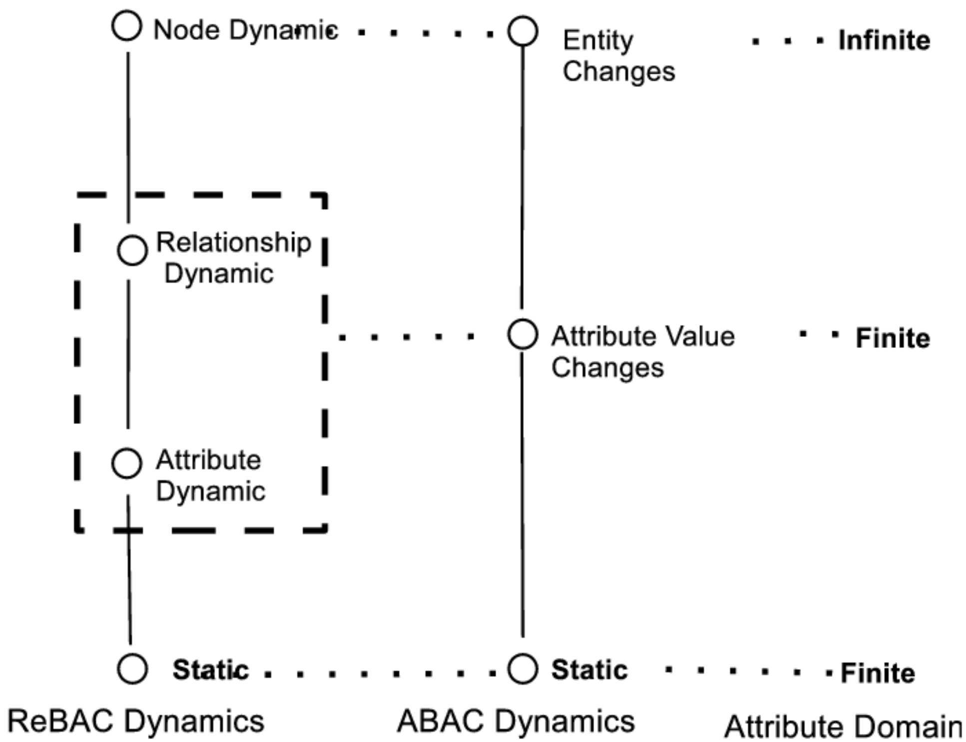
friend (“Alice”) = {“Carol”}
coworker (“Alice”) = { }
friend (friend(“Alice”)) = { “John”}
coworker(coworker(“Alice”)) = { }
friend (coworker(“Alice”)) = { }
coworker (friend(“Alice”)) = {“Bob”}

Composite Attribute:

friend (“Alice”) = {“Carol”}
coworker (“Alice”) = { }
friendOfFriend(“Alice”) = { “Carol.John”}
coworkerOfCoworker(“Alice”)) = { }
friendOfCoworker(“Alice”) = { }
coworkerOfFriend(“Alice”)) = {“Carol.Bob”}

Figure 12: Multilevel Relationship Expression with Attribute

Comparison: On Dynamics



$ABAC_X \equiv ReBAC_Y\ Means$

- Static and finite attribute domain $ABAC_X \equiv Static\ ReBAC_Y$
- $ABAC_X\ Attribute\ value\ changes\ with\ finite\ domain \equiv Relationship\ Dynamic\ ReBAC_Y$
- $ABAC_X\ with\ entity\ changes\ and\ infinite\ domin\ entity\ attribute \equiv node\ dynamic\ ReBAC_Y$

Figure 12: ReBAC Dynamics, ABAC Dynamics and Attribute Domain wise Comparison between ReBAC and ABAC

Comparison: Equivalent Structural Models for ReBAC and ABAC

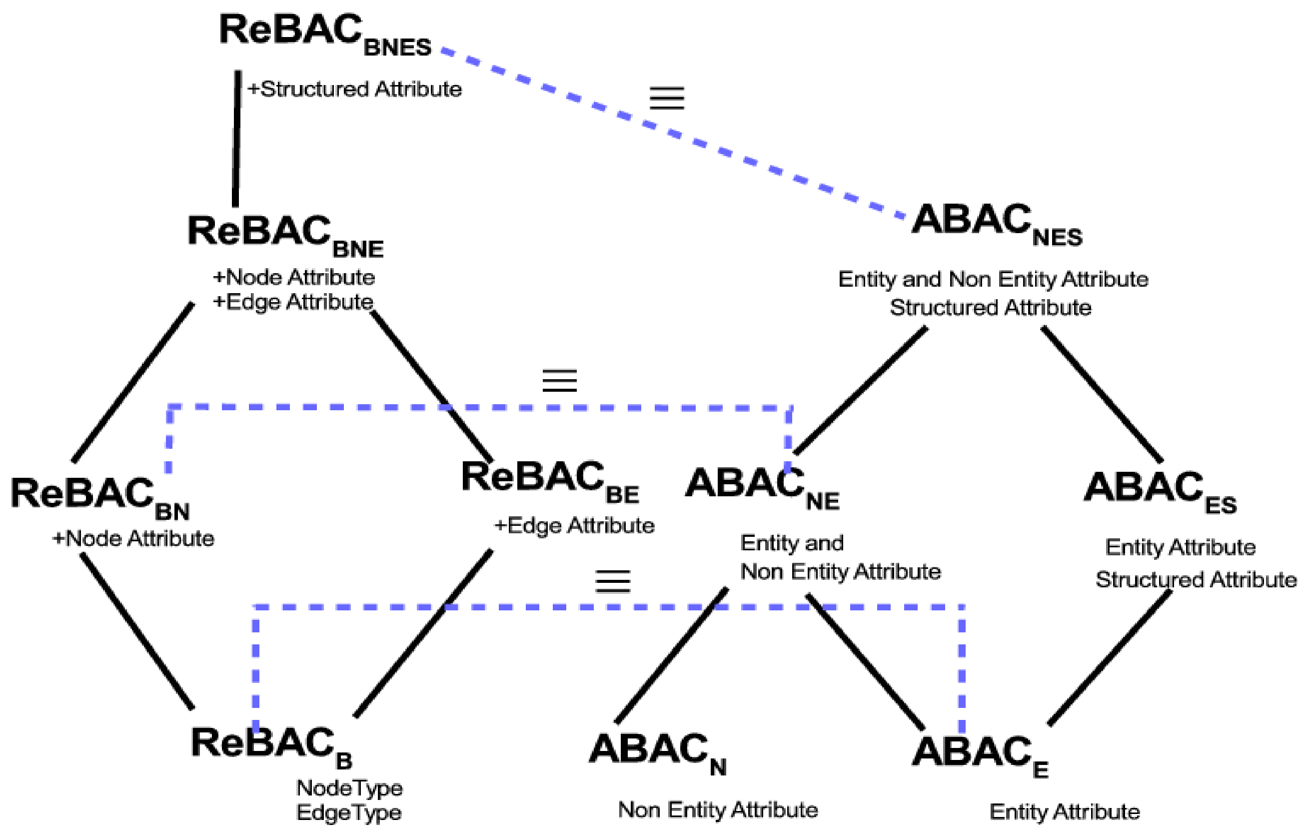


Figure 13: Equivalence of ReBAC and ABAC Structural Classification

Comparison: Non-Equivalent Structural models for ReBAC and ABAC

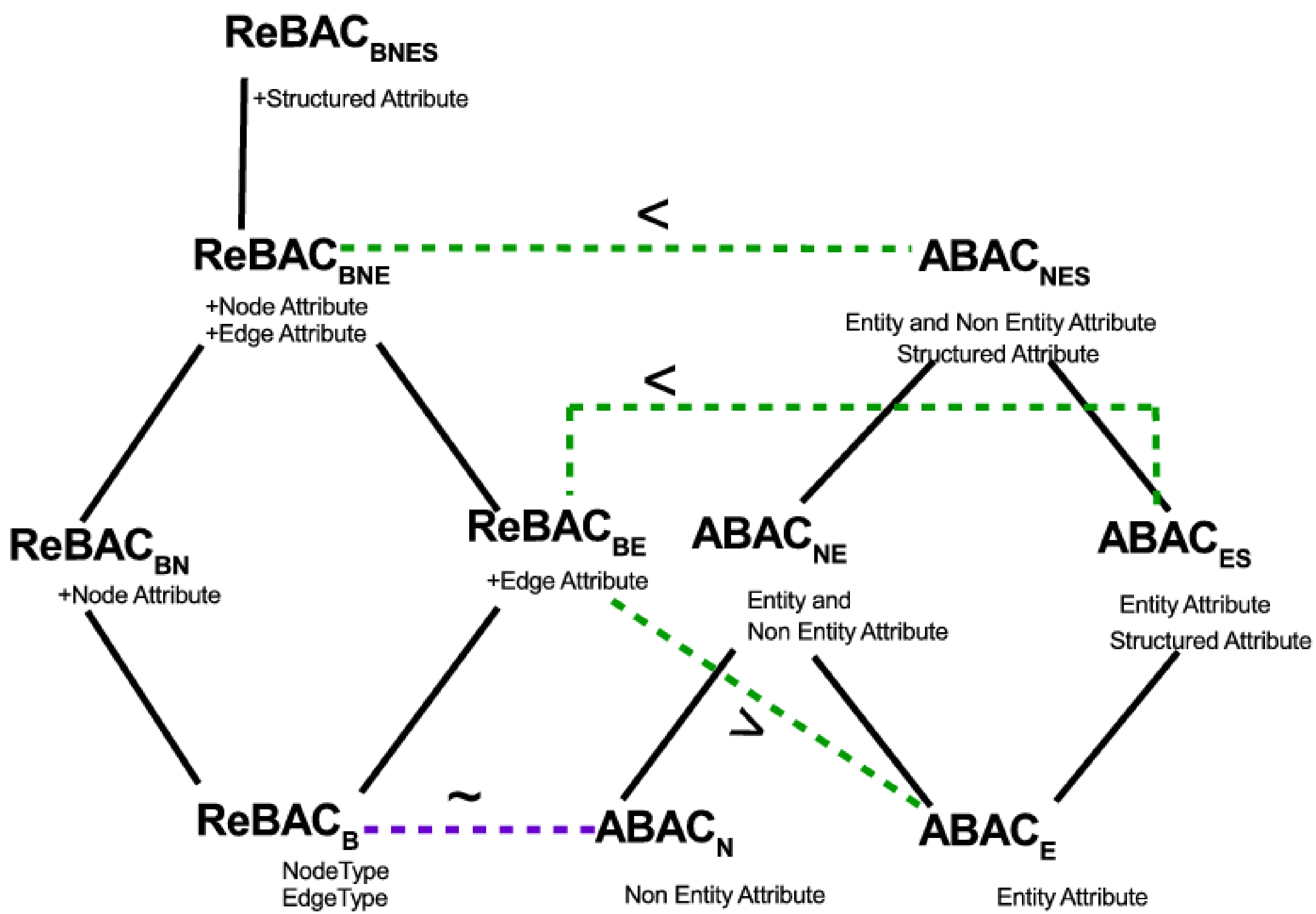


Figure 14: Non-Equivalence of ReBAC and ABAC Structural Classification

Comparison: On Performance

- Attribute Composition is similar to ReBAC and Both have polynomial complexity for authorization policy and constant complexity on update
- Composite attribute has constant complexity on authorization policy and polynomial complexity on update to maintain relationship changes.
- Performance Depends on :
  - Node Dynamics
  - Relationship Dynamics
  - Density of the Relationship Graph

Comparison: Choice of Models

- For static system or only change or non entity attribute------Composite attribute is the best approach
- System with huge node dynamics, relationship dynamics and high relationship density----- Attribute composition is the best option
- If the system is in the middle between two extremes ---- A hybrid approach where both composite attribute and attribute composition is used.
- Hybrid Approach:

To achieve p level relationship composition it uses m level composite attribute and n level attribute composition where p = n X m.

Comparison: In Respect of PEI Framework

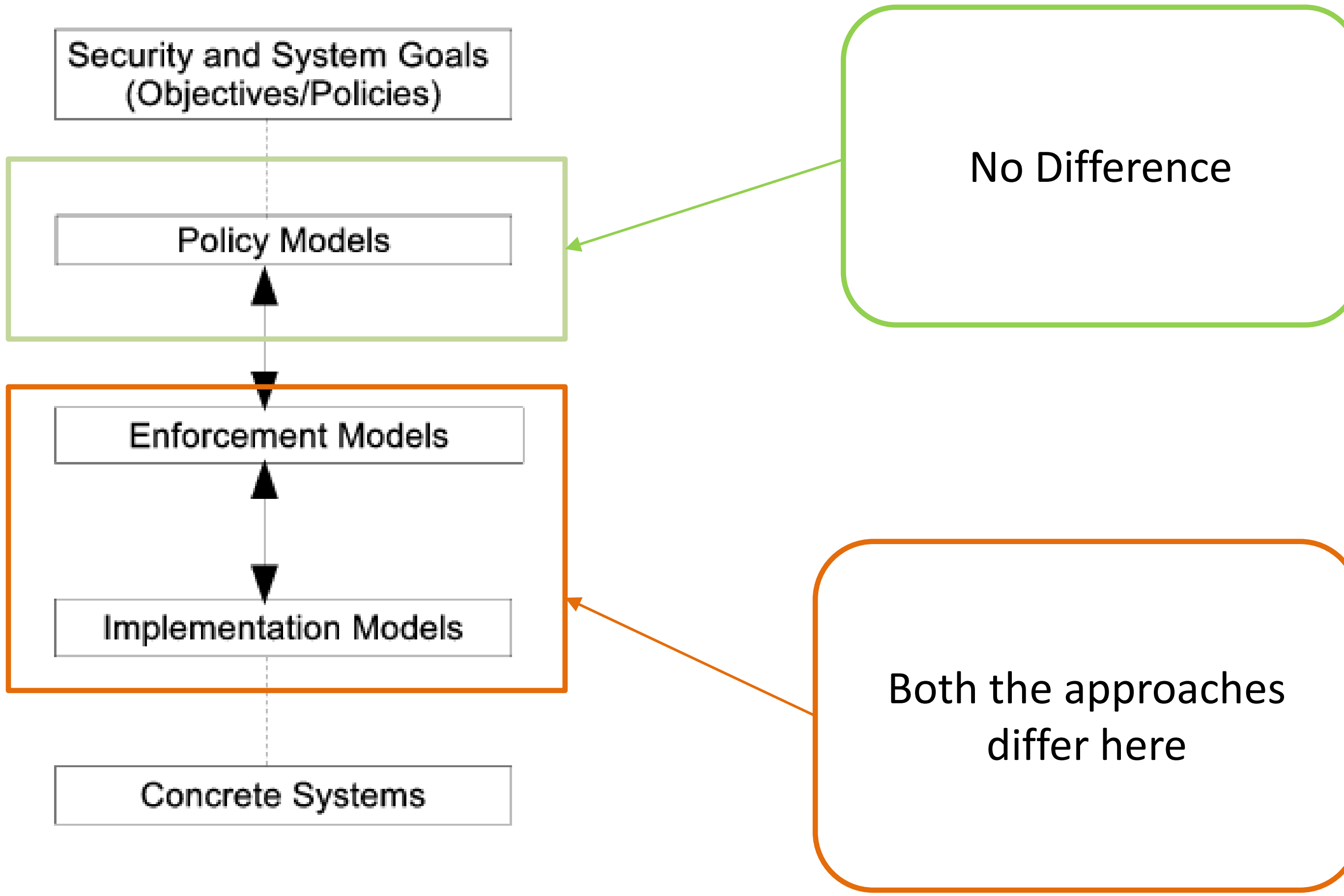


Figure 15: PEI Framework