The GURA_G Administrative Model for User and Group Attribute Assignment

Maanak Gupta(✉) and Ravi Sandhu

Institute for Cyber Security and Department of Computer Science,
University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA
gmaanakg@yahoo.com, ravi.sandhu@utsa.edu

Abstract. Several attribute-based access control (ABAC) models have been recently proposed to provide finer-grained authorization and to address the shortcomings of existing models. In particular, Servos et al [33] presented a hierarchical group and attribute based access control (HGABAC) model which introduces a novel approach of attribute inheritance through user and object groups. For authorization purposes the effect of attribute inheritance from groups can be equivalently realized by direct attribute assignment to users and objects. Hence the practical benefit of HGABAC-like models is with respect to administration. In this paper we propose the first administration model for HGABAC called GURA_G. GURA_G consists of three sub-models: UAA for user attribute assignment, UGAA for user-group attribute assignment and UGA for user to user-group assignment.

Keywords: Attribute based access control, attribute inheritance, group hierarchy, group attribute administration, user-group assignment

1 Introduction

Interest in attribute-based access control (ABAC) has been developing over the past two decades, in part due to limitations of the widely deployed role-based access control (RBAC) model [32]. A number of ABAC models have been published over the years [10, 11, 12, 15, 27, 34, 36, 37], although none of these is quite regarded as the definitive characterization of ABAC.

Since ABAC access mechanism revolves around the attributes of entities, Servos et al [33] proposed the hierarchial group and attribute-based (HGABAC) model, which leverages user and object groups for allocating attribute values to users and objects. In this model, a user can be assigned to a user-group and instead of assigning attributes individually to each user in the group, a collection of attribute values is assigned to the group and inherited by all users in that group. A similar mechanism applies on the object side with object groups.

The essential benefit of HGABAC is convenient administration of attribute values for users and objects. Our contribution in this paper is to present the first administrative model for HGABAC, called GURA_G. GURA_G builds upon

the GURA model [14] for user attribute assignment (UAA) but further adds components for user-group attribute assignment (UGAA) and user to user-group assignment (UGA). For this purpose we introduce an alternate formalization of the HGABAC model which is compatible with the GURA and GURA_G models.

Remaining paper has been organized as follows. An overview of HGABAC followed with re-formalized model is discussed and specified in Section 2. In Section 3, we propose a formal role and attribute based administration model for user and user groups (GURA_G). Section 4 discusses some limitations of the proposed model. Section 5 reviews previous work related to ABAC and administration models, followed by conclusion in Section 6.

2 HGABAC Model

This section gives an informal characterization of groups in HGABAC [33], followed by a formal specification. Our formalization is in the style of ABAC_α [15], different from but equivalent to the formalization of Servos et al [33]. Our alternate formalization of HGABAC enables us to build upon the GURA administrative model [14] for ABAC_α in Section 3.

2.1 Groups in HGABAC

Similar to many ABAC models, HGABAC recognizes the entities of users, subjects and objects. A user is a human being which interacts directly with the computer, while subjects are active entities (like processes) created by the user to perform actions on objects. Objects are system resources like files, applications etc. Operations correspond to access modes (e.g. read, write) provided by the system and can be exercised by a subject on an object. The properties of entities in the system are reflected using attributes. Users and subjects hold the same set of attributes whereas objects have a separate set of attributes reflecting their characteristics. We assume all attributes are set valued. Also each attribute has a finite set of possible atomic values from which a subset can be assigned to appropriate entities.

In addition to the above familiar ABAC entities, HGABAC further introduces the notion of a group as a named collection of users or objects. Each group has attribute values assigned to it. A member of the group inherits these values from the group. Users will inherit attributes from user groups and objects from object groups. A partially ordered group hierarchy also exists in the system where senior groups inherit attribute values from junior groups.

An example user-group hierarchy is illustrated in Figure 1. Senior groups are shown higher up and the arrows indicate the direction of attribute inheritance. Since Graduate group (G) is senior to both CSD and UN, G will hold the attribute values directly assigned to it as well as values inherited from CSD and UN. The values of univId and college attributes for group G are respectively inherited from UN and CSD, values of userType and studType are directly assigned to G while the values of roomAcc are a mix of directly assigned values,

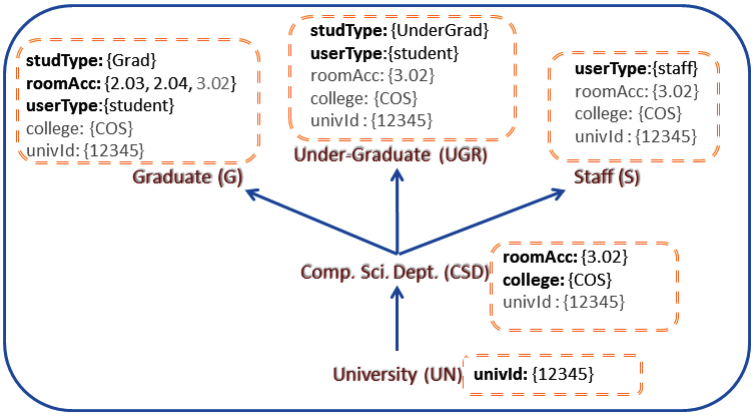

Fig. 1. Example User Groups (values in black are direct and in gray are inherited)

2.03 and 2.04, and inherited value 3.02 from CSD. Each user is assigned to a subset of user groups. Similarly there is an object-group hierarchy wherein attribute values of objects are analogously inherited.

The core advantage of introducing groups is simplified administration of user and object attributes where an entity obtains a set of attributes values by group membership in lieu of assigning one value at a time. In context of Figure 1 assigning an attribute value to CSD potentially saves hundreds or thousands of assignments to individual student and staff. Likewise changing the CSD level room from 3.02 to, say, 3.08, requires only one update as opposed to thousands.

2.2 HGABAC model: An alternate formalization

We now develop a formalization of the HGABAC model different from that of Servos et al [33]. This alternate formalization will be useful in the next section where we develop the GURA$_G$ for administration of HGABAC. Our formalization uses the conceptual model of HGABAC shown in Figure 2. The complete HGABAC formalization is given in Table 1, which we will discuss in the remainder of this subsection. An example configuration of HGABAC is given in the next subsection.

Basic sets and functions of HGABAC are shown at the top of Table 1. U, S, O and OP represent the finite set of existing users, subjects, objects and operations respectively. UG and OG represent sets of user and object groups in the system. UA is the set of user attributes for users, user groups and subjects. OA is similarly the set of object attributes for objects and object groups. All these sets are disjoint.

Attribute values can be directly assigned to users, objects, user groups and object groups (we will consider subjects in a moment). These are collectively called entities. Each attribute of an entity is set valued. The value of an attribute att for an entity is some subset of Range(att) which is a finite set of atomic values,

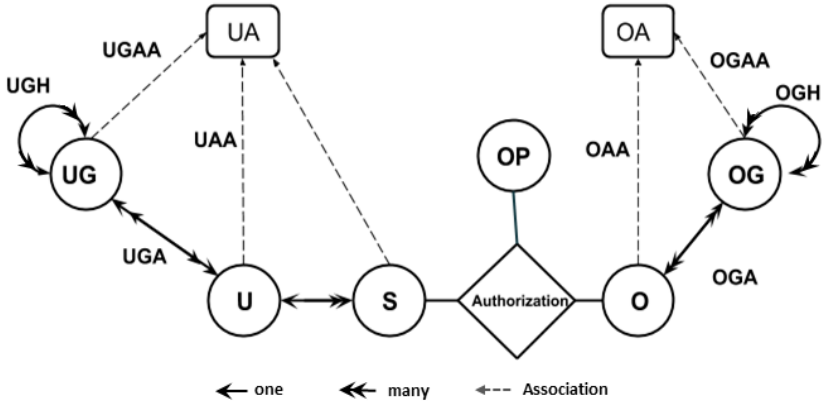

Fig. 2. A Conceptual Model of HGABAC

as indicated by the functions att_u and att_o in Table 1. These functions specify the attribute values that are directly assigned to entities. The function directUg specifies the user groups to which the user is assigned, and similarly the function directOg specifies the object groups to which an object is assigned.

User group hierarchy (UGH) is a partial order on UG, written as $\succeq_{ug}$, where $ug_1 \succeq_{ug} ug_2$ denotes ug_1 is senior to ug_2 or ug_2 is junior to ug_1. This many to many hierarchy results in attribute inheritance where the effective values of user attribute function att_u for a user-group ug (defined by effective$UG_{att_u}(ug)$) is the union of directly assigned values for att_u and the effective attribute values of all groups junior to ug. The assignment of a user to a user-group will inherit values from this group to that user. The function $effective_{att_u}$ maps a user to the set of values which is the union of the values of att_u directly assigned to the user and the effective values of attribute att_u from all user groups directly assigned to the user. Similar sets and functions are specified for objects and object groups.

A subject is created by a user, denoted by the SubUser function. The effective attribute values of a subject are under control of its creating user. These values are required to be a subset of the corresponding effective attribute values for the creator. In general these values can change with time but cannot exceed the creator's effective values. The exact manner in which a subject's effective attributes are modified by its creator is not specified in the model, and can be realized differently in various implementations.

Each operation op ∈ OP in the system has an associated boolean authorization function $Authorization_{op}$(s,o) which specifies the conditions under which subject s ∈ S can execute operation op on object o ∈ O. The condition is specified as a propositional logic formula using the policy language given in Table 1. This formula can only use the effective attribute values of the subject and object in question. The authorization functions are specified by the security policy architects when the system is created. Thereafter, a subject s_i ∈ S is allowed to execute operation op on object o o_j ∈ O if and only if $Authorization_{op}(s_i, o_j)$ evaluates to True.

Table 1. HGABAC: An Alternate Formal Model

Basic Sets and Functions

– U, S, O, OP (finite set of users, subjects, objects and operations respectively)
– UG, OG (finite set of user and object groups respectively)
– UA, OA (finite set of user and object attribute functions respectively)
– For each att in UA ∪ OA, Range(att) is a finite set of atomic values
– For each att_u in UA, $\text{att}_\text{u} : \text{U} \cup \text{UG} \rightarrow 2^{\text{Range}(\text{att}_\text{u})}$, mapping each user and user group to a set of values in $\text{Range}(\text{att}_\text{u})$
– For each att_o in OA, $\text{att}_\text{o} : \text{O} \cup \text{OG} \rightarrow 2^{\text{Range}(\text{att}_\text{o})}$, mapping each object and object group to a set of values in $\text{Range}(\text{att}_\text{o})$
– $\text{directUg} : \text{U} \rightarrow 2^{\text{UG}}$, mapping each user to a set of user groups
– $\text{directOg} : \text{O} \rightarrow 2^{\text{OG}}$, mapping each object to a set of object groups
– UGH ⊆ UG×UG, a partial order relation $\succeq_{ug}$ on UG
– OGH ⊆ OG×OG, a partial order relation $\succeq_{og}$ on OG

Effective Attributes (Derived Functions)

– For each att_u in UA,
  • $\text{effectiveUG}_{\text{att}_\text{u}} : \text{UG} \rightarrow 2^{\text{Range}(\text{att}_\text{u})}$, defined as $\text{effectiveUG}_{\text{att}_\text{u}}(\text{ug}_\text{i}) = \text{att}_\text{u}(\text{ug}_\text{i}) \cup \left(\bigcup_{\forall \text{g} \in \{\text{ug}_\text{j} | \text{ug}_\text{i} \succeq_{ug} \text{ug}_\text{j}\}} \text{effectiveUG}_{\text{att}_\text{u}}(\text{g}) \right)$
  • $\text{effective}_{\text{att}_\text{u}} : \text{U} \rightarrow 2^{\text{Range}(\text{att}_\text{u})}$, defined as $\text{effective}_{\text{att}_\text{u}}(\text{u}) = \text{att}_\text{u}(\text{u}) \cup \left(\bigcup_{\forall \text{g} \in \text{directUg(u)}} \text{effectiveUG}_{\text{att}_\text{u}}(\text{g}) \right)$
– For each att_o in OA,
  • $\text{effectiveOG}_{\text{att}_\text{o}} : \text{OG} \rightarrow 2^{\text{Range}(\text{att}_\text{o})}$, defined as $\text{effectiveOG}_{\text{att}_\text{o}}(\text{og}_\text{i}) = \text{att}_\text{o}(\text{og}_\text{i}) \cup \left(\bigcup_{\forall \text{g} \in \{\text{og}_\text{j} | \text{og}_\text{i} \succeq_{og} \text{og}_\text{j}\}} \text{effectiveOG}_{\text{att}_\text{o}}(\text{g}) \right)$
  • $\text{effective}_{\text{att}_\text{o}} : \text{O} \rightarrow 2^{\text{Range}(\text{att}_\text{o})}$, defined as $\text{effective}_{\text{att}_\text{o}}(\text{o}) = \text{att}_\text{o}(\text{o}) \cup \left(\bigcup_{\forall \text{g} \in \text{directOg(o)}} \text{effectiveOG}_{\text{att}_\text{u}}(\text{g}) \right)$

Effective Attributes of Subjects (Assigned by Creator)

– $\text{SubUser} : \text{S} \rightarrow \text{U}$, mapping each subject to its creator user
– For each att_u in UA, $\text{effective}_{\text{att}_\text{u}} : \text{S} \rightarrow 2^{\text{Range}(\text{att}_\text{u})}$, mapping of subject s to a set of values for its effective attribute att_u. It is required that : $\text{effective}_{\text{att}_\text{u}}(\text{s}) \subseteq \text{effective}_{\text{att}_\text{u}}(\text{SubUser(s)})$

Authorization Function

For each op ∈ OP, $\text{Authorization}_\text{op}$ (s:S, o:O) is a propositional logic formula, returning true or false and is defined using the following policy language:

  • $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\, \text{x} \in \text{set}.\alpha \mid \forall\, \text{x} \in \text{set}.\alpha \mid \text{set} \triangle \text{set} \mid \text{atomic} \in \text{set} \mid \text{atomic} \notin \text{set}$
  • $\triangle ::= \subset \mid \subseteq \mid \not\subseteq \mid \cap \mid \cup$
  • $\text{set} ::= \text{effective}_{\text{att}_{\text{u}_\text{i}}}(\text{s}) \mid \text{effective}_{\text{att}_{\text{o}_\text{i}}}(\text{o})$ for $\text{att}_{\text{u}_\text{i}} \in \text{UA}$, $\text{att}_{\text{o}_\text{i}} \in \text{OA}$
  • atomic ::= value

Access Decision Function

A subject $\text{s}_\text{i} \in \text{S}$ is allowed to perform an operation op ∈ OP on a given object $\text{o}_\text{j} \in \text{O}$ if the effective attributes of the subject and object satisfy the policies stated in $\text{Authorization}_\text{op}(\text{s} : \text{S}, \text{o} : \text{O})$. Formally, $\text{Authorization}_\text{op}(\text{s}_\text{i}, \text{o}_\text{j}) = \text{True}$

Table 2. Example HGABAC Configuration

Basic Sets and functions
– UA = {studId, userType, skills, studType, univId, roomAcc, college, jobTitle, studStatus}
– OA = {readerType}
– OP = {read}
– UG = {UN, CSD, G, UGR, S}, OG = { }
– UGH is given in Figure 1, OGH = { }
– Range of each att_u in UA, denoted by Range(att_u): studId = {er35, abc12, fhu53}, userType = {faculty, staff, student}, skills = {c, c++, java}, studType = {Grad, UnderGrad}, univId = {12345}, roomAcc = {1.2, 2.03, 2.04, 3.02}, college = {COS, COE, BUS}, jobTitle = {TA, Grader, Admin}, studStatus = {graduated, part-time, full-time}
– Range of each att_o in OA, Range(readerType) = {faculty, staff, student}
Authorization Function: $\text{Authorization}_{\text{read}}(s : S, o : O) \equiv \text{effective}_{\text{userType}}(s) \in \text{effective}_{\text{readerType}}(o) \wedge \text{java} \in \text{effective}_{\text{skills}}(s)$

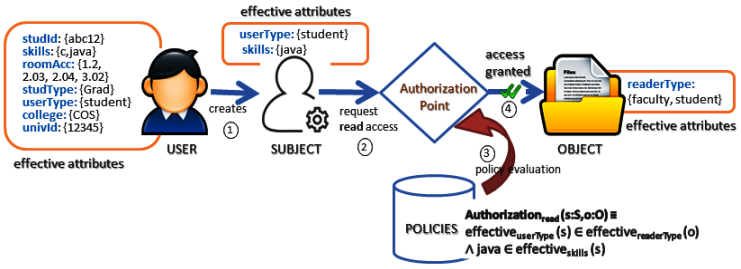

Fig. 3. Example Access Request Flow

2.3 Example HGABAC Configuration

An example HGABAC configuration is given in Table 2, utilizing the user group hierarchy of Figure 1. For simplicity, we do not include any object groups. The authorization policy for the read operation is specified. The access request flow in Figure 3 assumes the user has the set of effective attributes shown. The subject has the given subset of its creator's effective attributes. The subject is thereby allowed to read the object as the authorization policy for read is satisfied by the effective attributes of the subject and object.

3 The GURA_{G} Administrative Model

The HGABAC model offers the advantage of easy administration of attributes for users and objects. The novel approach of assigning attributes to groups and users to groups is analogous to the permission-role and user-role assignment

Table 3. GURA_G Administrative Model

Administrative Roles and Expressions

- AR : a finite set of administrative roles
- EXPR(UA) : a finite set of prerequisite expressions composed of user attribute functions as defined in section 3.1 and 3.2
- EXPR(UA ∪ UG) : a finite set of prerequisite expressions composed of user attribute functions and user groups as defined in section 3.3

Administrative Relations

- User Attribute Assignment **(UAA)** & User-Group Attribute Assignment **(UGAA):** For each att_u in UA,

$$\text{canAdd}_{\text{att}_\text{u}} \subseteq \text{AR} \times \text{EXPR(UA)} \times 2^{\text{Range}(\text{att}_\text{u})}$$

$$\text{canDelete}_{\text{att}_\text{u}} \subseteq \text{AR} \times \text{EXPR(UA)} \times 2^{\text{Range}(\text{att}_\text{u})}$$

- User to User-Group Assignment **(UGA):**

$$\text{canAssign} \subseteq \text{AR} \times \text{EXPR(UA} \cup \text{UG)} \times 2^{\text{UG}}$$

$$\text{canRemove} \subseteq \text{AR} \times \text{EXPR(UA} \cup \text{UG)} \times 2^{\text{UG}}$$

in RBAC [32]. By assigning a user to a user-group, the user inherits all the effective attribute values of that group in a single step, as compared to one by one attribute value assignment. Further, if an inherited attribute value has to be changed for multiple users, instead of changing per user, the value in a group can be changed, making administration very convenient.

The essence of HGABAC model is in simple administration as the effect of attribute inheritance can also be realized by direct attribute assignment for authorization purposes. Changing the attribute values of a group can impact large numbers of users and objects, thus reducing the administrative effort, and leading to better comprehension of attribute values. For example, in Figure 1 the fact that groups G, UGR and S inherit the roomAcc value 3.02 from CSD is visible because of the group structure.

This section presents the GURA_G administrative model for managing the user side of HGABAC. GURA_G is inspired by the GURA model [14] which in turn evolved from URA97 [30]. All these models require a set of administrative roles AR that will be assigned to security administrators. Administrative role hierarchy also exists, wherein senior administrative roles inherit permissions from junior ones. GURA_G regulates the powers of an administrative role with respect to user attribute assignment (UAA), user-group attribute assignment (UGAA) and user to user-group assignment (UGA) (see Figure 2). The *Add* and *Delete* operations enable addition or deletion of attribute values from user and user groups. Assignment or removal of a user from a user-group is accomplished by *Assign* and *Remove* operations. Table 3 depicts the various sets and administrative relations required to administer the user side of HGABAC. The prerequisite conditions are specified with slight modifications to the policy language described in Table 1. We now define the three sub-models of GURA_G.

Table 4. Example rules in UAA

canAdd$_{\text{jobTitle}}$ rule :
(DeptAdmin, Grad $\in$ effective$_{\text{studType}}$(u), {TA, Grader})
canDelete$_{\text{roomAcc}}$ rule :
(BuildAdmin, graduated $\in$ effective$_{\text{studStatus}}$(u), {1.2, 2.03, 2.04, 3.02})

3.1 The UAA Sub-Model of GURA$_G$

The UAA sub-model deals with addition or deletion of values to a set-valued attribute of a user. It is composed of two relations as shown in Table 3. The meaning of $(\text{ar}, \text{Expr(ua)}, \text{Z}) \in \text{canAdd}_{\text{att}_\text{u}}$ is that a member of an administrator role ar (or senior to ar) is authorized to add any value in the allowed range Z of attribute att_u of a user whose attributes satisfy the condition specified in Expr(ua). EXPR(UA) is the set of all prerequisite conditions represented as propositional logic expressions. The expressions return true or false and are specified using earlier defined policy language (Table 1) with following changes.

$\text{set} ::= \text{att}_{\text{u}_\text{i}}\text{(u)} \mid \text{effective}_{\text{att}_{\text{u}_\text{i}}}\text{(u)} \mid \text{constantSet} \qquad \text{for att}_{\text{u}_\text{i}} \in \text{UA}$
$\text{atomic} ::= \text{constantAtomic}$

The meaning of $(\text{ar}, \text{Expr(ua)}, \text{Z}) \in \text{canDelete}_{\text{att}_\text{u}}$ is that the member of administrator role ar (or senior) is authorized to delete any value in allowed range Z of attribute att_u of a user whose attributes satisfy the condition specified in Expr(ua). The delete operation will only impact directly assigned attribute value of the user (i.e. $val \in \text{att}_\text{u}(u)$). If the value to be deleted is inherited from a group, the operation will not have any effect. Further, if a value is both inherited and directly assigned to user, deletion will only delete the direct value, thereby, the user will still hold the value inherited from the group. It is worth mentioning that any change in prerequisite conditions after the attribute value assignment has been made, will not have any retrospective effect and the entity involved will still retain the value. This is consistent with the GURA and URA97 models.

Table 4 illustrates example UAA relation. First rule allows administrator role DeptAdmin (or senior to DeptAdmin) to add any value in {TA, Grader} to user attribute jobTitle if the user's studType attribute includes Grad. Second rule allows administrator role BuildAdmin (or senior to BuildAdmin) to remove any of the specified room values from the roomAcc attribute of a user whose status includes graduated.

3.2 The UGAA Sub-Model of GURA$_G$

This sub-model controls addition and deletion of attributes to user-groups as shown in Table 3. The relations for UAA and UGAA have slightly different policy languages for EXPR(UA), which in UGAA is defined as follows.

$\text{set} ::= \text{att}_{\text{u}_\text{i}}\text{(ug)} \mid \text{effectiveUG}_{\text{att}_{\text{u}_\text{i}}}\text{(ug)} \mid \text{constantSet} \qquad \text{for att}_{\text{u}_\text{i}} \in \text{UA}$
$\text{atomic} ::= \text{constantAtomic}$

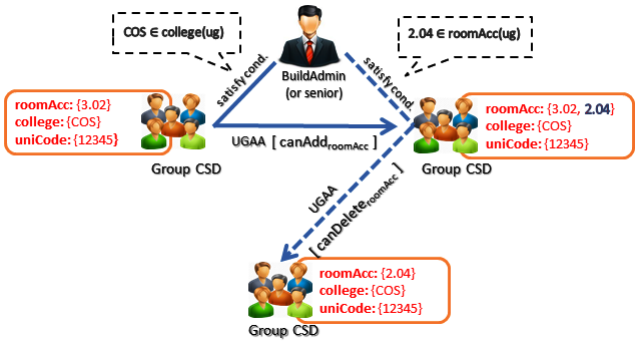

Fig. 4. Example User-Group Attribute Assignment (UGAA)

Table 5. Example rules in UGAA

canAdd$_{\text{roomAcc}}$ rule:	(BuildAdmin, COS ∈ college(ug), {2.04})
canAdd$_{\text{skills}}$ rule:	(DeptAdmin, Grad ∈ studType(ug), {c++})
canDelete$_{\text{roomAcc}}$ rule:	(BuildAdmin, 2.04 ∈ roomAcc(ug), {3.02})

The meaning of canAdd and canDelete are similar to those in UAA sub-model. In particular, the delete operation in UGAA only impacts directly assigned attribute values of a user-group (i.e $val \in \text{att}_u(ug)$) and will not delete inherited values from junior groups.

Figure 4 shows addition and deletion of attribute values to user-group CSD in context of Table 5. Addition of value 2.04 to roomAcc attribute of CSD group by administrator role BuildAdmin (or senior to BuildAdmin) is allowed by first rule in Table 5. Figure also shows deletion of 3.02 value from roomAcc attribute authorized by third rule.

3.3 The UGA Sub-Model of GURA$_G$

The UGA sub-model is composed of two authorization relations in the lower part of Table 3. These control the assignment of user to user-groups, as well as removal of a user from a user-group. The meaning of $(\text{ar}, \text{expr}, \{\text{g}_1, \text{g}_2, \text{g}_3\}) \in \text{canAssign}$ is that member of administrator role ar (or senior) can assign any user-group in $\{\text{g}_1, \text{g}_2, \text{g}_3\}$ to a user which satisfy the conditions in expr. EXPR(UA ∪ UG) now includes the current membership or non-membership of user in user-groups along with user attributes. The policy language has the following changes.

$$\text{set} ::= \text{att}_{u_i}(u) \mid \text{effective}_{\text{att}_{u_i}}(u) \mid \text{directUg}(u) \mid \text{effectiveUg}(u) \mid \text{constantSet}$$
$$\text{atomic} ::= \text{constantAtomic}$$
$$\text{where effectiveUg}(u) = \text{directUg}(u) \cup \Big(\bigcup_{\forall \text{ug}_i \in \text{directUg}(u)} \{\text{ug}_j | \text{ug}_i \succeq_{ug} \text{ug}_j\} \Big)$$

The canRemove relation in Table 3 controls the removal of a user from user-group memberships. The remove operation is said to be weak in that it will only

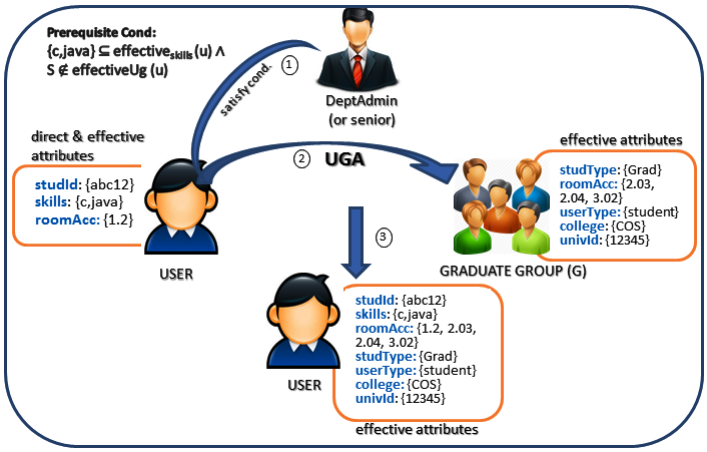

Fig. 5. Example User to User-Group Assignment (UGA)

Table 6. Example rules in canAssign UGA

Admin Role	Prereq. Cond	AllowedGroups
DeptAdmin	$\{c, java\} \subseteq effective_{skills}(u) \wedge S \notin effectiveUg(u)$	{G,CSD}
StaffAdmin	$\{G,UGR\} \cap effectiveUg(u) = \emptyset \wedge Admin \in effective_{jobTitle}(u)$	{S}
DeptAdmin	$U \in directUg(u) \wedge 3.02 \in roomAcc(u) \wedge S \notin effectiveUg(u)$	{UGR,CSD}

impact explicit memberships of user. A user is an explicit member of group ug if $ug \in$ directUg(u) whereas a user is an implicit member of ug if for some $ug_i \in$ directUg(u), $ug \in \{ug_j | \ ug_i \succeq_{ug} \ ug_j\}$ exists. It should be mentioned that removal of a user from any explicit membership ug will automatically result in removal from all implicit membership due to ug.

Figure 5 shows assignment of user to user-group G allowed by first rule in Table 6. This assignment results in updates on effective attributes of user as user now inherits all attributes from group G along with direct attributes assigned through UAA. In case of weak removal (using Figure 1), suppose a user is an explicit member of groups CSD and G and administrator role DeptAdmin removes user from CSD (authorized by second rule in Table 7), the user will still have attributes of CSD through its membership in G.

3.4 Operational Specification of $GURA_G$

Table 8 outlines administrative operations required for user-group membership and attribute assignment. In all operations: $ar \in AR, u \in U, att_u \in UA, ug \in UG$. A request (first column) succeeds only if a tuple exists in administrative relation and the entity satisfies the conditions (second column), in which case the update (third column) is performed.

Table 7. Example rules in canRemove UGA

Admin Role	Prereq. Cond	AllowedGroups
UniAdmin	graduated $\in$ effective$_{\text{studStatus}}$(u) $\wedge$ {G,UGR} $\cap$ effectiveUg(u) $\neq \emptyset$	{G,UGR}
DeptAdmin	COS $\notin$ effective$_{\text{college}}$(u)	{CSD}

Table 8. Operational Specification

Operations	Conditions	Updates
In following operations: $VAL' \in 2^{\text{Range}(att_u)}$, $val \in VAL', expr \in$ EXPR(UA)		
Add(ar, u, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in$ **canAdd**$_{att_u}$ $\wedge$ $expr(u)$ = True $\wedge$ $val \notin att_u(u)$	$att'_u(u) = att_u(u) \cup \{val\}$
Delete(ar, u, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in$ **canDelete**$_{att_u}$ $\wedge$ $expr(u)$ = True $\wedge$ $val \in att_u(u)$	$att'_u(u) = att_u(u) \setminus \{val\}$
Add(ar, ug, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in$ **canAdd**$_{att_u}$ $\wedge$ $expr(ug)$ = True $\wedge$ $val \notin att_u(ug)$	$att'_u(ug) = att_u(ug) \cup \{val\}$
Delete(ar, ug, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in$ **canDelete**$_{att_u}$ $\wedge$ $expr(ug)$ = True $\wedge$ $val \in att_u(ug)$	$att'_u(ug) = att_u(ug) \setminus \{val\}$
In following operations: $UG' \in 2^{\text{UG}}, ug \in UG', expr \in$ EXPR(UA $\cup$ UG)		
Assign(ar, u, ug)	if $\exists\langle ar, expr, UG'\rangle \in$ **canAssign** $\wedge$ $expr(u)$ = True $\wedge$ $ug \notin$ directUg(u)	directUg$'(u)$ = directUg$(u) \cup \{ug\}$
Remove(ar, u, ug)	if $\exists\langle ar, expr, UG'\rangle \in$ **canRemove** $\wedge$ $expr(u)$ = True $\wedge$ $ug \in$ directUg(u)	directUg$'(u)$ = directUg$(u) \setminus \{ug\}$

3.5 GURA$_G$ model extensions

This section proposes some enhancements to GURA$_G$.

Strong Removal: We can define a strong removal operation as per the following example using Figure 1. If a user is explicit member of CSD and G and administrator role DeptAdmin removes this user from CSD (allowed by second rule in Table 7), the user will also be removed from group G along with CSD if allowed by authorization rules. If the user cannot be deleted from G, the operation will have no effect.

Inherited Value Deletion in User: Let Alice have administrator role r_1 and Alice tries to delete inherited value val from attribute att_u of user u_1. Let there be a $canDelete_{att_u}$ rule $(r, cond, allowedVal)$ and if $r_1 \geqslant r, val \in allowedVal$ and u_1 satisfies $cond$, find all user groups ug in directUg(u_1) from where the attribute value val is inherited. There are two possibilities:(i) If there exists a $canRemove$ rule $(r, cond, allowedGroup)$ and if $r_1 \geqslant r, ug \in allowedGroup$ and u_1 satisfies the $cond$, remove u_1 from all such ug groups. (ii) If such a rule doesn't

exist or u_1 cannot be removed from some ug groups, the operation will have no effect.

Inherited Value Deletion in User Group: Let Alice have role r_1, and Alice tries to delete inherited value val from attribute att_u of user group ug_1. Let there exists a $canDelete_{att_u}$ rule $(r, cond, allowedVal)$ and if $r_1 \geqslant r, val \in allowedVal$ and ug_1 satisfies $cond$, find all user groups ug junior to ug_1 which has val directly assigned. Delete val from all such ug as if Alice did this delete. If any delete fails this operation is aborted.

4 Discussion and Limitations

The principal advantage of HGABAC model is convenient and simplified administration of attributes. GURA_G proposes first administration model for HGABAC. Reachability analysis in GURA [16] discusses whether a user can be assigned specific values with a given set of administrator roles. Since GURA_G proposes the authorization relations in line with GURA, we conjecture that similar reachability analysis is feasible for GURA_G.

At the same time, GURA_G inherits some weaknesses of URA97 and GURA as discussed in [23]. Authorization rules in UAA and UGAA may require a user or user-group to have attribute values to satisfy prerequisite conditions to get other attribute values. To attain prerequisite attribute values, entity might need to satisfy another condition which itself would require some other attributes and so on. A single GURA_G attribute assignment may require multiple attribute assignments to get final attribute values, possibly involving several administrators. These multi-step assignments may also result in some attribute values to be assigned to an entity solely for administrative purposes, but not otherwise needed.

Likewise, UGA rules may require a user having existing attribute values or membership in groups, which might also require multiple user groups or attribute pre-assignments and security administrators. If some rule has prerequisite junior groups requirement to assign a senior group membership, it will unnecessarily necessitate a user to be explicit member of junior groups, though same attribute inheritance can be achieved through senior group membership only. Thus, junior group assignments would be redundant and may lead to multiple step revocations when the user is deleted from system. An approach similar to [23] could be proposed to resolve these shortcomings where users and user-groups are assigned to organizational structure based user or group pools. Organizational pool is a group of users or user-groups with similar goals. Entities are assigned to pools and then attribute values depending on the requirements. Pools are used in prerequisite conditions instead of attributes overcoming multiple pre-assignments for user and user-groups. A similar approach can also be followed in user to user-group assignment.

The object side of HGABAC has not been discussed but it seems to be a pragmatic approach to extend URA97 for object administration as well. Though user and object have different properties, for attribute assignments we believe it

will not make any difference. For user and object group hierarchies, RRA97 [30] could be a base model to be worked upon.

5 Related Work

Several papers [4, 7, 8, 22] have been published to associate attributes to encrypted data, policies and keys. A fine grained ABAC for data outsourcing system is discussed in [12]. Work in [28] proposes key distribution center and encryption using cloud owner attributes. RBAC has been extended to use attributes for role assignment [5, 24]. [20] discussed approaches to relate roles and attributes while RB-RBAC [3] dynamically assign roles to users using attribute based rules. Role activation based on time constraints is explored in [17]. Mutable attributes in access decisions is discussed in [25]. Xin et al [15] also presented an ABAC model with DAC, MAC and RBAC configurations. Lang et al [21] proposed a model by extending XACML [1] and SAML [2] to support multi-policy ABAC. Using location attribute to secure social networks is discussed in [9]. [13] enforces separation of duty in ABAC systems. Automatic security risk adjustment based on attributes is presented in [18]. Yuan et al [37] presented an authorization architecture and policy formulation for ABAC in web services. Wang et al [36] provided framework using logic based programming to model ABAC. Preference based authorization [19] is proposed by extending XACML. Context based policy redeployment is discussed in [26]. [35] proposes an extension to assertion based policy language for federated systems. Administrative models include URA97 [31], PRA97 [29], ARBAC97 [30], GURA [14] and work by Crampton et al [6].

6 Conclusion and Future work

The paper presents first generalized URA97, called $GURA_G$, for HGABAC administration. Propositional logic conditions together with administrative roles are used to make administrative authorization decisions. $GURA_G$ has three sub-models: user attribute assignment (UAA), user-group attribute assignment (UGAA) and user to user-group assignment (UGA). The authorization relations in UAA and UGAA control addition and deletion of direct attributes from user and user-group. UGA governs assignment and removal of a user from user-groups based on the current membership (or non-membership) and attributes of user. Some extensions to $GURA_G$ have also been discussed. As $GURA_G$ proposes manual assignment of attribute values and user-groups to users, a potential foray can be to develop automated $GURA_G$ like model. An administrative model for group hierarchies and objects can also be a future prospect.

Acknowledgement

This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481.

References

1. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
2. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
3. Al-Kahtani, M.A., Sandhu, R.: A model for attribute-based user-role assignment. In: Proc. of IEEE ACSAC. pp. 353–362 (2002)
4. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proc. of IEEE Security and Privacy. pp. 321–334 (2007)
5. Chadwick, D.W., Otenko, A., Ball, E.: Role-based access control with X.509 attribute certificates. IEEE Internet Computing 7(2), 62–69 (2003)
6. Crampton, J., Loizou, G.: Administrative scope: A foundation for role-based administrative models. ACM TISSEC 6(2), 201–231 (2003)
7. Emura, K., Miyaji, A., Nomura, A., Omote, K., Soshi, M.: A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In: ISPEC, pp. 13–23. Springer (2009)
8. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proc. of ACM CCS. pp. 89–98 (2006)
9. Hsu, A.C., Ray, I.: Specification and enforcement of location-aware attribute-based access control for online social networks. In: Proc. of ACM ABAC'16. pp. 25–34 (2016)
10. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (ABAC) definition and considerations. NIST Special Publication 800-162 (2014)
11. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. IEEE Computer (2), 85–88 (2015)
12. Hur, J., Noh, D.K.: Attribute-based access control with efficient revocation in data outsourcing systems. IEEE TPDS 22(7), 1214–1221 (2011)
13. Jha, S., Sural, S., Atluri, V., Vaidya, J.: Enforcing separation of duty in attribute based access control systems. In: ICISS, pp. 61–78. Springer (2015)
14. Jin, X., Krishnan, R., Sandhu, R.: A role-based administration model for attributes. In: Proc. of ACM SRAS. pp. 7–12 (2012)
15. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering dac, mac and rbac. In: DBSec. pp. 41–55. Springer (2012)
16. Jin, X., Krishnan, R., Sandhu, R.: Reachability analysis for role-based administration of attributes. In: Proc. of ACM DIM. pp. 73–84. ACM (2013)
17. Joshi, J.B., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE TKDE 17(1), 4–23 (2005)
18. Kandala, S., Sandhu, R., Bhamidipati, V.: An attribute based framework for risk-adaptive access control models. In: Proc. of IEEE ARES. pp. 236–241 (Aug 2011)
19. Kounga, G., Mont, M.C., Bramhall, P.: Extending XACML access control architecture for allowing preference-based authorisation. In: TrustBus, pp. 153–164. Springer (2010)
20. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. IEEE Computer 43(6), 79–81 (2010)
21. Lang, B., Foster, I., Siebenlist, F., Ananthakrishnan, R., Freeman, T.: A flexible attribute based access control method for grid computing. Journal of Grid Computing 7(2), 169–180 (2009)
22. Liang, K., Fang, L., Susilo, W., Wong, D.: A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security. In: Proc. of IEEE INCoS. pp. 552–559 (2013)

23. Oh, S., Sandhu, R., Zhang, X.: An effective role administration model using organization structure. ACM TISSEC 9(2), 113–137 (2006)
24. Oppliger, R., Pernul, G., Strauss, C.: Using attribute certificates to implement role-based authorization and access controls. Sicherheit in Informationssystemen. pp. 169–184 (2000)
25. Park, J., Sandhu, R.: The UCON ABC usage control model. ACM TISSEC 7(1), 128–174 (2004)
26. Preda, S., Cuppens, F., Cuppens-Boulahia, N., Garcia-Alfaro, J., Toutain, L.: Dynamic deployment of context-aware access control policies for constrained security devices. Journal of Systems and Software 84(7), 1144–1159 (2011)
27. Priebe, T., Dobmeier, W., Kamprath, N.: Supporting attribute-based access control with ontologies. In: Proc. of IEEE ARES. pp. 8–pp (2006)
28. Ruj, S., Nayak, A., Stojmenovic, I.: DACC: Distributed access control in clouds. In: Proc. of IEEE TrustCom. pp. 91–98 (2011)
29. Sandhu, R., Bhamidipati, V.: An Oracle implementation of the PRA97 model for permission-role assignment. In: Proc. of ACM RBAC workshop. pp. 13–21 (1998)
30. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM TISSEC 2(1), 105–135 (1999)
31. Sandhu, R.S., Bhamidipati, V.: The URA97 model for role-based user-role assignment. In: DBSec. pp. 262–275. Chapman & Hall, Ltd. (1998)
32. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer (2), 38–47 (1996)
33. Servos, D., Osborn, S.L.: HGABAC: Towards a formal model of hierarchical attribute-based access control. In: FPS, pp. 187–204. Springer (2014)
34. Shen, H.b., Hong, F.: An attribute-based access control model for web services. In: Proc. of IEEE PDCAT. pp. 74–79 (2006)
35. Squicciarini, A.C., Hintoglu, A.A., Bertino, E., Saygin, Y.: A privacy preserving assertion based policy language for federation systems. In: Proc. of ACM SACMAT. pp. 51–60 (2007)
36. Wang, L., Wijesekera, D., Jajodia, S.: A logic-based framework for attribute based access control. In: Proc. of ACM FMSE. pp. 45–55 (2004)
37. Yuan, E., Tong, J.: Attributed based access control (ABAC) for web services. In: Proc. of IEEE ICWS (2005)