Theorem
Given framework morphisms $f : SP_0 \rightarrow SP_1$ and $g : SP_0 \rightarrow SP_2$ with SP_1 and SP_2 coherent, the pushout framework $SP = (TG^{int}, (P^{int}, r_{P^{int}}), Pos^{int}, Neg^{int})$ is coherent w.r.t. the set of negative constraints Neg^{int}.

Theorem
Given framework morphisms $f : SP_0 \rightarrow SP_1$ and $g : SP_0 \rightarrow SP_2$ with SP_1 and SP_2 coherent, the pushout $SP = (TG^{int}, (P^{int}, r_{P^{int}}, Pos^{int}, Neg^{int})$ of f and g is incoherent if and only if SP is incoherent w.r.t. positive constraints of Pos^{int} containing types in TG_0.

CONCLUDING REMARKS

We have started the development of a method for specifying Access Control policies in terms of allowed and disallowed state graphs and constructive rules
We can begin to perform meaningful analyses of the relationships among different policies.

NEXT ?

- Transition between policies
- More general meta-policies
- Tool to verify properties of a specification
- Tool to convert a specification into policy enforcement code

If the integration of constraints is the union of the corresponding sets Pos and Neg, the integrated system graph may not be coherent. If the integrated system graph does not satisfy a constraint:

Changing the integrated system graph: The system graph can be changed. If not possible (constraints are contradictory, a change is not desired, etc.), the set of constraints must be changed.

Reducing the set of constraints: The constraint could be also deleted to get coherence. Removing of constraints is necessary if constraints are contradictory. Several strategies for removing constraints depending on the application domain.

Among the formal results

Theorem The category **SP** of security policy frameworks and sp-morphisms is cocomplete.

Theorem Let SP' be evolved from a coherent security policy framework SP and $C(SP')$ the security policy framework after modifying SP' according to the table then $C(SP')$ is coherent.

To resolve rule conflicts in policy integration, several possibilities:

Priority for one policy. This strategy sets the priority to one policy by choosing a major and a minor policy.
Radical solution: deletion of the conflicting rule of policy B from the set of available graph rules. Policy A "survives" during the subsequent evolution of the system, whereas policy B "dies".
Weak solution: the radical strategy can be weakened by keeping the conflicting rule of policy B, but changing it so that it is not applicable at conflict points. At non-conflict points the rule is still applicable.

Priority for rules. Instead of favoring one security policy in general, the choice is limited to the pair of conflicting rules.
Static solution: For each pair of conflicting rules, the preferred rule is chosen.
Dynamic solution: This strategy chooses the rules at run-time at each occuring conflict point. The rules of both policies remain unchanged.

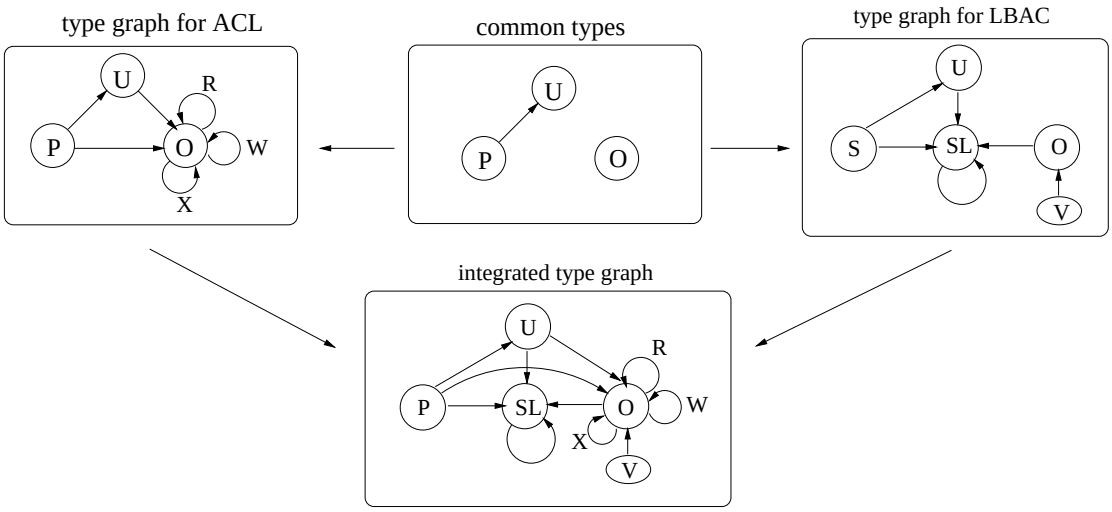


Figure 16: *Integrated type graph for the combined LBAC and ACL.*

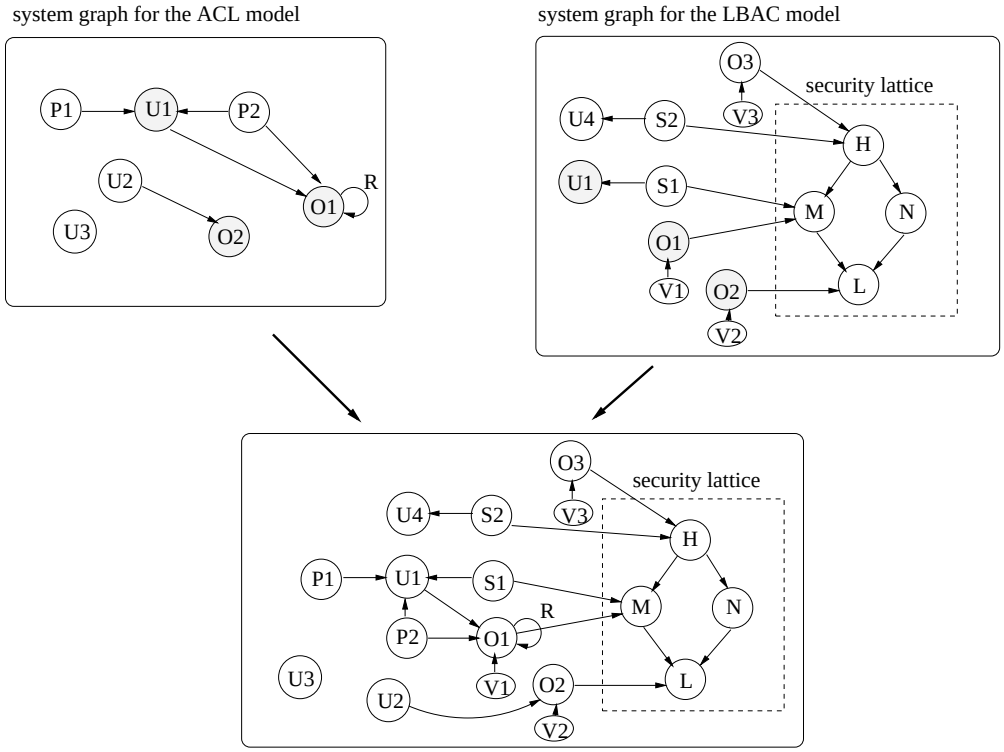


Figure 17: *Integrated system graph for the combined framework.*

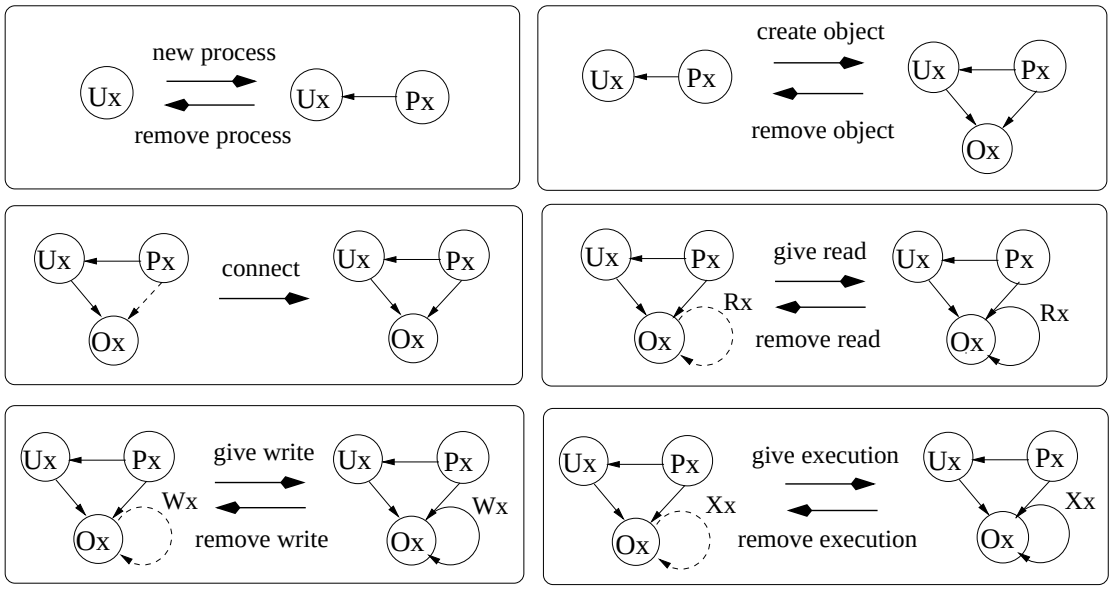


Figure 15: *Graph rules for the ACL model.*

The merging of two systems
- at the syntactical level with the security policy frameworks, and
- at the semantical level with the system graphs representing the state at the point of the merge.

The integrated type graph TG_{INT} is given by the gluing of the type graphs TG_1 and TG_2 along common objects.

On the semantical level, need to identify instances common to both system graphs (of a common type !). After the system graphs are 'translated' into the new type graph, they are merged.

Arbitrary changes in a security policy framework do not preserve coherence. To obtain a coherent evolution

type of change	step to ensure coherence
add rule r	construct application condition for r w.r.t. Neg
remove rule r	construct application condition for r w.r.t. Pos
add negative constraint c	construct application conditions for $Rules$ w.r.t. c
add positive constraint c	construct application conditions for $Rules$ w.r.t. c
remove neg. constraint c	no effect on coherence
remove pos. constraint c	no effect on coherence

Table 1: *How the basic steps of transformation can be made coherent.*

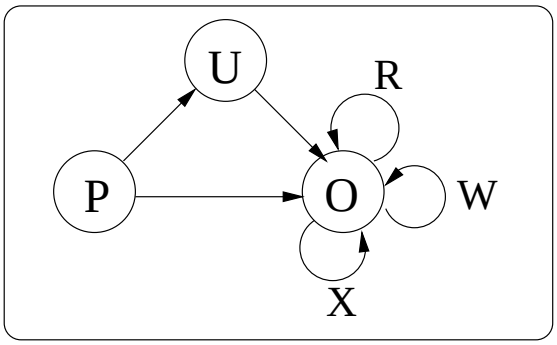


Figure 13: *Type graph for ACL.*

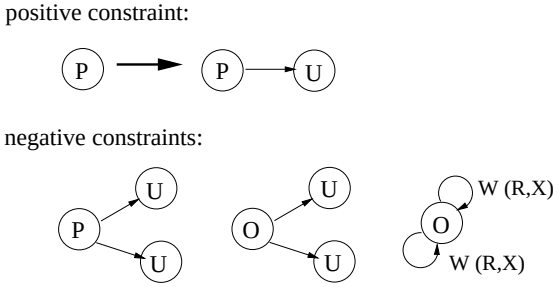


Figure 14: *Positive and Negative ACL Constraints.*

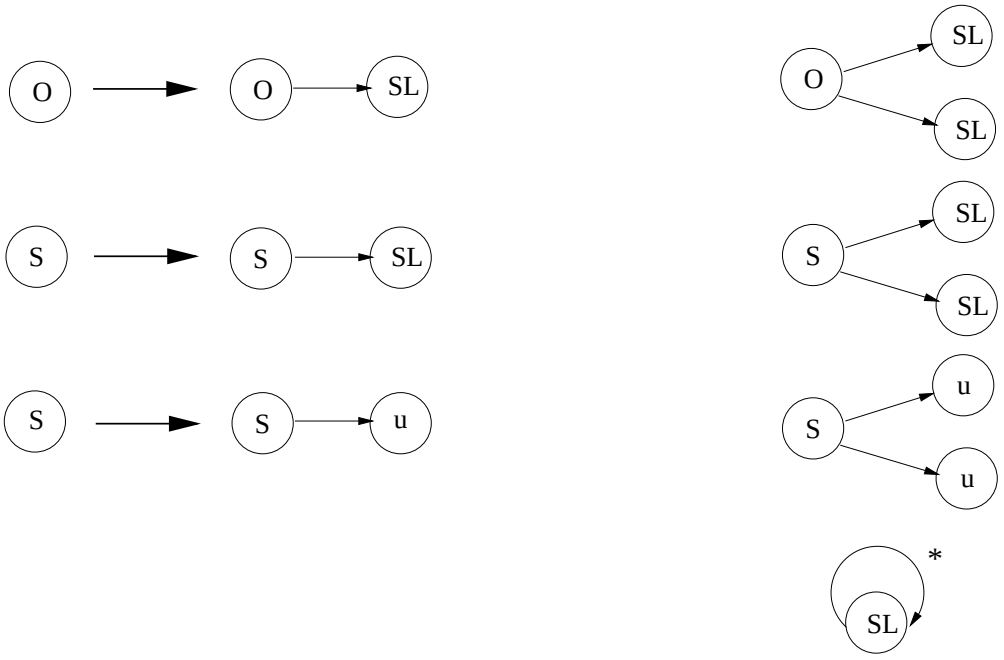


Figure 12: *Graphical constraints for LBAC.*

1. Evolution of one policy. A security policy can be restricted by new constraints, new rules can be introduced to provide new features, etc.

2. Integration of policies. Two models must be combined into a unique model. Problems may occur if both policies are available for parts of the combined model. The theory of graph transformations provides results to decide statically how and when rules may conflict.

3. Transition of policies. This strategy changes completely from one policy to another one.

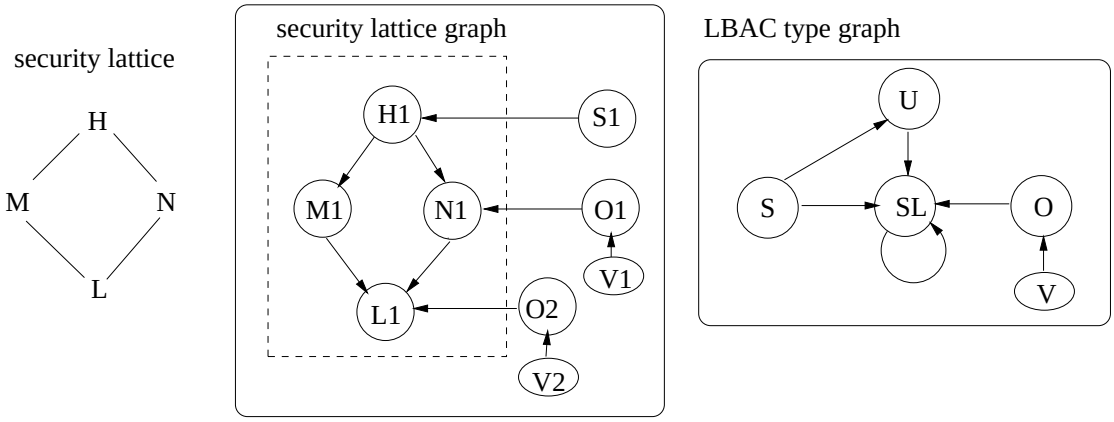


Figure 10: *A security lattice, its graph representation, and the type graph.*

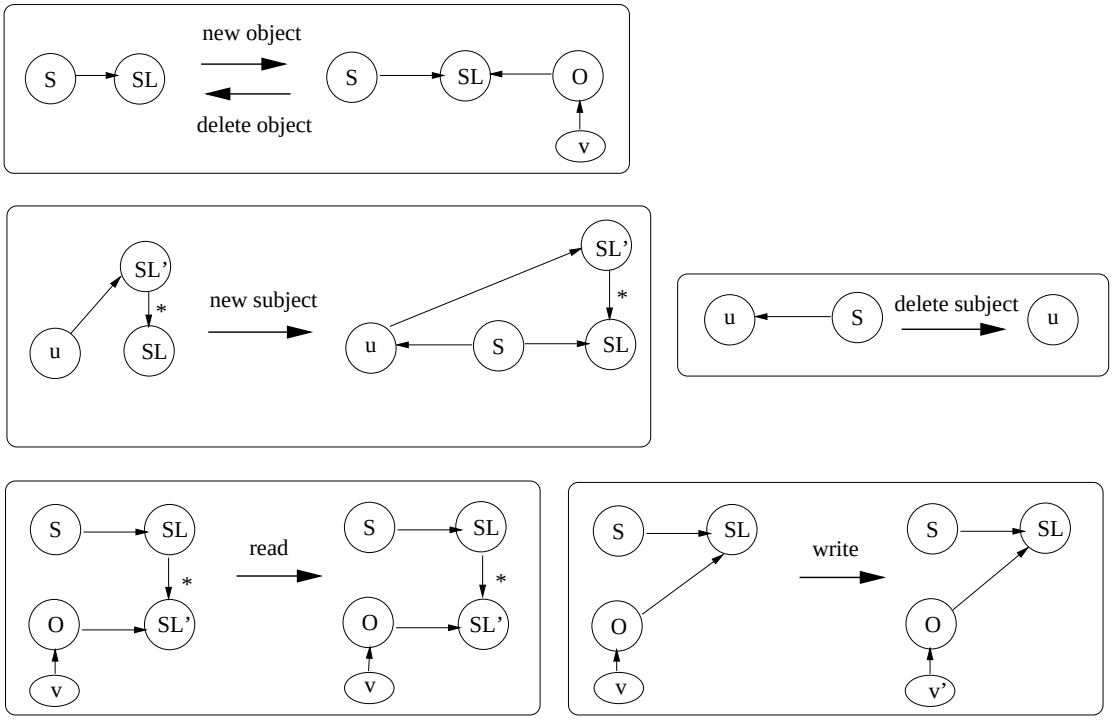


Figure 11: *Graph rules for the LBAC model.*

A *security policy framework* consists of four components:

- A type graph that provides the type information of the security policy.
- A set of graph rules for the specification of the policy rules that generate the system graphs accepted by the security policy.
- two sets of *constraints* that specify graphs that shall not be constructed (*negative constraints*) and graphs that must be explicitly constructed (*positive constraints*).

In the actual implementation the constraints are redundant (only acceptable states are those explicitly built).
Positive and negative constraints are a formal documentation of the initial requirements and the development process of rules.

Definition A *security policy framework* is a tuple $SP = (TG, (P, r_P), Pos, Neg)$, where TG is a type graph, the pair (P, r_P) consists of a set of rule names and a total mapping $r_P : P \to |\mathbf{Rule}(TG)|$ mapping to each rule name a rule, Pos is a set of positive and Neg is a set of negative constraints.

The graphs that can be constructed by the rules of a security policy framework are the *system graphs*.
A security policy framework is *coherent* if all system graphs satisfy the constraints in Pos and Neg.

Three models are proposed to tackle the first three problems

- **Static single assignment** At most one assignment per user; static means changed by deleting it and inserting a new one. The deletion of the assignment implies the loss of the authorization for roles.
- **Dynamic single assignment** One assignment per user, but assignment edge can move through the role hierarchy graph. Only the lowest role in the hierarchy can delete the assignment, since it cannot be moved down.
- **(Strong) Multiple assignment** Arbitrary number of assignments, deferring the decision on the deletion of an assignment edge to a higher level in the role hierarchy. Any administrator can stop the propagation of the deletion by keeping the assignment. Actual deletion takes place only at the top of the hierarchy. There can be multiple branches.

Comparing the four proposed specifications with respect to the addition and the deletion of assignments

policy	add assignment	remove assignment	comment
static single	easy	very easy	unflexible, restricted
dynamic single	easy	easy	more flexible
weak multiple	easy	not easy	more flexible but weak revocation
strong multiple	easy	complex	flexible and veto

In the decentralized administration of roles, the responsibility of a range of roles is not sufficient to guarantee the desired effect.

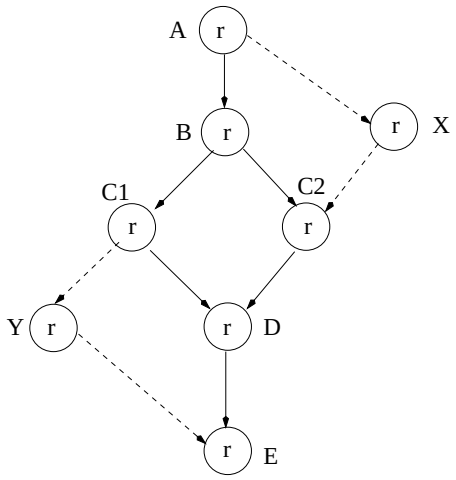


Figure 9: *Possible Problems in the RBAC model.*

Deletion of a user from a role: The removal of a user from a role may not have an effect if the user is a member of a senior role (*weak revocation*)

Deletion of a permission from a role: A revocation of a permission from a senior role has no effect on the senior if a junior role still has this permission.

Deletion of roles: The range for an administrator is given by an interval over the partial order and deleting the boundaries of the interval destroys the range definition: only roles inside the interval can be deleted.

Special hierarchy graphs: If a special structure for the hierarchy graph for (administrative) roles is required, changes to the graph may destroy it.

The basic operations of the RBAC model are
add user, *remove user*,
add session, *remove session*,
add assignment, *remove assignment*,
add inheritance, *remove inheritance*,
add role, *remove role*,
activate role, *deactivate role*.

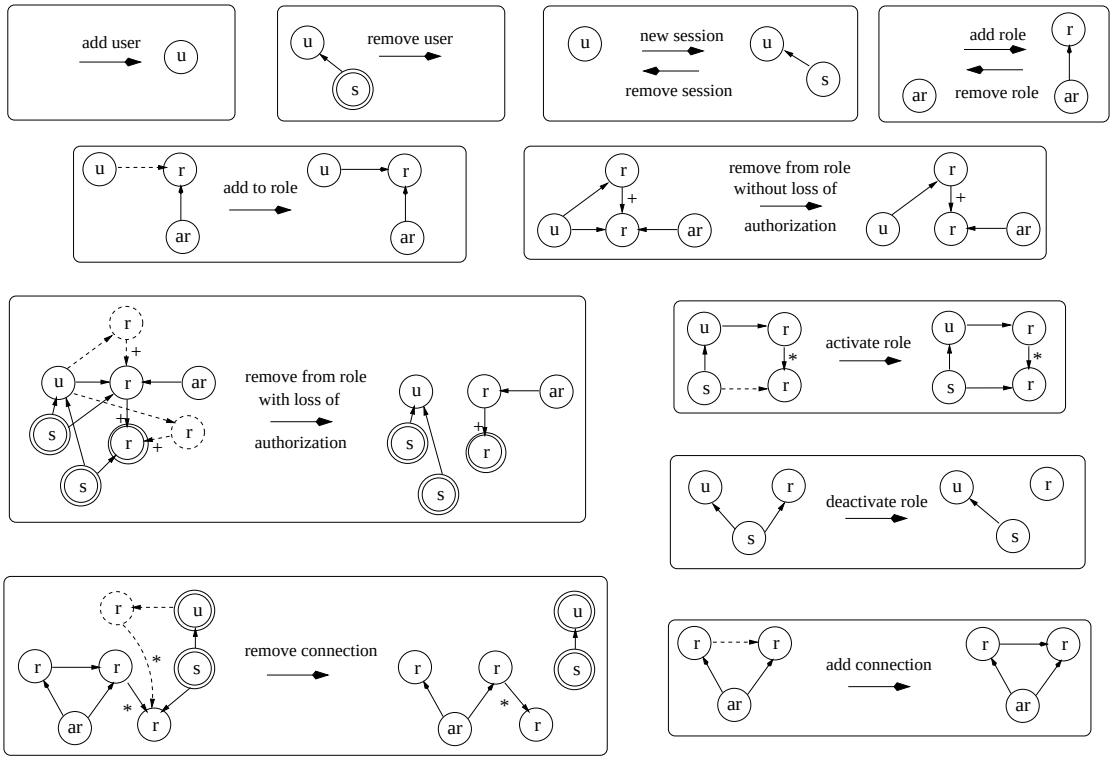
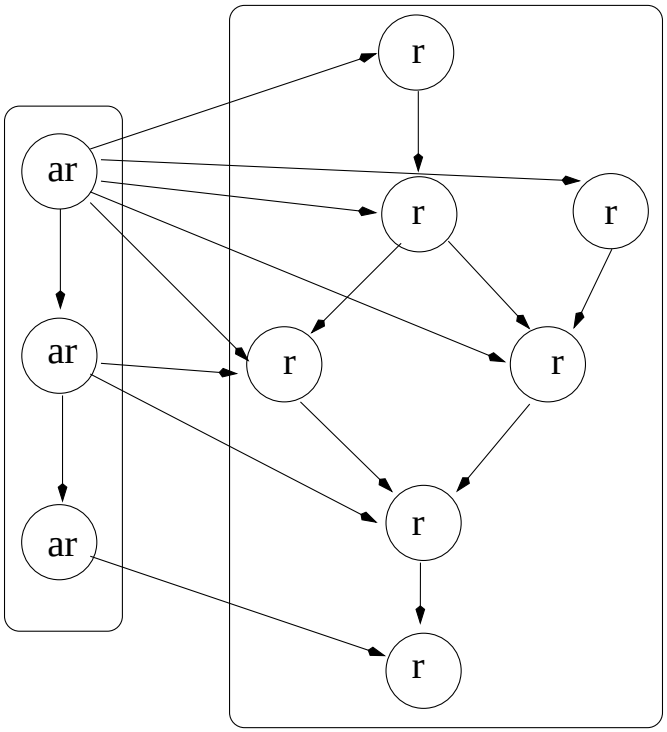


Figure 8: *Graph rules for the RBAC Model.*

In the ARBAC97 model, the (administrative) role hierarchy is given by a partial order over (administrative) roles.

Edges between administrative roles and user roles represent the authorization to modify the user role. The set of roles reachable by such edges is the *range* of the administrative role.

Role management (creation and deletion of roles, assignment and revocation of users and permissions to roles) is the responsibility of *administrative roles*.



In our approach the designer has to perform step 1 **only** !
The derivation of the conditions in step2 can be performed automatically and the result is *guaranteed* to satisfy the given consistency properties.

Algorithm:
INPUT rule $r : L \to R$ and a graphical constraint C;
OUTPUT a set $A(C)$ of application conditions
Step 1: Construction of all possible non-empty overlappings K of the right-hand side R of the rule r and the graphical constraint C.
Step 2: For each gluing found in Step 1 an application condition (L, N) is constructed by applying the *inverse rule* of r to each overlapping K.
Step 3: The algorithm minimizes the set of application conditions found in Step 2 by removing application conditions that are always satisfied in consistent graphs.

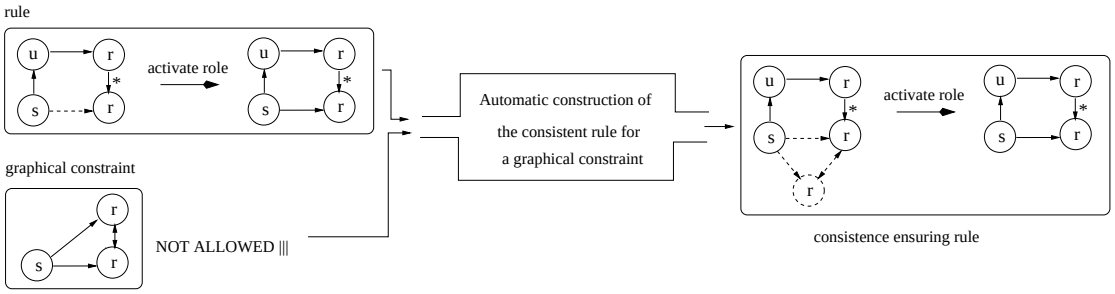


Figure 7: *Construction of a consistent rule from a constraint.*

Theorem: Given a rule r, a graphical constraint gc, a graph G consistent w.r.t. gc, and the rule $r(gc)$ modified as above, the graph H resulting from an application of $r(gc)$ to G is consistent w.r.t. gc.

Rules to add a new role and to assign a permission to a role

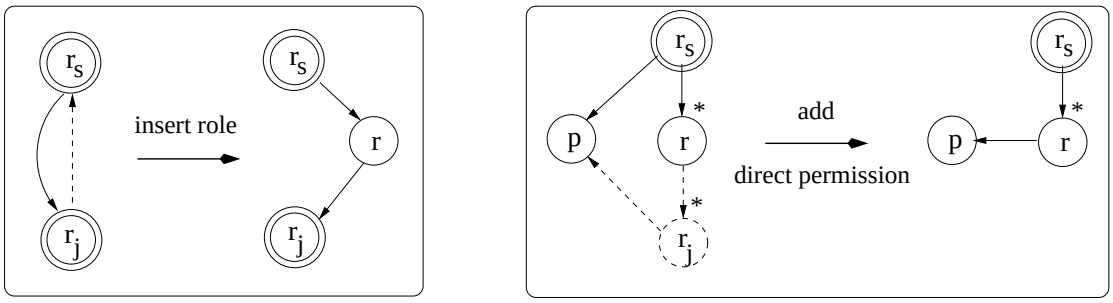


Figure 6: *Rules for role/permission management: role insertion.*

Rules have been defined to model role deletion (with permission elimination or with permission distribution), and role partition (both horizontally and vertically)

How can we prove the correctness of a RBAC specification ?

In the approach by GavrilaBarkley98, consistency requirements are defined in logical terms and a state is consistent if it satifies the requirements.

The designer has to:
1. define the consistency properties on the entire system,
2. derive from step1 the conditions for each operation, and
3. prove that the execution of each operation, satisfying the condition in step2, preserves the properties of step1.

Our framework is suitable to specify the Role Graph Model of NyanchamaOsborne 1999.

Permissions are modeled by nodes of type p connected to objects.
The role-permission assignment is also modeled by edges.
The authorization of a user for an object is given by the existence of a path from the user to the object.
Our graph model is similar to Baldwin90 privilege graphs (PG). A PG is a three-layered acyclic graph:

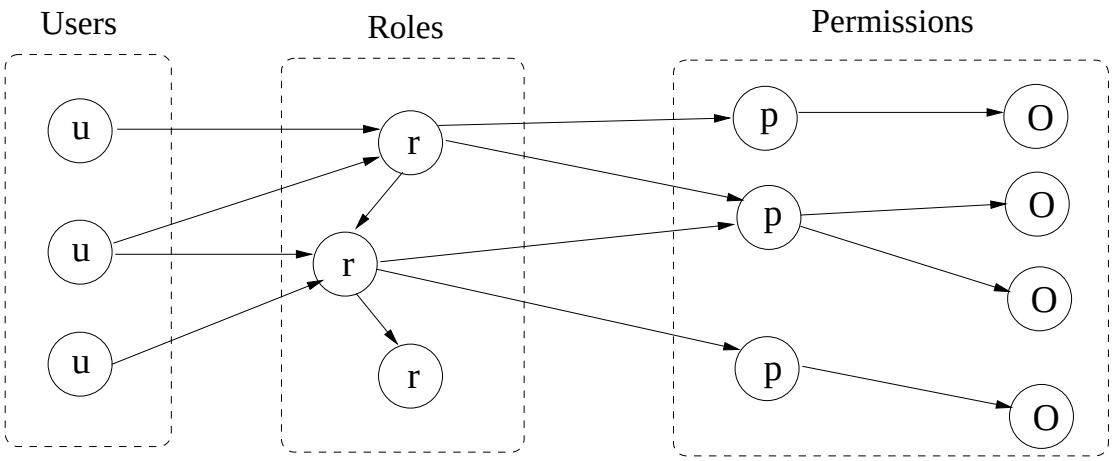


Figure 5: *The graphical specification of user-role, role-permission and permission-object assignment.*

Centralised RBAC model: only one administrative role responsible for any role in the role graph.

A user can be assigned to or revoked from a role.
A user is a member of a role if directly assigned to a role.
She/he is authorized for a role, if the role is inherited from a role to which the user is assigned.

The basic operations of the RBAC model are *add user*, *remove user*, *add to role*, *remove from role*, *add session*, *remove session*, *activate role* and *deactivate role*.

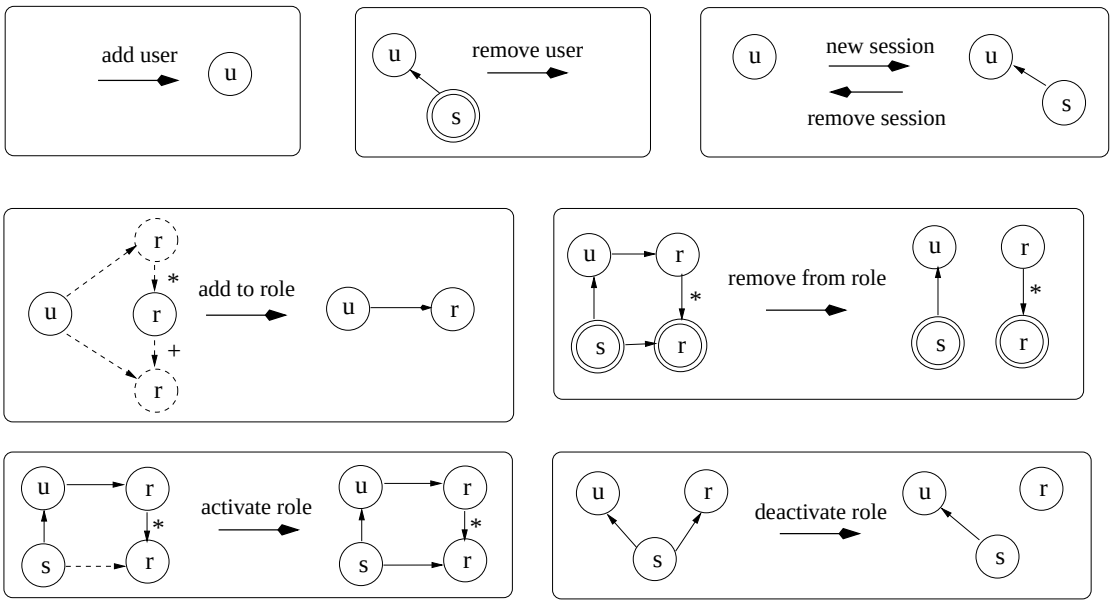


Figure 4: *Graph rules for the centralized RBAC model.*

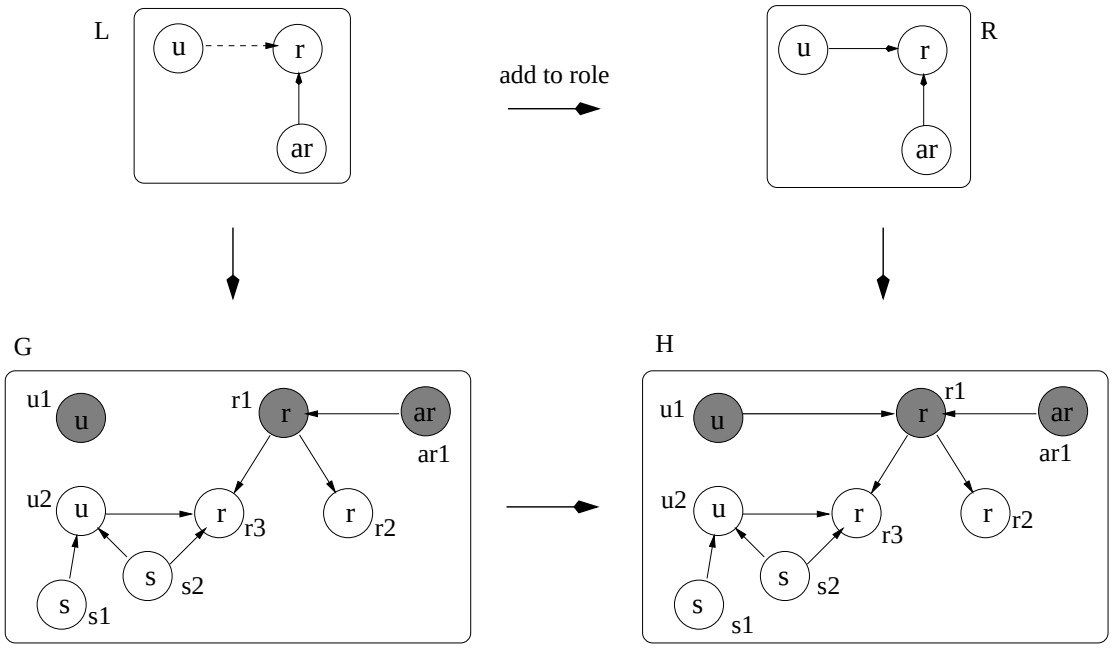


Figure 2: *Application of rule* `add to role`.

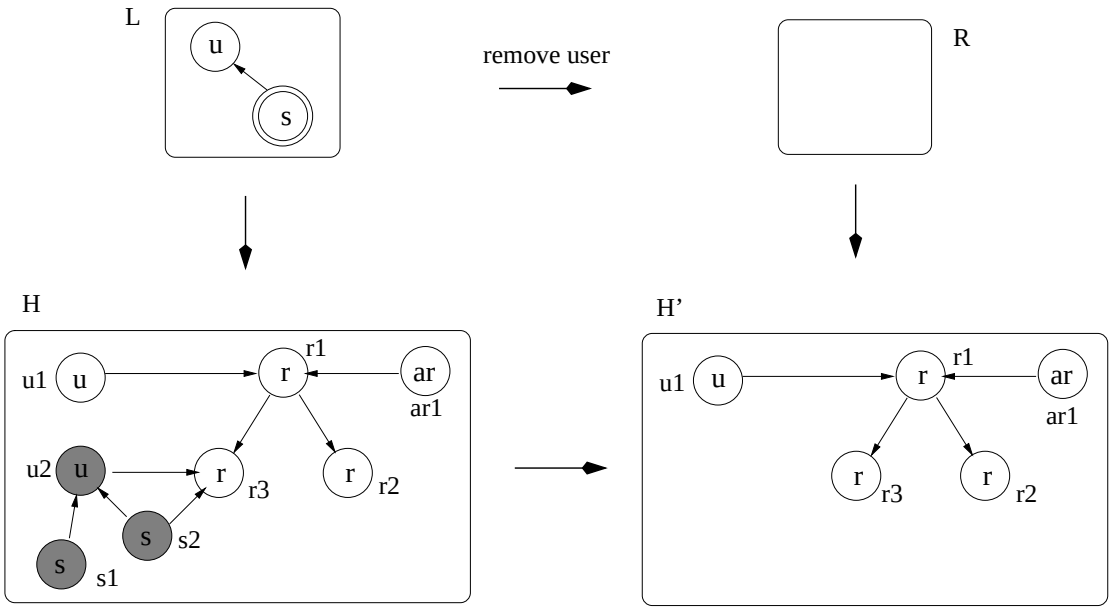


Figure 3: *Application of rule* `remove user`.

A negative application condition (NAC) for a rule $r : L \to R$ is a pair (L, N), where the graph L is a subgraph of N. .

A graph G *satisfies* a rule $r : L \to R$ with NAC (L, N) if L occurs in G and it is not possible to extend L to N.

The application of a rule $r : L \to R$ to a graph G takes place in four steps:

1. Find L as subgraph $L(G)$ in G;
2. If every negative application condition (L, N_i) is satisfied, then:
   (a) Remove all nodes and edges from $L(G)$ that are not in R.
   (b) Add all nodes and edges in R that are not in L. (The nodes both in L and in R are used "as glue").

A *total graph morphism* $f = (f_N, f_E) : G \to G'$ is a pair of total functions $f_N : N \to N'$ and $f_E : E \to E'$ such that
$src' \circ f_E = f_N \circ src$ and $tar' \circ f_E = f_N \circ tar$
(conditions on labels and attributes as well)

A *partial graph morphism* $f : G \rightharpoonup G'$ is a total graph morphism $\bar{f} : dom(f) \to G'$ from a subgraph $dom(f) \subseteq G$ to G'.

for a fixed *typed graph* $TG \in \mathbf{Graph}$
a *TG-typed graph* (G, t_G) is a graph G and a graph morphism $t_G : G \to TG$.

A *graph rule* $p : r$ is given by a rule name $p \in RNames$, and a label preserving *morphism* $r : L \rightharpoonup R$.

- L describes what a graph must contain for p to be applicable;
- Parts of L for which r is undefined are intended to be deleted;
- Parts of L in $dom(r)$ are intended to be preserved;
- Parts of R without a pre-image in L are newly created.

The actual deletions/additions are performed on the graphs to which the rule is applied.

Advantages of using a graph based formalism for RBAC

- an intuitive visual description of the manipulation of graph structures as they occur in AC;
- an expressive specification language (for a detailed specification of various schema for decentralizing administrative roles);
- a specification of static and dynamic consistency conditions on graphs and graph transformations;
- a uniform treatment of user roles and administrative roles;
- a symmetric treatment of user-role assignment and permission-role assignment;
- an executable specification that exploits existing tools to verify the properties of a given graph-based RBAC description.

BACKGROUND

A *graph* G is (N, E, src, tar) where
N is the set of nodes, E is the set of edges and
$src, tar : E \to N$ are the *source* and *target* functions

Nodes and edges are typed, labelled and can be attributed

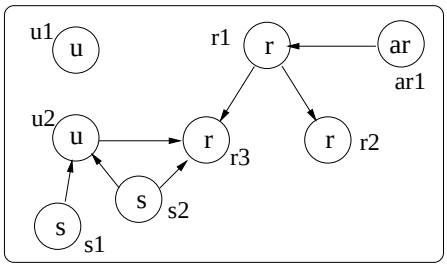


Figure 1: *A RBAC state graph.*

A framework for the Specification and Evolution of Access Control Policies

Francesco Parisi-Presicce

Univ.Roma La Sapienza (visiting GMU)

joint work with L.V.Mancini and M.Koch

Partially supported by the European Community under TMR GETGRATS and WG APPLIGRAPH

The aim of this line of research is

- the development of a uniform and precise framework for the specification of Access Control Policies and the comparison of different policy models,
- the definition of a methodology to systematically generate conditions on operations to guarantee consistency, and
- the formalization of the notion of evolution of a policy and of integration of policies.