Administrative Scope in the Graph-based Framework

M. Koch*
Institut für Informatik
Freie Universität Berlin
Berlin, Germany
mkoch@inf.fu-berlin.de

L. V. Mancini†
Dipartimento di Informatica
Università di Roma
Roma, Italy
lv.mancini@di.uniroma1.it

F. Parisi-Presicce‡
Dept of Info. and Soft. Eng.
George Mason University
Fairfax, VA USA
fparisip@gmu.edu

ABSTRACT

The use of the graph-based framework to specify the administration of RBAC systems has several advantages, from the intuition provided by the visual aspect to the precise semantics and the systematic verification of constraints. Here the benefits of this framework are illustrated using SARBAC (scoped administration of role based access control), providing the first steps towards its operational semantics and a more expressive constraint language.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access control*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Verification

Keywords

role-based access control, administration, graph transformations

1. INTRODUCTION

While role based access control (RBAC) models have been widely investigated ([4, 9, 10]), the use of RBAC to specify the administration of RBAC systems has received less attention. One of the models for the administration of RBAC96 is ARBAC97 ([11]). In the ARBAC97 model, role management, that is, the creation and deletion of roles as well as the assignment and the revocation of users and permissions to roles, is the responsibility of *administrative roles*. The role hierarchy is given by a partial order over roles, and the administrative role hierarchy is given by a partial order over roles based on interval inclusion. The range of an administrative role is represented by an interval over the first partial order and the ranges of junior administrators are required to be subranges of the senior administrators range. The approach proposed in [11] is not particularly suited to managing dynamic changes in administrator's ranges, as discussed in [6, 5], where solutions to the revocation and to other problems are presented in detail using the graph-based framework.

A different approach is proposed by Crampton and Loizou in [2], with the concept of administrative scope in role hierarchies. The authors propose a family of models for role administration and define a decentralized model for role-based administration, called SARBAC (scoped administration of role-based access control). It is shown in [2], that the SARBAC model has several advantages compared to the ARBAC97 model.

We have presented a graph-based security framework for the specification of access control policies in [6], where several models to decentralized role administration are also proposed. In this article we show how to express the concept of administrative scope in the graph-based security framework. Beside a visual and (subjectively) a more intuitive presentation, the SARBAC model can benefit from the graph-based specification in several ways:

1. As stated in [2], a "priority is to complete the SARBAC model by incorporating administration of separation of duty constraints into SARBAC". The graph-based security framework comprises a graphical constraint language for access control constraints. This language can be used to specify the administration of constraints, e.g., separation of duty constraints.

2. In addition, [2] states that the authors "intend to give an operational semantics for SARBAC by writing pseudo code functions to implement the SARBAC operations". The specification of the SARBAC operations by graph rules gives an operational semantics to the SARBAC operations. The specification based on graph rules can be executed using graph transformation tools [3].

3. Graph transformations provide theoretical results to verify the specified access control constraints with respect to the SARBAC model operations.

4. The specification of role hierarchy operations of the SARBAC model by graph transformations helps in the maintenance of a consistent role hierarchy. The maintenance of the consistency of the role hierarchy due to side-effects of the operations on the role hierarchy is more difficult in the presentation given in [2].

*supported by the European Union under RTN SegraVis
†partially supported by the Italian MIUR under FIRB WEB-MINDS project and PRIN 2003 project: "WEB-based management and representation of spatial and geographic data.
‡supported in part by the European Union under RTN SegraVis (http://www.segravis.org)

The remainder of the article is organized as follows: Section 2 gives a brief overview on Graph Transformations. Section 3 reviews the concept of administrative scope and introduces the RHA_4 model and its specification by graph transformations. Section 4 considers the SARBAC model and specification of this model by graph transformations. In Section 5 the benefits of a graph-based specification for the SARBAC model are discussed. Section 6 concludes the article.

2. BACKGROUND ON GRAPH TRANSFORMATIONS

We introduce the graph transformation concepts necessary in this article. A more detailed introduction can be found in [8].
A *graph* consists of a set of nodes and a set of directed edges. An edge points from the *source* node to the *target* node. Nodes and edges carry labels from disjoint sets of variables and constants. In the figures, node labels are written inside the node, edge labels are attached to the edge. The example graph in Figure 1 consists of four nodes and three edges. The labels of the nodes are a, $R1$, $R2$ and u, where lower case letters are used for variables (u, a) and upper case letters for constants ($R1$,$R2$). The intended meaning of the example graph is that there are two fixed roles $R1$ and $R2$ that are administrated by a variable administrator. A variable user u is assigned to role $R2$. The edges in the example do not have labels.

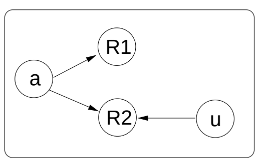

Figure 1: *Graph*

Graphs are related by *graph morphisms*. A *total* graph morphism $f : G \to H$ between graphs G and H maps the nodes of G to nodes of H and the edges of G to edges of H. A graph morphism must respect the graph structure, i.e., the source (target) node of an edge e in G is mapped to the source (target) node of the edge in H to which e is mapped by f. In addition, f relates nodes and edges with a constant label to nodes and edges of the same label. Variables can be related to any label. A *partial* graph morphism $f : G \to H$ is a total graph morphism $\bar{f} : dom(f) \to H$ from a subgraph $dom(f) \subseteq G$ to H. A graph morphism is *injective* if the mappings between nodes and edges are injective. Figure 2 is an example of an injective partial graph morphism. The presentation of graph morphisms, i.e., the mappings for nodes and edges, are represented by the position of nodes in the graphs: if a node n has the same position in G and H, then there is a mapping for n from G to H. In the example, $R1$ in G is mapped to $R1$ in H, $R2$ to $R2$, a to $A1$ (possible since variables can be mapped to constants), and u to u. The edge $a \to R1$ in G is mapped to the edge $A1 \to R1$ in H (similarly for the edge $a \to R2$). The edge $u \to R2$, however, is not mapped to H, i.e., the mapping is partial on edges.

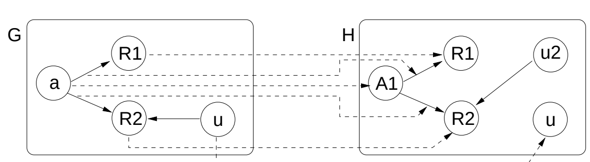

Figure 2: *Injective Partial Graph Morphism*

Graphs can be manipulated by *graph rules* consisting of an injective partial graph morphism $r(p_1, .., p_n) : L \to R$, where $p_1....p_n$ are variables for nodes in L and R. The *left-hand side* L of a *rule* describes the elements a graph must contain for p to be applicable. The partial morphism r is undefined on nodes/edges that are intended to be deleted, defined on nodes/edges that are intended to be preserved. Nodes and edges of R, *right-hand side*, without a pre-image are newly created. The application of a rule r to a graph G requires a total graph morphism $m : L \to G$, called *match*, and the direct derivation of r at m is done in two steps: first delete from G all the elements in $m(L \setminus dom(r))$, then add to the result all the elements in $R \setminus r(L)$.
An example of a graph rule *rule(a,p,c)* and its application to a graph is shown in Figure 3. The left-hand side L of the rule consists of

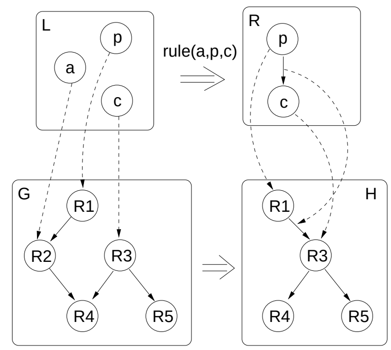

Figure 3: *Application of a graph rule.*

three nodes carrying the variables a, p and c (variables are lower cases). The rule is defined for the p and c node and undefined for the a node (i.e., this node shall be deleted). The right-hand side R has an edge between nodes p and c. The match for the rule in graph G maps the a node to node $R2$, p to node $R1$ and c to $R3$. The application of the rule removes the node $R2$ (together with all connected edges) and adds the edge between $R1$ and $R3$.

The application of graph rules may be restricted by *negative application conditions* (NAC). A NAC prevents the rule application in distinguished cases even if the left-hand side can be found in a graph. A NAC for a rule $r(p_1, ..., p_n) : L \to R$ consists of a set $A(p)$ of pairs (L, N), where the graph L is a subgraph of N. The part $N \setminus L$ represents a structure that must not occur in a graph G for the rule to be applicable. In the figures, we represent the pair (L, N) with N, where the subgraph L is drawn with solid lines and $N \setminus L$ with dashed lines. Figure 4 shows the graph rule `rule(a,p,c)` of Figure 3 with an additional NAC consisting of one edge from node p to node a. A rule $r(p_1, ..., p_n) : L \to R$

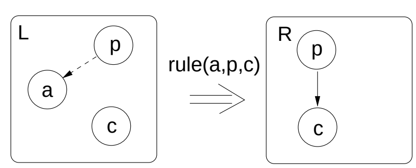

Figure 4: *Graph rule with NAC.*

with a NAC $A(p)$ is applicable to G if L occurs in G via m and it is not possible to extend m to N for each (L, N) in $A(p)$. The rule `rule(a,p,c)` with NAC is not applicable at the match given in Figure 3, since the forbidden edge between the nodes p and a can be found between the nodes $R1$ and $R2$. The rule is applicable to G via the match associating a to $R3$, p to $R1$ and c to $R5$.

Graph rules specify in an operational way how to build the accepted graphs starting from a given start graph. *Graphical constraints* give the possibility to specify declaratively which graph structures are required or forbidden in accepted graphs. A *simple graphical constraint*, or just *constraint*, is given by a total morphism $x : X \rightarrow Y$ and can be either *negative* or *positive*. A graph G ***satisfies*** a positive constraint x_{pos} if for all morphisms $p : X \rightarrow G$ there is a morphism $q : Y \rightarrow G$ with $q \circ x_{pos} = p$. A graph G ***satisfies*** a negative constraint x_{neg} if for all morphisms $p : X \rightarrow G$ there does **not** exist a morphism $q : Y \rightarrow G$ with $q \circ x_{neg} = p$. Figure 5 shows a negative constraint which forbids an edge between a node labeled with the constant T and a node r. The graph $G1$ does not satisfy the constraint, since there is an edge between node T and $R1$. The graph $G2$, however, satisfies the negative constraint.

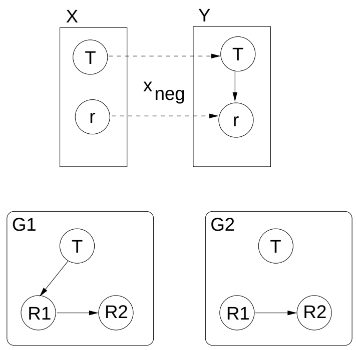

Figure 5: ***Simple negative graphical constraint.***

A conditional constraint $c = (n, x)$ consists of an injective total morphism $n : X \rightarrow N$ and a simple positive constraint $x : X \rightarrow Y$. The morphism n gives the condition under which the constraint x must be satisfied. A graph G ***satisfies*** c if for all total morphisms $p : X \rightarrow G$ for which there is not total morphism $a : N \rightarrow X$ with $a \circ n = p$, there is a total morphism $q : Y \rightarrow G$ with $q \circ x = p$. Figure 6 shows an example. The lower part of the figure depicts the presentation of conditional constraints used in the rest of the article: we show the graph N in which the parts $N \setminus n(X)$ are drawn by dashed lines, the elements of $n(X)$ by solid lines. The conditional constraint specifies that there must be an edge between node T and r whenever T is not connected to another node r_z. Graph $G1$ of Figure 5 satisfies the constraint. If we consider T and $R1$, the condition of the constraint is satisfied and we have to check the positive constraint that requires an edge between T and $R1$. This edge exists. If we consider T and $R2$, the condition of the constraint is not satisfied, since we can find another node r connected to T. Therefore, we do not have to check the positive constraint. On the other hand, graph $G2$ does not satisfy the conditional constraint, since there is no edge from T at all.

To express a more complex behavior than that describable with simple rules, it is necessary to have ways to combine them. Rules can be combined in *rule expressions*, with which it is possible to specify the application of rules in a prescribed order. A rule expression $ruleExpr$ is a term generated by the following syntax, where $ruleNames$ is a set of ruleNames and $r \in ruleNames$. The $guard_i$ are conditional or simple constraints.

$ruleExpr$::= [**not**]$guard_1$ **and ....and** [**not**] $guard_n$: $expr$
$expr$::= r | $expr1$; $expr2$ | **as long as possible** r

The rules in $expr$ can be applied to a graph G if G satisfies all

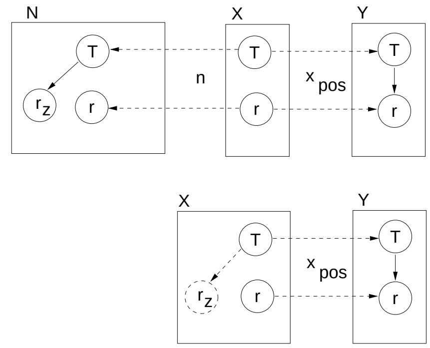

Figure 6: ***Conditional constraint.***

guards. A guard may be negated by the keyword **not** which requires that the guard is not satisfied. The term $expr1$; $expr2$ specifies a sequential application of two expressions and the intended meaning of **as long as possible** r is the application of rule r to G as long as r can be applied. Examples of rule expressions will be shown in the next sections.

3. ADMINISTRATIVE SCOPE

Crampton and Loizou present in [2] the concept of administrative scope in a role hierarchy and a role-based administrative model based on administrative scope (SARBAC). They show the advantages of their model compared to ARBAC97 [11]. This section shows that the SARBAC model can be expressed in the graph-based framework used to specify RBAC models in [6, 5]. In the process, we extend the concepts in [6] to include rule expressions. In [2] a family of role hierarchy administration (RHA) models is presented. The most complex of them, called RHA_4, is the basis for the SARBAC model. We introduce next the RHA_4 model and give a specification by graph transformations.

3.1 The RHA_4 model

The RHA_4 model assumes a role hierarchy, that is, a partial ordered set of roles $(R, \leq)$. We define $\uparrow x = \{r \in R | x \leq r\}$ and $\downarrow x = \{r \in R | r \leq x\}$. In addition, there is a binary relation *admin-authority* $\subseteq R \times R$ that specifies the roles that a role controls: a role a controls a role r if $(a, r) \in$ *admin-authority* and $C(a) = \{r \in R | (a, r) \in admin - authority\}$ denotes the set of roles controlled by a. In [2] is required, that, if $(a, r) \in$ *admin-authority* then $a \not\leq r$, *admin-authority* is antisymmetric, and each role $r \in R$ is controlled by at most one administrative role.

The administrative scope of a role r is given by

$$S(a) = \{r \in R | \uparrow r \setminus \uparrow C(a) \subseteq \downarrow C(a)\}.$$

Informally, a role r is in the administrative scope of the role a, if each path upwards from r goes through a role controlled by a. We define $S^+(a) = S(a) \setminus C(a)$.

Figure 7 shows an example of a role hierarchy (taken from [2]). The administrative role $PS01$ controls role $PL1$, DSO controls $PS01$ and DIR. Then, the administrative scope of DSO contains all roles in the hierarchy including $PS02$, the administrative scope of $PS01$ contains the roles $S(PSO1) = \{ENG1, PE1, QE1, PL1\}$.

In [2], there are the following role hierarchy operations: *AddRole(a,r,Δr, ∇r)*, *DeleteRole(a,r)*, *AddEdge(a,c,p)* and *DeleteEdge(a,c,p)*, where a is the administrative role which initiates the operation, Δr is the set of immediate children of the new role r, ∇r is the set of immediate parents of r, c is the child role, and p the

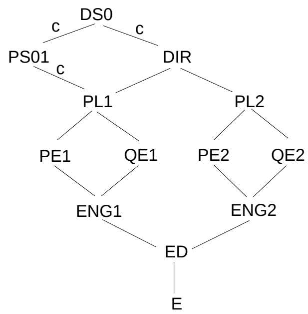

Figure 7: ***Role Hierarchy.***

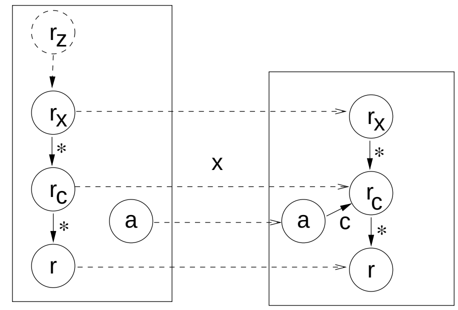

Figure 9: ***Conditional Constraint for administrative scope with admin-authority.***

parent role. The transitivity of the partial order must be preserved by these operations: If an edge is deleted from the role hierarchy, edges that had previously been implied by transitivity may need to be added. If an edge is added, transitive edges may need to be deleted. Role deletion has the side effect of inserting transitive edges that would be lost and role insertion deletes any transitive edge that arises.

The *admin-authority* relation in the RHA_4 model is dynamic. It can be changed indirectly (as side effect of role hierarchy operations) or directly by the operations *AddAdminAuthority(a,a',r)* and *DeleteAdminAuthority(a,a',r)* which add and remove the tuple (a', r) respectively to and from the *admin-authority* relation.

3.2 RHA_4 by graph transformations

We present next the specification of the RHA_4 model by graph transformations. We specify the role hierarchy by a graph in which an edge $r \rightarrow r'$ specifies $r' \leq r$. An edge $r \xrightarrow{*} r'$ specifies a (possibly empty) path through the

role hierarchy from r to r'. An edge $r \xrightarrow{+} r'$ specifies a non-empty path through the role hierarchy from r to r'. The *admin-authority* relation is modeled by an edge $a \xrightarrow{c} r$ for each pair $(a, r) \in$ *admin-authority*. Figure 8 shows the role hierarchy of Figure 7 as a graph.

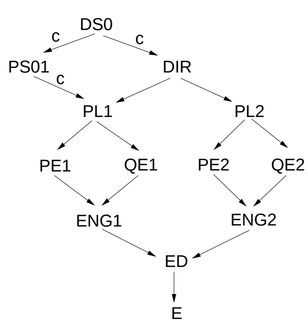

Figure 8: ***Role Hierarchy Graph.***

The property of administrative scope can be expressed by the conditional constraint $adScope(a, r) = (n, x)$ in Figure 9. It requires that each maximal path starting from r up the hierarchy includes a role r_c controlled by role a. The positive constraint x of $adScope(a, r) = (n, x)$ requires that there must be a node in the path from r_x to r which is controlled by a. This is specified by the required c-labeled edge from a to node r_c. To ensure that only paths of maximal length are considered, the negative part n of $adScope(a, r)$ (the dotted role r_z) forbids a predecessor role for the role r_x. Then, a role r is in the administrative scope of a if and only if the role hierarchy graph satisfies the conditional constraint $adScope(a, r)$ for r and a. Consider as an example the administrative role $PS01$ and the roles $PE1$ and $PE2$ for which we check $adScope(PS01, PE1)$ and $adScope(PS01, PE2)$, respectively. Each path upwards from $PE1$ includes the role $PL1$, which is controlled by $PS01$. Therefore, the conditional constraint is satisfied and $PE1$ is in the administrative scope of $PS01$. By contrast, the paths upwards from $PE2$ include the roles $PL2$ and DIR, neither one controlled by $PS01$. Therefore, the conditional constraint is not satisfied for $PS01$ and $PL2$.

If the roles r and r_c in the conditional constraint $adScope$ in Figure 9 were required to be distinct, we label the edge $r_c \rightarrow r$ by $+$ instead of $*$ to require a non-empty path from r_c to r. This conditional constraint is denoted by $adScope^+$.

Next, we present the rule expressions for the role hierarchy operations *AddRole(a,r,Δr, ∇r)*, *DeleteRole(a,r)*, *AddEdge(a,c,p)* and *DeleteEdge(a,c,p)*. The following two rule expressions specify the insertion and the deletion of a role to and from the role hierarchy.

- addRole(a,r,Δr,∇r):
  $adScope(a, \nabla r)$ **and** $adScope^+(a, \Delta r)$:
  addRole; **as long as possible** clean

- deleteRole(a,r):
  $adScope^+(a, r)$:
  complete(r); deleteRole

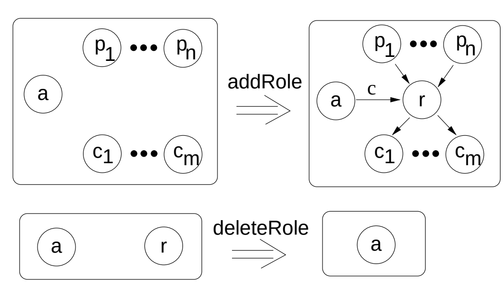

Figure 10: ***Graph rules for administrative operations for roles.***

The guard $adScope(a, \nabla r)$ **and** $adScope^+(a, \Delta r)$ of the rule expression addRole(a,r,Δr,∇r) requires that the conditional constraint $adScope$ be satisfied for the node a and each node $p_i \in \nabla r$, and that the graphical constraint $adScope^+$ be satisfied for the node a and each node $c_i \in \Delta r$. Then, the graph rule *addRole* in Figure 10 is applied once. The graph rule inserts a new role r between the parent roles $\nabla r = \{p_1, ..., p_n\}$ and the children roles $\Delta r = \{c_1, ..., c_m\}$ and connects them by edges. In the figure, we use the dots between nodes p_1 and p_n as well as c_1 and c_m as an abbreviation for the sequence of nodes. The graph rule connects the new role r and the administrative role a with a c-labeled edge, that is, the pair (a, r) is added to *admin-authority*. After the new role r is inserted by the rule *addRole*, the graph rule

clean in Figure 11 is applied as long as possible to the role hierarchy. The rule *clean* removes an edge between two roles r_1 and r_2 if there is a path between them through another role r. The rules are applied also to c-labeled edges.

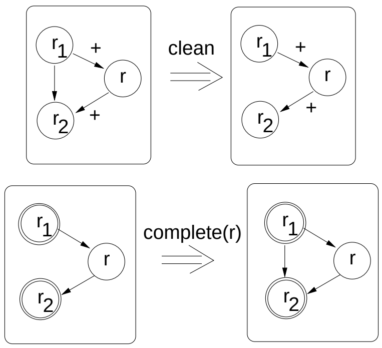

Figure 11: ***Rules to ensure a non-redundant and transitive role hierarchy.***

The guard $adScope^+(a, r)$ of the rule expression *delete-Role(a,r)* ensures that r is in the administrative scope of a. Then, the graph rule *complete(r)* in Figure 11 is applied once. This graph rule ensures that all transitive relations are preserved between the parent and children nodes of r (again, also c-labeled edges are considered by this rule). The graph rule $complete(r)$ adds an edge between each parent node and each child node of the role r. The double circle around r_1 and r_2 specifies the complete set of parents of r (i.r., ∇r) and children of r (i.e., Δr), respectively. The edges to and from r are not deleted by this rule. After the rule $complete(r)$, the graph rule *deleteRole* in Figure 10 is applied to delete the role r (all the edges incident to the node r are automatically deleted).

The following two rule expressions specify the insertion and the deletion of an edge between two roles in the role hierarchy.

- addEdge(a,c,p):
  $adScope(a, c)$ **and** $adScope(a, p)$:
  addEdge; **as long as possible** clean

- deleteEdge(a,c,p):
  $adScope(a, c)$ **and** $adScope(a, p)$:
  complete(c); complete(p); deleteEdge;
  as long as possible clean

The guard $adScope(a, c)$ **and** $adScope(a, p)$ checks if the parent and children role of the edge are in the administrative scope of the administrator role a. If this is true, the graph rule *addEdge* in Figure 12 adds the edge between node p and c. Then, the rule *clean* in Figure 11 is applied as long as possible to remove edges not necessary due to transitivity.

The rule expression *deleteEdge(a,c,p)* has the same guard as the expression *addEdge(a,c,p)* to ensure that p and c are in the administrative scope of a. The successive applications of the rules *complete(c)* and *complete(p)* ensure that all the edges implied by the transitivity of the deleted edge are added. After these edges are added, the edge between p and c is deleted by the graph rule *deleteEdge* in Figure 12.

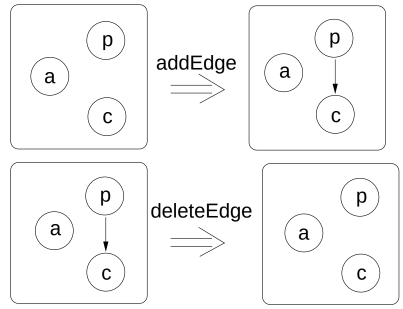

Figure 12: ***Graph rules for administrative operations for edges.***

We consider now the modification of the relation *admin-authority*, which we introduced by c-labeled edges in the role hierarchy graph. First we consider direct updates by the operations *AddAdminAuthority* and *DeleteAdminAuthority*.

- addAdminAuthority(a,a',r):
  $adScope(a, r)$ **and** $adScope(a, a')$ **and**
  not $adScope(a', r)$:
  addAdminEdge; **as long as possible** clean

- deleteAdminAuthority(a,a',r):
  $adScope(a, r)$ **and** $adScope(a, a')$:
  complete(a'); deleteAdminEdge

The guard of the rule expression ***addAdminAuthority(a,a',r)*** checks that the roles r and a' are in the administrative scope of a. Since it is redundant to assign the role a' to a role that is already controlled by a', the guard **not** $adScope(a', r)$ requires that r not be in the administrative scope of a'. If all the constraints of the guard are satisfied, the graph rule *addAdminEdge* in Figure 13 adds a c-labeled edge between the roles a' and r. The rule expression ends with the graph rule *clean* which removes all the edges not explicitly necessary due to transitivity.

The rule expression ***deleteAdminAuthority(a,a',r)*** specifies the deletion of a c-labeled edge between an administrative role a' and a role r. The guard ensures that both a' and r are in the administrative scope of a. The graph rule ***complete(a')*** adds the edges which concern a' but are not drawn explicitely because of transitivity. Afterwards, the graph rule *deleteAdminEdge* in Figure 13 deletes the c-labeled edge between a' and r.

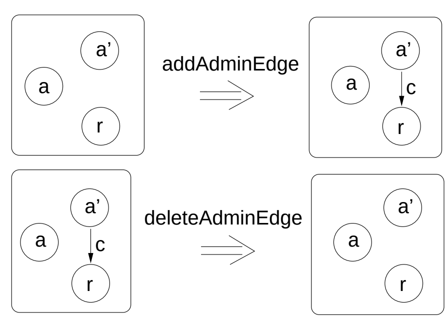

Figure 13: ***Graph rules for changing*** $admin - authority$**.**

The role hierarchy operations may require an indirect update of the *admin-authority* relation. The specification of the rule expressions maintain the administrative scope and eliminate redundancy with the rules *clean* and *complete*. Therefore, no special treatment is necessary for this case. This is different from [2], where a list of indirect updates is given, that is not investigated in terms of exhaustiveness, i.e., whether other indirect updates may be necessary. These investigations are not necessary when graph transformation expressions are considered.

4. SARBAC

This section concerns the specification of the SARBAC (scoped administration of role-based access control) model by graph transformations. SARBAC extends the RHA_4 model to include constraints and relations between users (or permissions) and constraints. The user-role assignment is specified by a binary relation $UA \subseteq U \times R$, where U is the set of users. The permission-role assignment is specified by a binary relation $PA \subseteq P \times R$, where P is the set of permissions.

4.1 SARBAC Constraints

A constraint in SARBAC is just a set of roles. A user satisfies a constraint if she is (possibly indirectly) assigned to all those roles, while a permission satisfies a constraint if it belongs (maybe indirectly via role inheritance) to all those roles. More formally, a constraint in SARBAC is a subset C of the roles

R in the role hierarchy and is denoted by $\bigwedge C$. A SARBAC constraint $\bigwedge C$ is satisfied by a user u if $C \subseteq\downarrow \{r \in R | (u, r) \in UA\}$. The constraint $\bigwedge C$ is satisfied by a permission p if $C \subseteq\uparrow \{r \in R | (p, r) \in PA\}$.

The corresponding graphical constraints are given in Figure 14. These two constraints are simple positive graphical constraints, i.e., they do not have a condition and must be satisfied always. A user

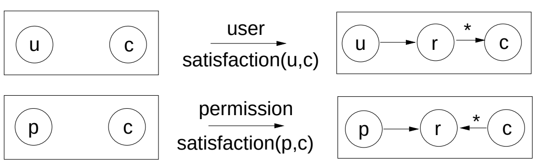

Figure 14: ***Positive graphical constraints for SARBAC constraints.***

u satisfies a SARBAC constraint $\bigwedge C$ for $C \subseteq R$ if the constraint *user satisfaction(u,c)* is satisfied by the role hierarchy graph for each $c \in C$. A permission p satisfies $\bigwedge C$ if the constraint *permission satisfaction(p,c)* is satisfied by the role hierarchy graph for each $c \in C$.

4.2 SARBAC Relations

To model user and permission assignment to and removal from roles, the SARBAC model defines the relations $ua - constraints \subseteq R \times A(R)$ and $pa - constraints \subseteq R \times A(R)$, where $A(R)$ is the set of antichains in R. A set $A \subseteq R$ is an antichain of R, if it contains only unrelated roles, i.e., for all $x, y \in A$, $x = y$ or $x \not\leq y$ and $y \not\leq x$. The roles in the antichain A are a prerequisite for a user to activate a role. An administrative role a can assign a user u (permission p) to a role r if $(r, A) \in ua - constraints$ (resp. $(r, A) \in pa - constraints$) so that u (p) satisfies the constraint $\bigwedge A$ and r is in administrative scope of A. In the sequel, we focus on the relation *ua-constraints*. The permission-role assignment can be modeled in a similar way.

The following rule expressions specify the assign and revoke operations for users. We model the *ua-constraints* relation for each pair (r, A) by edges $r \rightarrow a$ labeled pre_{ua} for each role $a \in A$.

- assignUser(a,u,r):
  $adScope(a, r)$ **and**
  userSatisfaction(u,$\{c \in R | r \overset{pre_{ua}}{\rightarrow} c\}$):
  assign user
- revokeUser(a,u,r):
  $adScope(a, r)$:
  revoke user

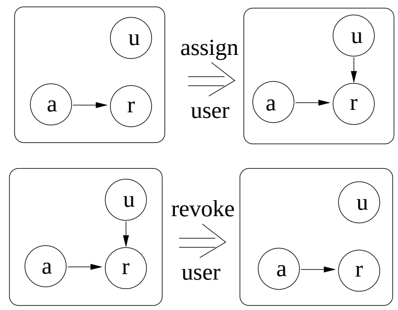

Figure 15: ***Graph rules for administrative operations for users.***

The graph rule *assign user* inserts an edge between the user and the role. This edge specifies the user-role assignment. The guards of the rule expression *assignUser* ensure that the role is in the administrative scope of the user and that u has all the roles required for getting role r. The graph rule *revoke user* removes the edge between the user and the role. The guard ensures that only an administrative role a revokes a user from a role that is in the administrative scope of the administrative role a.

The relation *ua-constraints* can be updated by an administrative role by adding or removing a tuple (r, A) to or from the relation. The following rule expressions specify these operations.

- addUARelation(a,r,c):
  $adScope^c(a, r)$ **and** $adScope^c(a, c)$:
  add(r,c)
- deleteUARelation(a,r,c):
  $adScope^c(a, r)$ **and** $adScope^c(a, c)$:
  delete(r,c)

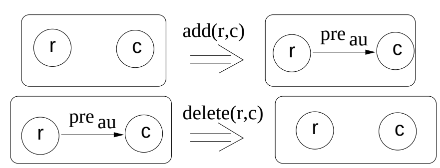

Figure 16: ***Graph rules for direct updates to*** $ua - constraints$.

Thr role hierarchy operations *addRole*, *deleteRole*, *addEdge* and *deleteEdge* may influence the *ua-constraints* relation. For example, the insertion of an edge between two roles in an antichain relates the roles and one role must be deleted from the antichain. In [2] the effects of the role hierarchy operations on the *ua-constraints* relation are given. The effects can be modeled by rule expressions as well similar to the other operations of the SARBAC model.

5. BENEFITS OF THE GRAPH-BASED FRAMEWORK

In previous sections we have shown that the concept of administrative scope can be specified in a graph-based security framework.

Besides a visual specification, whose advantages or disadvantages may be a question of taste, we present in the sections the benefits of the graph-based specification of the administrative scope by explaining the points beyond the approach given in [2].

5.1 Operational Semantics

No operational semantics for the SARBAC operations is described in [2]. Crampton and Loizou write that they "intend to give operational semantics for RBAC96/SARBAC by writing pseudo code functions to implement the SARBAC operations". The specification of the SRBAC operations by graph transformation rules and rule expressions as shown in the previous sections gives the SRBAC operations a formal operational semantics. Such a graph-based specification can be executed by means of graph transformation tools (e.g., AGG [12]).

Besides the sequential operational semantics presented in this article, graph transformations provide concepts of parallelism for a parallel execution of operations [8]. To ensure that any parallel application can be achieved by a sequential application of the rules independent of the application order, the rules applied in parallel must be *parallel independent*. Two applications of rules to a graph are parallel independent if the first rule does not delete anything needed by the second rule and it does not create anything that the NAC of the second rule forbids. The same must be satisfied for the second rule w.r.t. to the first rule. In the case of parallel independence, the application of rule p_1 at match m_1 and the subsequent application of rule p_2 at match m_2 (now in the result graph of the first application) results in the same graph as the application of rule p_2 at match m_2 and the subsequent application of rule p_1 at m_1 in the resulting graph.

By exploiting the parallelism results in our SARBC graph transformation model, we can, for example, apply all the *clean* rules (see Figure 11) in parallel and obtain a consistent hierarchy in one parallel step. Also the *complete* rules can be applied in parallel. On the other hand, the rule *complete* and the rule *delete edge* cannot be applied in parallel to the same roles, since the application order in this case is crucial. After deleting an edge between two roles by *delete edge*, the *complete* rules can no longer be applied to the deleted edge.

5.2 Constraints and their Verification

Crampton and Loizou write in [2] that "it is not possible in SARBAC to define arbitrary constraints on user-role and permission-role assignments". For example, the requirement that a user cannot be assigned to a role R if the user is currently assigned to some specified role R' is not expressible in SARBAC constraints. Instead of extending the concept of SARBAC constraints, the authors propose to use an access constraint language such as RCL2000 [1] to specify these kinds of constraints.

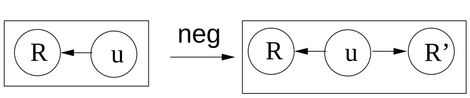

Figure 17: ***A negative constraint not expressible in SARBAC.***

We propose the use of graphical constraints for the specification of constraints. Like RCL2000, graphical constraints can express arbitrary access control constraints. Besides their expressive power, graphical constraints have the advantage of theoretical concepts suitable to verify the constraints with respect to the specified SARBAC model. A verification becomes possible since both the SARBAC operations and the SARBAC constraints are based on a common formal semantics based on graph transformations. The negative graphical constraint in Figure 17 models the requirement mentioned above. It is a negative graphical constraint which forbids the assignment of a user to role R if the user is currently assigned to role R'. Furthermore, graphical constraints can be used to formally specify informal requirements assumed in [2]. For example, [2] assumes in the RHA_3 model, that a) the relation *admin-authority* is antisymmetric, b) $a \not\leq r$ for each pair (a, r) in *admin-authority*, or c) that each role $r \in R$ is controlled by at most one administrative role. Figure 18 shows the corresponding negative graphical constraints. Constraint a) is a negative graphical constraint expressing the antisymmetry of the relation *admin-authority*, constraint b) is a negative graphical constraint for the requirement that $a \not\leq r$ for each pair (a, r) in *admin-authority*, and constraint c) is negative and specifies that each role $r \in R$ is controlled by at most one administrative role.

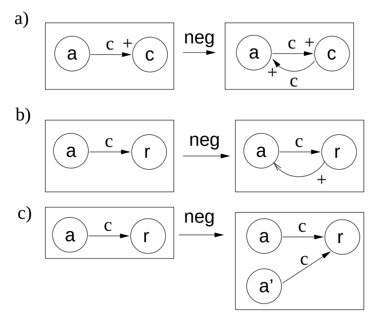

Figure 18: ***Negative graphical constraints for RHA_3 requirements.***

Formal results in the graph-based security framework enable the verification of requirements expressed in graphical constraints with respect to the graph rules used to specify the SARBAC model operations. As an example, consider the constraints in Figures 17 and 18 as well as the graph rule *assign user* in Figure 15 (which is part of the rule expression *assigUser(a,u,r)*. The graph rule can assign a user u to a role r provided that the requirements for administrative scope are satisfied. This rule may create a system state that violates the constraint in Figure 17 by assigning a user to role R that is currently assigned to role R'.

An automatic construction in [6] modifies graph rules in such a way, that they do not create any state in which they may violate a graphical constraint. Instead of a detailed introduction in this formal construction (which can be looked up in [6]), we give only the result of its application to the graph rule *assign user* and the graphical constraint in Figure 17. The modified rule (shown in Figure 19) has an additional application condition (drawn as dashed elements). This application condition specifies that the graph rule can be applied to a user u and role R only if the user is not already assigned to role R'.

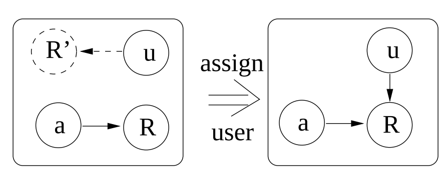

Figure 19: ***The consistent graph rule.***

5.3 Administration of SOD-constraints

Crampton and Loizou write in [2] a "priority is to complete the SARBAC model by incorporating administration of separation of duty constraints into SARBAC". We present a solution in the context of graph-based framework by graphical constraints using the example of static separation of duty, i.e., a set of particular roles are never assigned to the same user. We model the conflicting rules by a node cr which connects all the conflicting nodes. The graphical constraint to express separation of duty is given in Figure 20. It is a negative graphical constraint which forbids a user assigned to two (or more) roles which are in separation of duty conflict (we have shown only the graph X of the constraint $c : X \to X$).

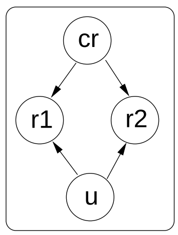

Figure 20: ***Negative graph constraint for separation of duty.***

The administration of the separation of duty constraints, i.e., the creation or deletion of conflicting role sets and the assignment and removal of single roles respectively to or from these sets is done by the following rule expressions:

- addConflictRole(a,r,cr):
  $adScope(a, r)$ **and** $adScope(a, cr)$:
  addConflictRole
- deleteConflictRole(a,r,cr):
  $adScope(a, r)$ **and** $adScope(a, cr)$:
  delConflictRole
- newConflictSet(a,cr):
  newConflictRole
- removeConflictSet(a,cr):
  $adScope(a, cr)$:
  removeConflictRole

The graph rules used in the rule expressions are shown in Figure 21.

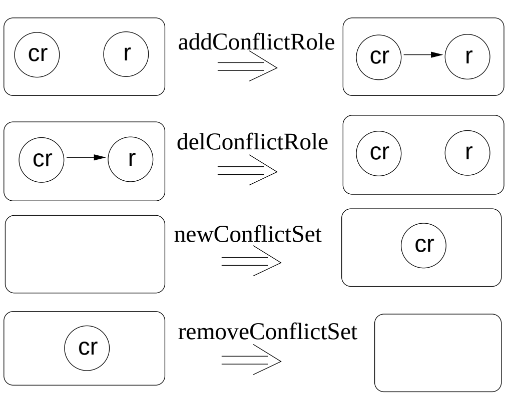

Figure 21: ***Graph rules for the administration of separation of duty conflicts.***

6. CONCLUSION

In this paper we have shown how the SARBAC model can benefit from a specification by graph transformations. Formally defined transformation rules provide an operational semantics for the different operations in SARBAC and an expressive access constraint language together with verification concepts.

Elsewhere ([6]) we have shown how to specify other role administration models ([11, 7]) with our graph transformation framework. A common specification formalism for these different administration models will facilitate a more detailed comparison of their properties.

A distinction between weak and strong revocation for users and permissions becomes necessary in SARBAC if the set of roles assigned to a user is not an anti-chain. We have already investigated weak and strong revocation in the context of a graph-based framework in [6] and plan to check the applicability of those results also in the SARBAC model.

7. REFERENCES

[1] G.-J. Ahn and R. Sandhu. Role-Based Authorization Constraints Specification. *ACM Transactions on Information and System Security*, 3(4):207–226, Nov. 2000.

[2] J. Crampton and G. Loizou. Administrative Scope: A Foundation for Role-Based Administration Models. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):201–231, 2003.

[3] 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.

[4] S. I. Gavrila, and J. 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), pp.81–90.

[5] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A Formal Model for Role-Based Access Control using Graph Transformation. In *Proc. of the 6th European Symposium on Research in Computer Security*, Volume 1895 of *Lect. Notes Comp. Sci.* (2000), pp.122–139. Springer Verlag.

[6] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A Graph Based Formalism for RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):332–365, August 2002.

[7] M. Nyanchama and S. Osborn. The Role Graph Model and Conflict of Interest. *ACM Trans. of Info. and System Security* 1, 2, 3–33.

[8] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. World Scientific, 1997.

[9] R. S. Sandhu. Role-based access control. In *Advances in Computers*, Volume 46. Academic Press, 1998.

[10] R. S. Sandhu, D. Ferraiolo and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. In Proc. of the 5th ACM Workshop on Role-Based Access Control (2000), pp.47–63.

[11] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. and Syst. Sec.*, 1(2):105–135, 1999.

[12] G. Taentzer, C. Ermel, and M. Rudolf. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, chapter The AGG Approach: Language and Tool Environment. World Scientific, 1999.