A Formal Model for Role-Based Access Control Using Graph Transformation [⋆]

Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce

Dip. Scienze dell'Informazione, Universitá degli Studi di Roma *La Sapienza*
Via Salaria 113, 00198 Roma, Italy, {carr,lv.mancini,parisi}@dsi.uniroma1.it

Abstract. Role-Based Access Control (RBAC) is supported directly or in a closely related form, by a number of products. This paper presents a formalization of RBAC using graph transformations which is a graphical specification technique based on a generalization to nonlinear structures of classical string grammars. The proposed formalization provides an intuitive description for the manipulation of graph structures as they occur in information systems access control, a specification of static and dynamic consistency conditions on graphs and graph trasformations, a uniform treatment of user roles and administrative roles, and a detailed analysis of the decentralization of administrative roles. Moreover, the properties of a given RBAC specification can be verified by employing one of the graph transformation tools available.

1 Introduction

The activities within a computer system can be viewed as a sequence of operations on objects. One of the primary purposes of security mechanisms is *access control* (AC) which consists of determining and enforcing which active entities, e.g processes, can have access to which objects and in which access mode.

This paper focuses on Role-Based Access Control (RBAC), an AC mechanism described in [SBM99]. It appears that RBAC tries to match the need of many organizations to base AC decisions on the roles assigned to users as part of the organization. In this context RBAC facilitates the administration of AC decisions while making the process less error-prone. A formal analysis of RBAC would be useful for the following reasons:

1. prove the properties of a given RBAC specification: RBAC consists of a family of conceptual models and the most advanced ones include role hierarchies, constraints, and decentralized administration of roles; this complexity requires a formal setting to ensure that a RBAC specification meets basic correctness properties of the system.

2. compare different AC models: for example, to better meet the needs of a specific application one could compare the pros and cons of discretionary AC, mandatory AC and RBAC;

⋆ partially supported by the EC under TMR Network GETGRATS,Esprit WG APPLIGRAPH, and by the Italian MURST.

3. predict the system behavior in combining different AC policies: for example, to support *role transition*, that is the means for moving towards RBAC in coexistence with previous models of AC.

This paper presents a formalization of RBAC using graph transformations which is a graphical specification technique based on a generalization of classical string grammars [Roz97] to nonlinear structures. The advantages of using the graph transformation formalism are: an intuitive visual description of the manipulation of graph structures as they occur in the AC; an expressive specification language, which allows, for example, the 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; an executable specification that exploits existing tools [EEKR99] to verify the properties of a given graph-based RBAC description.

The issues addressed in this paper include: the presentation of a formal framework for RBAC which allows the specification and verification of consistency requirements; the discussion of some open problems, such as the revocation cascade of users membership when administrative roles are decentralized, and a comparison of several possible solutions. In particular, this paper focuses on the issue of (decentralized) administration of user/role assignment and revocation in the RBAC model; the administration of permission/role assignment and revocation, discussed in [NO99] for the case of one administrator, is not explicitly treated here, but can be described in a similar way. The comparison of different AC policies within the graph grammar formalism and the analysis of the system behavior in combining different AC policies are the subject of another paper [KMPP00].

The rest of this paper is organized as follows. Section 2 overviews the basic notions of graph transformations. Section 3 describes the graph based modeling of RBAC. Section 4 shows how to prove the graph based correctness of a RBAC specification and briefly discusses the use of constraints. In section 5, we show that the formalism is sufficiently expressive to describe possible solutions to some nontrivial open problems mentioned in [San98], such as the decentralized administration of roles. To our knowledge, this was an open issue, now solved here at the specification level with a proposal of different solutions, compared in section 6. Section 7 contains some concluding remarks and additional comparison with existing work.

2 Graph Transformation

A graph consists of a set of nodes and a set of edges. Edges are directed, i.e. they run from a node (the *source* of the edge) to a node (the *target* of the edge). Each node and each edge has a *type*. Several nodes and edges of the same type may occur in one graph. Figure 1 depicts a graph with four node types and only one edge type. The node types are u, s, r and ar and will be used to describe the role-based access control model introduced later on. Nodes of type

u represent users, nodes of type r are roles, nodes of type ar are administrative roles and nodes of type s model sessions. A detailed explanation of the graph model follows in Sec. 3. This section focuses on the main components of graph transformation, which are a *transformation rule* and a *transformation step*. We introduce the concepts in this section only informally. A detailed (also formal) introduction can be found in the handbook to graph transformation [Roz97]. A

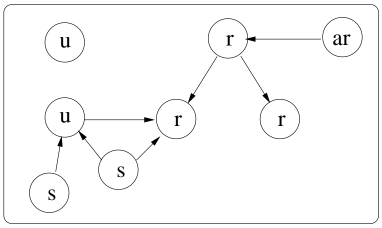

Fig. 1. *A RBAC state graph.*

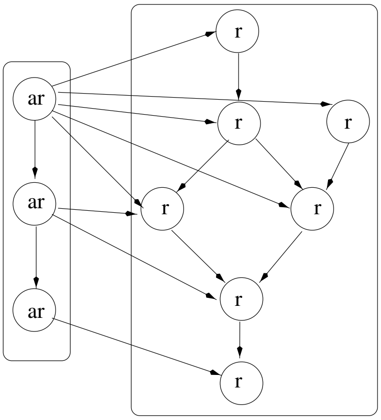

Fig. 2. *Administrative role hierarchy graph (left) and role hierarchy graph (right).*

graph represents a state of a system, for example a security system based on roles as shown in Fig. 1. State changes are specified by graph transformation rules, called just *rules* in the following. A rule is formally given by a *graph morphism* $r : L \to R$, where both L and R are graphs, called *left-hand side* and *right-hand side*, respectively. A graph morphism consists of an injective partial mapping r_n between the sets of nodes and an injective partial mapping r_e between the sets of edges of L and R. The mappings must be compatible with the graph structure and the types. Compatible with the graph structure means that whenever the mapping for edges is defined for an edge e, the mapping for nodes is defined for the source s and the target t node of the edge e and $r_n(s)$ and $r_n(t)$ are the source and target nodes of the edge $r_e(e)$ in the right-hand side. Compatible with the type means that nodes and edges are mapped only to nodes and edges of the same type. If the mappings are total, we call the graph morphism *total*. The graph L describes which objects a graph G must contain for the rule $r : L \to R$ to be applicable to G. Nodes and edges of L for which the partial mappings are undefined are deleted by the rule. Nodes and edges of L for which the partial mappings are defined are preserved and have an image in R. Nodes and edges of R without a preimage in L are newly created.

The complete set of rules for the RBAC model is given in Section 3 and 5. We choose now two of them to explain the application of rules. The first rule is given by `add to role` in Fig. 4. The intended meaning of this rule is to make a user a member of a role. The membership of a user to a role is modeled by an edge, from the u to the r node, that this rule must create. The assignment of a user to a role takes place by an administrative role that is responsible for the role. In our graph model, responsibility is indicated by an edge from the ar to the r node. So, the left-hand side L consists of one node of type ar, one of type r and one of type u, and an edge between the ar node and the r node to show the responsibility. The dashed edge between the u and the r node represents a *negative application condition.* A negative application condition represents a structure that, as a whole, must not occur in G for the application of the rule. For the rule `add to role`, a membership edge between the u and r nodes must not exist in G before the rule application. If the unwanted structure occurs, then the negative application condition is not satisfied. Note that the presence of a part only of the unwanted structure is acceptable: for example in the rule `veto deletion` in Fig. 9, there could be a senior role, provided that it is not assigned to the user.

Whereas `add to role` is a preserving rule (the graph morphism for the rule is total, so that the rule does not delete anything), the rule `remove user` (see Fig. 4) removes a user from the system. Since we do not want to have active sessions that do not belong to any user, also all the sessions of the deleted user must be closed. Therefore, the left-hand side of `remove user` consists of a user node u connected to a session node s. The double circle around the s node (these nodes are called *set nodes*) specifies that the rule has to be applied to *all* session nodes s connected to the user; this means, in particular, that the rule can be applied also to users without active sessions. The right-hand side of the rule is empty to specify the deletion of the user with all her/his sessions.

The application of a rule $r : L \to R$ to a graph G takes place in four steps:
1. Find the left-hand side L as a subgraph in G. The subgraph is denoted by $L(G)$.
2. If there is a negative application condition that is not satisfied (that is, L(G) can be extended in G to an unwanted structure) then stop.
3. Remove all nodes and edges from $L(G)$ that are not present in R.
4. Add all nodes and edges in R that are not present in L. The nodes occuring both in L and in R are used to connect the new parts to the old ones.

The left-hand side L of the rule `add to role` occurs several times in the graph of Fig. 1. One possible matching is shown in Fig. 3 by the gray node pattern. For simplicity, labels on nodes have been omitted from graph and rules in the example and only their types are shown. In concrete graphs (like the one in Fig. 6), nodes have unique labels (e.g., specific names for users and roles) and there is no ambiguity on the application of a rule to a graph, since matching nodes have the same labels. Next, we have to check the negative application condition of the rule that requires that there is no edge between the u and the r node. Since the edge does not exist for the given matching, the rule is applied. It deletes nothing and inserts the edge between the nodes determined by the match of L. The transformed graph contains a new edge indicating that the gray user is a member of the gray role.

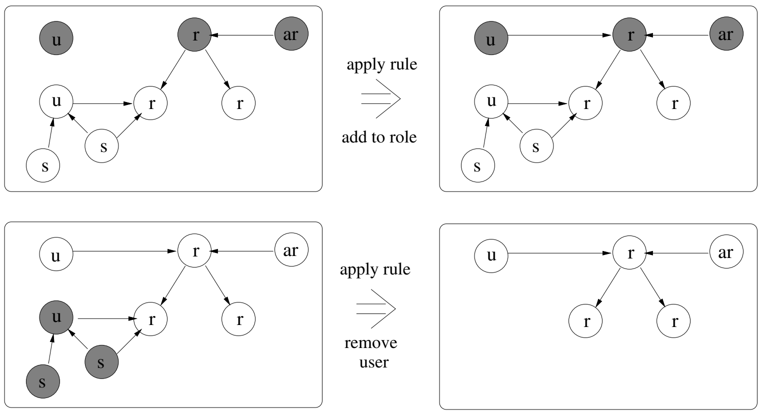

Fig. 3. *Application of rule* `add to role` *(top) and rule* `remove user` *(bottom).*

We now apply the rule `remove user` to this new graph. We can apply it to the upper or to the lower user node, and in Fig. 3 the lower node is chosen (again node labels can be used to avoid ambiguity). Since this user has two active sessions, they also have to be removed. The double circle in the rule *remove user* requires that we match all the sessions of the user, and it is not enough to match only one session or none at all. Note that set nodes can be given a variable label that can match any label in a concrete graph [PEM87]. There is no negative application condition to check. We delete the whole gray part, since a node or an edge does not exist in the right-hand side of `remove user`. Notice that the deletion of the right session node leaves a dangling edge, i.e., an edge without a source node. The same is true for the edge between the gray user and the connected role node. Since dangling edges are not allowed by the definition of graphs, dangling edges are automatically deleted in the graph transformation semantics as well.

In Section 3 the concept of a *path* is used. A path in a graph is a sequence of edges, where the target node of each edge in the sequence coincides with the source node of its successor edge in the sequence. We denote paths between two nodes by an edge equipped by $*$ for paths of arbitrary length including length 0 or a $+$ for paths that have at least length 1 (see for example the rules in Fig. 4).

Graph transformations are supported by several tools described in [EEKR99]. They provide mainly graphical editors to insert/draw the graphs, to specify rules with negative application conditions and a graph transformation machine to apply rules.

3 Role-Based Access Control

This section concerns with the specification of the **ARBAC97** model introduced by Sandhu et al in [SBM99] using graph transformations. The graph model includes administrative roles as well as inheritance of roles. In the **ARBAC97** model, the (administrative) role hierarchy is given by a partial order over (administrative) roles. In Figure 2 an example of an administrative role hierarchy (on the left-hand side) and a role hierarchy (on the right-hand side) is shown, where the hierarchies are given by graphs. Roles as well as administrative roles are given by nodes of type r and ar, respectively, and edges between roles show the inheritance relation. Graphs generalize the notion of a partial order, and by defining specific rules one could restrict the graph structure to any role hierarchy. Additionally, Figure 2 shows edges between administrative roles and user roles, representing the authorization to modify the user role by the administrative role. The set of roles reachable by such edges from an administrative role is the *range* of the administrative role. For instance, the range of the upper administrative role is given by the upper five roles, whereas the lower administrator has the authorization only for the lowest role. The use of a set of roles to represent a range differs from the approach of [SBM99], where the range is represented by an interval over the partial order, and ranges of junior administrators are required to be subranges of the senior administrator. The approach proposed there is not particularly suited to manage dynamic changes in administrator's ranges. We discuss these problems and others in a later section in more detail. Our approach is more general since the concept of an interval can be seen as a particular assignment of administrative roles to user roles. Moreover, our approach allows the modelling of administrators with disjoint role responsibility.

A user can be a member of a role and authorized for a role: a user is authorized for a role, if the role is inherited from a role to which the user is assigned. A user can establish a session during which the user activates a subset of roles of which she/he is a member. Note that the concept of sessions corresponds to the notion of *subjects* in the classical AC terminology. Role management, that is the creation and deletion of roles as well as assignment and revocation of users and permissions to roles, is the responsibility of *administrative roles*. The basic operations of the RBAC model are *add user*, *remove user*, *add assignment*, *remove assignment*, *add inheritance*, *remove inheritance*, *add role*, *remove role*, *add session*, *remove session*, *activate role* and *deactivate role*. All these operations are now modeled by graph transformation rules (see Fig. 4). The resulting graph specification uses a weak revocation, meaning that the deletion of a user from a role is not propagated to roles higher in the hierarchy; this corresponds to the URA97 model in [San98].

add user and **remove user**: The rule `add user` has an empty left-hand side, since users can be created at any time. The result of the rule is a new user represented by a node of type u. The rule `remove user` removes a user by deleting the corresponding user node u. To ensure that there are no active sessions of this user after the deletion of the user, all his/her sessions are deleted as well. This is indicated by the double circled session node in the left-hand side of rule `remove user`. The interpretation is that *all* sessions connected to the user are deleted. The deletion of the session implies the deletion of all the connections to roles that the session has, this is guaranteed by the graph trasformation approach chosen. The roles themselves remain.

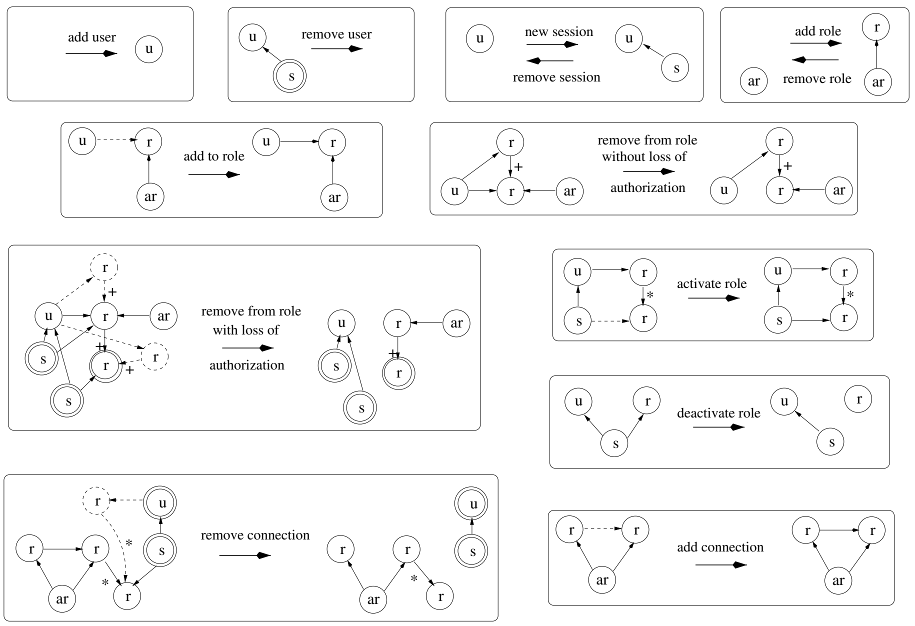

Fig. 4. *Graph rules for the RBAC Model.*

add session and **remove session**: A session is graphically presented by a node of type s. It has always a connection to one user. The rules for the creation and deletion of sessions are `new session` and `remove session`. A session node s is immediately connected by an edge to the user who is using s. A session can be deleted at any time regardless of the presence of active roles of the session. The session is deleted by removing the session node. This implies that all session-to-role edges are deleted as well, automatically.

add role and **remove role**: The creation of roles is modeled by the rule `add role`. Roles are added and removed by administrative roles. The new role becomes an element of the administrator's range indicated by the edge shown in the rule. The deletion of a role is only possible by an administrative role responsible for that role. Then, the r node and the connecting edge are removed.

add assignment and **remove assignment**: The assignment of users to roles is modeled by the rule `add to role`. To ensure that only an administrator responsible for the role assigns the user, an ar node connected to the r node is required. An administrator assigns a user to a role by connecting the user node and the role node by an edge, called *assignment edge*. Then, the user becomes a member of this role. This is allowed if and only if the user is not already a member of role r, specified by the negative application condition indicated by the dashed assignment edge. The deletion of the assignment edge does not necessarily remove the authorization of the user u for the role r. In particular, if there is an assignment edge to a higher role which is inherited by r, then u remains authorized for r. This case is modeled by the graph rule `remove from` `role without loss of authorization`. The rule requires that u is assigned to a role higher in the hierarchy. The + at the edge between the two roles indicates a non empty path in the role hierarchy. It ensures that the two role nodes are different. In this case, the assignment edge can be simply deleted from the lower role. No other actions are necessary since the user is still authorized for it. On the contrary, if the higher role does not exist, the user looses the authorization for the role r. This implies that r must leave all sessions of the user. Moreover, all roles that are transitively authorized by the deleted assignment have to be deactivated from all the user sessions unless user u gets the authorization from another role of which u is a member. The left-hand side of the rule `remove from role with loss of authorization` requires that the user not be a member of a higher role in the hierarchy, as specified by the negative application condition indicated by the dashed upper role. In addition, all roles that inherit the role where the assignment edge is removed (indicated by the roles reachable by a + path) and that are not authorized by other roles (indicated by the dashed lower role) have to be removed from the sessions of the user as well.

activate role and **deactivate role**: A user can activate any role r for which she/he is authorized. A user is authorized for r, if there is a path starting with an assignment edge and ending in r. The corresponding graph rule is `activate role`. An edge between the session node and the role node shows that the role is active in the session. This edge is created by the user of the session. The star $*$ at the edge between the roles indicate a (possibly empty) path through the role hierarchy. An empty path indicates that a user can also activate a role to which she/he is directly assigned. Role r can only join a session if r is not already a member of that session, as indicated by the dashed edge between the session node and the role node. The deactivation of a role from a session is specified by deleting the edge between the session and the role node.

add inheritance and **remove inheritance**: if the administrative role is authorized for two roles r and r', then an administrative role can establish a new inheritance relation between them. The inheritance relation is indicated by an edge between the two roles. The rule `add connection` adds this edge if it does not exist already. The deletion of an inheritance relation may cause a user to loose the authorization for some of his inheriting roles. In particular, a user looses these autorizations if and only if there are no other paths from that user to the role r'. In this case, the roles have to be deactivated from the sessions. The corresponding rule `remove connection` removes edge (r,r') and deactivates the roles of all sessions of all users that do not have the authorization for this role through another path of the role hierarchy (a negative application condition indicated by the dashed objects). Note that an administrator can only add roles within her/his range because a newly created role must be connected to roles for which the administrator is already responsible (see the rule `add connection`).

4 Graph-Based Correctness of RBAC

We now show how the graphical formalism can be used to prove the correctness of a RBAC specification. We clarify the graphical concepts with a running example taken from [GB98]. There, a set of properties is given that defines the consistency requirement for a RBAC database. A state of the RBAC database having these properties is consistent. One property out of this set is

$$(1)\ \forall u \in USERS, \forall r_1, r_2 \in ROLES, r_1, r_2 \in active\_roles(u) \Rightarrow (r_1, r_2) \notin dsd.$$

The *dynamic separation of duties* (*dsd*) is a relation on roles. Roles related by a *dsd* relation must not be active in the same session of a user at the same time.

In [GB98], the semantics of the basic operations of the RBAC model is specified using both set theory and additional logical conditions. It is also shown that a given RBAC specification is correct in the following sense: if the RBAC database is in a consistent state, then the database remains in a consistent state after the operation is performed. The specification of the *activate roles* operation in [GB98] (where it is called *addActiveRoles*) is reported below. Note that to prevent the operation *activate roles* from violating the property (1) an additional logical condition must be used.

addActiveRoles
Arguments:
user,roleset
Semantics:
$active\_roles' = (active\_roles \setminus$
$\{user \mapsto active\_roles(user)\}) \cup$
$\{user \mapsto active\_roles(user) \cup roleset\}$
Conditions:
$user \in USERS$
$roleset \subseteq authorized\_roles(user)$
$\forall r_1, r_2 \in roleset \cup active\_roles(user).(r_1, r_2) \notin dsd$

Hence in [GB98], the designer has to perform three steps: 1. define the consistency properties on the entire system, 2. derive from step 1 the conditions for each operation, and 3. prove that the execution of each operation, satisfying the condition in step 2, preserves the consistency properties defined in step 1. In contrast, in our approach the designer has to perform step 1 only, that is, the definition of the consistency properties of the system. The derivation in step 2 of the conditions for each operation from the consistency properties can be performed automatically following a theoretical construction proposed in [HW95]. The result of such an automatic construction is a set of graph rules which is *guaranteed* to satisfy the given consistency properties and therefore the complex proofs of step 3 are not needed anymore. Our approach also presents some advantages when the consistency properties must be modified in a system already specified. In such a case the designer can define a new consistency property and can use the automatic construction to change the existing rules for each operation to satisfy the new consistency properties. In [GB98], such an incremental modification requires a human intervention in all three steps described above.

In the following, we explain the automatic construction of the condition of an operation to guarantee the consistency properties, by discussing as an example the activate role operation. Consistency properties are specified by *graphical constraints*. A graphical constraint is a graph that depicts a forbidden or a required structure. A graph is *consistent* w.r.t. a graphical constraint that forbids (requires) a structure, if this structure does not occur (does occur) in the graph. The property (1) is specified by the graphical constraint in Fig. 5. The graphical constraint shows the structure that must not occur.

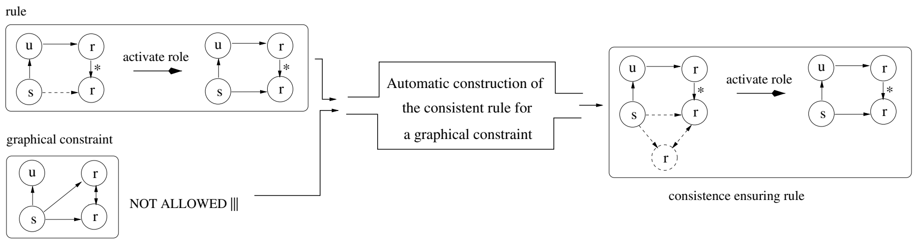
rule
u
r
s
r
activate role
*
graphical constraint
NOT ALLOWED |||
Automatic construction of the consistent rule for a graphical constraint
consistence ensuring rule

Fig. 5. *Automatic construction of a consistent rule from a graphical constraint.*

The double-headed edge between two roles specifies that the two roles are in the *dsd*-relation. On the left-hand side of the figure, rule *activate roles* from Fig. 4 is shown. This rule, if applied without modification, could produce an inconsistent state by creating an edge between a session node and a role which is in *dsd*-relation to another role of the session. Therefore, this rule has to be completed by adding a negative application condition which prevents the activation of incompatible roles. The negative application condition can be constructed automatically during the design phase. Figure 5 shows the overall scheme for this construction. We call the modified rule *consistent* w.r.t. the graphical constraint in the sense of the following theorem:

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.

In the example in Fig. 5, the automatic construction adds a negative application condition to the rule. This condition forbids the rule application if an edge exists between the session node and an active role (modeled by the dashed role and the edge connected to the session node) in *dsd*-relation with the role that shall be activated. This rule can never create a state graph that violates the graphical constraint for the *dsd*-relation and therefore an "a posteriori check" is not needed. In a similar way, all the properties defined in [GB98] can be modeled by graphical constraints and for each of them a consistent rule can automatically be derived.

5 Decentralized Administration of Roles

There are some open problems in the decentralized administration of roles of the RBAC model stemming from the fact that the responsibility of a range of roles is not sufficient to guarantee the desired effect of an action performed inside the range by the administrative role for that range. To illustrate the problems, consider the concrete role hierarchy in Fig. 6 in the following discussion.

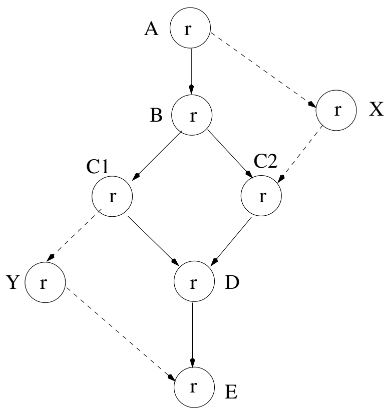

Fig. 6. *Problems of the RBAC model.*

Deletion of a user from a role: if a user is a member of roles $C1$ and B, by inheritance, the user is authorized for roles $B, C1, C2, D, Y$ and E. An administrative role ar with range $C1, D$ can remove the user from role $C1$. However, since the user is still a member of B, by inheritance he still has the permission of $C1$. Since the role B is not in the range of the administrator ar, he cannot remove the membership of the user from role B. The problem of this kind of user assignment revocation is called *weak revocation* [San98]. The case where the user must loose the authorization to the role if the assignment edge is deleted (*strong revocation*) may be more desirable but it requires a greater administrative effort.

Deletion of a permission from a role: This problem is dual to the previous one. 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: In the RBAC model, the range for an administrator is given by an interval over the partial order. Deleting the boundaries of the interval destroys the range definition, therefore 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. For example, if we require a partial order, an edge from E to A in Fig. 6 creates a cycle. Similarly if each range must have unique senior and junior roles (as in Fig. 6, B is the unique senior and D is the unique junior of the diamond range), the addition of the node X and an edge from X to $C2$ violates the constraint of the unique senior. The insertion of an edge (from $C2$ to $C1$) may also create a dependency between previously unrelated roles (X and Y), and therefore a local change by an administrator may have impact outside his/her range.

We propose now three models using graph transformation that tackle the first three problems. We introduce here only the rules for the user assignment; the rules for the assignment of permissions are similar. We will discuss in Sec. 6 how to deal with the last problem exploiting graph transformation concepts.

5.1 Static Single Assignment

The idea is to have at most one assignment per user, i.e. a user can be in at most one role. The assignment is static in the sense that it can only be changed by deleting it and inserting a new one. The deletion of the assignment implies the loss of the authorization for roles that was given by the assignment edge. The graph rules for the static single assignment are shown in Fig. 7. A user can be assigned to a role if he is not yet a member of a role. Membership is indicated by the color of the u node. A white u node indicates that the user is not yet in a role; a black one indicates that he is a member of a role. The rule `add to role` in Fig.7 changes the white user node to a black one when it sets the assignment. The removal of the assignment is simpler than in the model of Fig. 4, since the authorization of roles for this user is known, namely all roles reachable from the unique assignment edge. The rule `remove from role` removes the assignment edge and deactivates all sessions of the user. All sessions are deactivated since all activated roles for a session are authorized by the one assignment edge. There cannot be roles activated that are not authorized by this assignment. The user node is changed to white again. Moreover, the case where

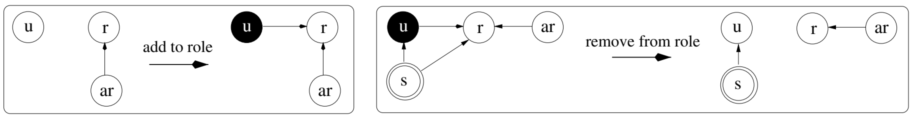

Fig. 7. *Graph rules for static single assignment.*

the removal of the assignment does not have an impact on the authorization is not possible in this model. Therefore, we need no rule corresponding to `remove from role without loss of authorization`. All other rules of the model in Fig. 4 remain unchanged.

This idea could be generalized to more than one "color". The model above uses one "color", namely black. The black color is valid for all roles in the hierarchy graph, i.e. one user can take the black "hat" to be assigned to an arbitrary role in the hierarchy graph. If there are several disjoint hierarchy graphs, each of them has a unique color and we could allow many assignments provided that there is only one assignment for each color. Problems occur in this model, if we want to connect two disjoint hierarchy graphs. Then, the two colors have to be "melt" into one color and assignments may have to be removed.

5.2 Dynamic Single Assignment

The approach of a dynamic single assignment follows the idea of one assignment per user. The rule `add to role` for adding a user to a role is equal to the corresponding rule in the static single assignment model. The difference of this model is that the assignment edge is not static anymore, but it can move through the role hierarchy graph. Only for the lowest role in the role hierarchy it is possible to delete the assignment, since the assignment cannot be moved down anymore. Whenever an administrator wants to assign the user to a role that is higher in the role hierarchy than the current assigned role, the assignment is simply moved up in the hierarchy. The authorization of roles for the user is only enhanced and no changes in the sessions are necessary as modeled by rule `move up`. When an administrator wants to remove a user from a role, the assignment is not deleted, but it is moved down in the role hierarchy with the rule `move down` and only this role has to be deactivated from all the sessions of the user. For the lower role it can again be decided whether the assignment remains or is moved down again. If a user was assigned to a role where there is no inheritance edge to another role, the assignment is deleted and the user is changed to a white node again. All sessions of the user can be deleted as well, since the user has no authorization for any role. This is done by rule `remove role`.

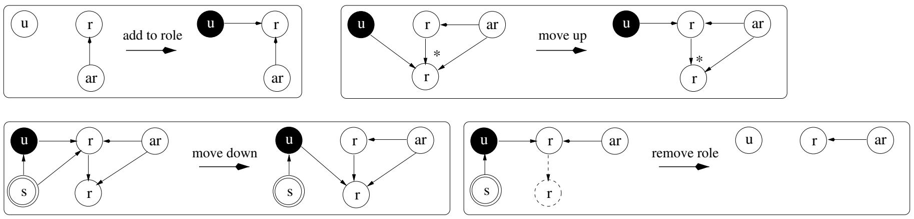

Fig. 8. *Graph rules for dynamic single assignment.*

5.3 (Strong) Multiple Assignment

The idea is to allow an arbitrary number of assignments and to defer the decision on the deletion of an assignment edge to a higher level in the role hierarchy. Any administrator can stop the propagation if she/he wants to keep the assignment and this decision is immediately propagated down. The actual deletion of the assignment can take place only at the top of the role hierarchy: since there is no higher role that may want to keep the assignment, the decision on the deletion can be made. Since we do not require any structure for the hierarchy graph of roles, there can be multiple branches. This means that one branch may decide to delete the role, while another one decides to keep the user in the role. In this case, the assignments of the branch which decided deletion are removed, since the user is not needed in this branch anymore, while at the intersection point the decision to keep the assignment is propagated down. The graph rules for this model are shown in Figure 9.

If an administrator wants to remove an assignment of a user to a role, she/he makes this visible to the administrators of higher roles by a special label. The graph rule `remove from role` inserts the label d on the edge for the assignment. This label can be seen from other administrators. By means of the rule `propagate up` the label d is moved up to the next higher role of which the user is a member. The rule ensures, by the negative application condition, that the label is propagated to the immediately higher role assigned to the user (no roles are skipped). This ensures that all memberships of the user in higher roles are checked. Whenever the label is set on the assignment edge between a user and a role, a responsible administrator can decide if the user shall be removed from the role or kept. The graph rule `veto no deletion` is applied, if the administrator wants to keep the assignment. The label is changed from d to $d-$ and the propagation of the label d to higher roles is stopped (rule `propagate up` is not applicable anymore). If the administrator wants to delete the assignment he propagates the label d by means of rule `propagate up` to a higher role. If the role is the highest role (w.r.t. a branch of the hierarchy) the up-propagation rule is not applicable anymore and the corresponding administrator can decide (beside no deletion, which is possible always) to delete the assignment. This is done by the graph rule `veto deletion` that changes the label d to $d+$. Its negative application condition ensures that the role is the highest role in the hierarchy w.r.t. one branch. After the decision, the label $d-$ or the label $d+$ is propagated down. The rule `propagate veto` $d-$ propagates the label $d-$ and the negative application conditions guarantees again that this propagation does not forget any assignment. The graph rule `propagate veto d+` is more complex, since the label $d+$ cannot be simply propagated down, because there may exist several branches and in some of them the label $d-$ is propagated down, in other ones the label $d+$ is. Since in this model keeping the assignment is stronger than removing the assignment, at an intersection point of two (or more) branches it must be decided which label is propagated further. Only if all the branches want to propagate the label $d+$, $d+$ is propagated further. If only one branch wants to propagate $d-$, $d-$ is propagated further. This condition is checked in the negative application condition of rule `propagate veto` $d+$. Only if there is no higher role with a $d-$ or a d label at an assignment, the $d+$ label is propagated by rule `propagate veto` $d+$. The step-wise propagation guarantees the second negative application condition. If the labels $d-$ and $d+$ are present, the graph rules `delete` and `no delete` can be applied. The rule `delete` is applicable if there is a label $d+$ and it removes the assignment edge, also deactivating the role from all the sessions of the user. This rule is also applied to branches of the hierarchy that have decided for deletion, even if the label $d+$ was not propagated completely down since other branches had decided to keep the assignment. When a label $d-$ is encountered, indicating that the assignment shall be kept, the label is simply deleted but the assignment remains.

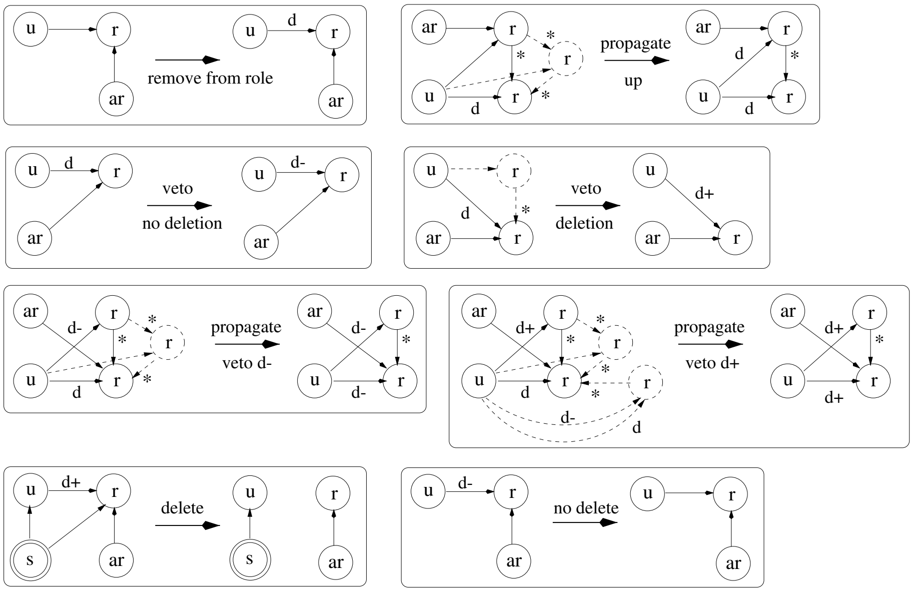

Fig. 9. *Graph rules for multiple assignment.*

6 Comparison of Proposals for Decentralized RBAC

The RBAC model has been described with the rules in Fig. 4 of Section 3 already in a setting that allows multiple administrators (explicit nodes of type ar). In the previous section, we have presented three graph specifications of the user-role assignment and deletion in the case of decentralized administration of roles. We discuss now how the four proposed specifications solve the issues of the RBAC model mentioned at the beginning of the previous section.

Deletion of a user from a role: The graph model in Fig. 4 can be seen as a *weak multiple assignment* model, because it models weak revocation and allows many assignments for a user. The three models proposed in Section 5 specify a strong revocation. Different assumptions are made to simplify the administrative effort. Adding an assignment is easy in all models. In the case of many assignments, no restrictions are needed for creation. In the case of a single assignment, creation is easy, since one has to check only, whether the user is already a member of a role or not. The main differences occur in the deletion of assignments. The single assignment models provide an easy deletion, since there is only one assignment and all authorization relations for the user are known. In the static case, the assignment is simply deleted. The many assignment models are more complex w.r.t. deletion of assignments, since the authorization relations for a user are not known. A more complex procedure is necessary to decide if the deletion of an assignment requires the deletion of other assignments or not. Altogether, the multiple assignment models are more flexible, but have a greater administrative effort. The following table summarizes the results.

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

Deletion of a permission from a role: This problem can be solved in a manner similar to the deletion of user from a role, by replacing user assignment with permission assignment.

Deletion of roles: This problem is solved by modeling the range of an administrator by a set of edges pointing to the roles the administrator is responsible for. By replacing the interval definition of a range in [SBM99] by a set definition, the deletion of a boundary role does not destroy the range of an administrator.

Comparing our model with [NO99], the algorithms presented there deal with the centralized administration of privileges (permissions) and roles. By explicitly introducing permission nodes, we could specify their model. For example, the deletion of a role while retaining its privileges could require the redirection of the edges to its permission nodes to other role nodes with a set of rules mimicking their algorithms.

Special hierarchy graphs: Our model does not require any structure for the hierarchy graph for (administrative) roles. Therefore, there are no special rules for maintaining a graph structure. If a special graph structure is required, the rules have to be constructed in such a way that they do not destroy the structure. Since these rules depend on the structure required, no general rules can be given. However, this problem could be solved by considering *graphical constraints* [HW95], which are a graphical means to express constraints like cardinality, mutual exclusion, prerequisite roles etc, and in particular, they can be used to define the desired structure of the hierarchy graphs. Constraints are also an important component of the RBAC model. Different constraints yield different access control models (see [San98,SBM99]). Graphical constraints can be automatically translated into negative application conditions for rules. The rules ensure the consistency w.r.t. the graphical constraint. That means for example, that the rules preserve the required graph structure.

7 Concluding Remarks

We propose a formalization of RBAC using graph transformations which is a graphical specification technique giving an intuitive visual description of the dynamic structures that occur in AC models. The use of graph structures allows a uniform treatment of user roles and administrative roles without the need for a meta-model to describe possible evolutions in the administrator structure. This formalization of the RBAC model can benefit from well-established results in graph transformations systems [Roz97]. Among them, the possibility of studying the sequential independence of rules (i.e. the final state is the same regardless of the order of application of two rules), the parallel application of rules (i.e.

the simultaneous application of two rules to produce an effect not attainable by their separate application), and in general the interference of rules.

Our approach is suitable for the specification and verification of the consistency requirement, as in [GB98], with the added feature of being able to systematically derive the new rules to reflect changes in the consistency conditions due to the evolution of an already designed system. Given a concrete specification (e.g., an assignment of names to roles and users) of a particular role-based system, properties can be verified using tools for graph transformations.

In [NO99], the authors present a specific way of implementing role-role relationships, representing as a graph the inclusion hierarchy generated by (part of) the powerset of the set of privilegies. Here graph transformations are used as a general formalism to specify access control policies based on roles. No specific assumptions are made on the (arbitrary) structure of the role graph: if a particular structure is required, the graph rules can be adapted to satisfy the additional requirements. The algorithms there deal with the centralized administration of roles, while we focus on a decentralized administration of roles. By explicitly introducing permission nodes, we could specify their model. For example, the addition of a role node could require that it be explicitly connected to all the nodes representing its permissions (called effective privilegies in [NO99]); the connection with other role nodes could be defined by a set of rules mimicking their algorithms, instead of being an arbitrary choice of the administrator as in `add connection` of Fig. 4

We also discuss and compare here several possible solutions to some of the issues left open in [San98]. In particular, we address the revocation cascade of users membership when administrative roles are decentralized, the out-of-range impact of local changes, and the removal of endpoint roles in an administrative range.

The proposed general framework is adequate for (but not restricted to) role-based access control policies. We are developing a methodology to compare different access control models within the graph grammar formalism and to analyse the effect of combining different access control policies [KMPP00]. A tool is under development to assist the systematic modification of rules in a system where the consistency condition may change.

Acknowledgement. Thanks to the anonymous reviewers for their pointed comments and usefull suggestions.

References

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. II: Applications, Languages, and Tools.* World Scientific, 1999.

[GB98] Serban I. Gavrila and John F. Barkley. Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. In *Proc. of 3rd ACM Workshop on Role-Based Access Control*, 1998.

[HW95] Reiko Heckel and Annika Wagner. Ensuring consistency of conditional graph grammars - a constructive approach. In *Proc. of SEGRAGRA'95 Graph Rewriting and Computation*, number 2. Electronic Notes of TCS, 1995. http://www.elsevier.nl/locate/entcs/volume2.html.

[KMPP00] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. Technical Report SI-2000/05, Dip.Scienze dell'Informazione, Uni. Roma La Sapienza, May 2000.

[NO99] M. Nyanchama and S.L. Osborne. The Role Graph Model and Conflict of Interest. *ACM Trans. of Info. and System Security*, 2(1):3–33, 1999.

[PEM87] F. Parisi-Presicce H. Ehrig and U. Montanari. Graph Rewriting with Unification and Composition. In *Proc. 3rd Workshop on Graph Grammars. Lecture Notes in Computer Science 291*, Springer-Verlag, 1987, pp.496-514.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations. Vol. I: Foundations.* World Scientific, 1997.

[San98] Ravi S. Sandhu. Role-Based Access Control. In *Advances in Computers*, volume 46. Academic Press, 1998.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security*, 2(1):105–135, Feb. 1999.