Simulation of the Augmented Typed Access Matrix Model (ATAM) using Roles

Qamar Munawer and Ravi S Sandhu
George Mason University, Fairfax, VA 22030

ABSTRACT:

Role-based Access Control (RBAC) is a promising alternative to traditional discretionary (DAC) and mandatory access (MAC) controls. In RBAC permissions are associated with roles, and users are made members of the roles thereby acquiring the roles' permissions. RBAC is policy neutral and flexible enough to accommodate diverse security policies. Access matrix models define another mechanism for enforcing the security policy. The Augmented Typed Access Matrix model (ATAM), an extension of Typed Access Matrix (TAM) model, defined by Sandhu is well known from this class of models. ATAM is defined by introducing strong typing (i.e., each subject or object created is to be of particular type which thereafter does not change). The ATAM is recognized as the current state of the art with respect to formal models for generalized access control policies.

In this paper we formally show that ATAM can be simulated by appropriate configuration of RBAC components. Our results attest to the flexibility of RBAC and its ability to accommodate a wide range of decentralized administrative models.

1 Introduction

The controlled sharing of information and other resources among multiple users has lead to the need for development of access control models such as [1, 3, 6, 9, 10]. The Access Control or protection models provide a formalism and framework for specifying, analyzing and implementing security policies in multi-user systems. Augmented Typed Access Matrix (ATAM) model [1] is one from this family of access control models that is recognized as the current state of the art with respect to models for generalized access control policies [11]. The model allows the individual users to specify access to other users to the objects they control. At the same time this discretionary power of individual users can be constrained to meet the overall objectives and policies of the organization. The model is defined in terms of well-known abstractions of subjects, objects, access rights and strong types.

Role-Based Access Control [7, 8] is considered as an alternative to traditional discretionary and mandatory access controls. In RBAC the permissions are assigned to the roles and users are made members of the roles thereby acquiring the roles' permissions. This greatly simplifies management of permissions. Roles are created as per job functions in the organization and users are made members of roles based on their responsibilities and qualifications. New permissions can be granted to the roles and the permissions can be revoked as needed. The reason for the RBAC popularity is that by itself it is policy neutral. The policy enforced in a system is the net result of the precise configuration and interaction of various RBAC components.

It has previously been shown in literature that Discretionary Access Control (DAC) can be easily accommodated in RBAC by configuring a few components [5]. This leads to the question that whether or not ATAM can be simulated in RBAC. In this paper we show that flexibility of RBAC makes it possible to simulate ATAM by doing configuration of some of its components. There is no formal model for RBAC, however we will use the RBAC96 framework proposed by Sandhu et. al. [8] for the simulation of ATAM. Our results are of theoretical importance to show that RBAC is highly decentralized model that is flexible enough to accommodate a wide range of administrative models.

The paper is organized as follows. Section 2 describes the brief review of ATAM. The RBAC96 is described in section 3. Section 4 provides the simulation of ATAM in terms of RBAC96 components with an example and paper is concluded in section 5.

2 The Augmented Typed Access Matrix Model (ATAM)

In this section we briefly review the Augmented Typed Access Matrix model (ATAM)[1].

The access matrix model was first formalized by Harrison, Ruzzo and Ullman and called HRU [3]. The model had broad expressive power, but weak safety property (i.e., the determination of whether or not a given subject can ever acquire access to a given object). Sandhu [6] proposed TAM to incorporate the good safety results and at the same time have the general expressive power of HRU. The principal innovation of TAM is to introduce strong typing of subjects and objects into the access matrix model of HRU and it allows the check for the absence of right. Each subject or object is created of specific type, which thereafter cannot be changed. The types and rights are specified as part of the system definition and are not predefined in the model. This adds up some flexibility in term of the implementation of the security policy of the organization. The extension of TAM proposed by Sandhu is ATAM [1], which allows checking for the absence of rights in the commands. TAM and ATAM are equivalent in expressive power, however from practical point of view it is beneficial to allow testing for absence of rights.

ATAM represents the distribution of rights by the access matrix. The matrix has a row and column for each subject and a row for each object. Subjects are also considered as objects. The rights a subject X possess for object Y are entered in cell [X, Y] of the access matrix.
The security officer specifies the following sets as the part of definition of the system:

1. Finite set of access rights denoted by R_{right}.
2. The finite set of object types T. There is a set of subject types T_S, $T_S \subseteq T$

For example T = {user, so, file} specifies there are three types, user, security officer, and file, with T_S = {user, so}. The set of rights is R_{right} = {r, w, o} where r stands for read, w for write and o for owner.

The rights in the access matrix serve two purposes. First is the authorization of the subject to perform some operation on the object or to perform some operation that changes the access matrix. For example right 'o' (owner) in [X, Y] authorizes X to change the matrix so that subject Z can read Y. The focus of ATAM is on the second purpose of rights i.e., the authorization by which access matrix gets changed. The changes are made by means of commands of following formats:

Command $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, \ldots, X_k : t_k)$
If $r_1 \in [Xs_1 , Xo_1] \wedge r_2 \in [Xs_2 , Xo_2] \wedge \ldots \wedge r_k \in [Xs_m , Xo_m] \wedge$
$R_{k+1} \notin [Xs_1 , Xo_1] \wedge r_{k+2} \notin [Xs_2 , Xo_2] \wedge \ldots \wedge r_m \notin [Xs_m , Xo_m]$
then
$op_1 ; op_2 ; \ldots..; op_n$
end
Or

Command $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, \ldots, X_k : t_k)$
$op_1 ; op_2 ; \ldots..;op_n$
end

Here α is name of the command X_1, X_2, X_3, …, X_k are formal parameters whose types are t_1, t_2, t_3, …, t_k whereas r_1, r_2, r_3, …, r_n are rights and $s_{1,}$ $s_{2,}$ ….$s_{m,}$ and o_1, o_2 ,…, o_m are integers between 1 and k. The predicate following the *if* part is called the condition and sequence of operations op_1 ; op_2 ; …..;op_n is called the *body* of the command. The ATAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The usual interpretation is that the ATAM command is initiated by the first subject in the parameters list. The condition part is evaluated with respect to its actual parameters. The body of the conditional command is executed only if the condition evaluates to be true. Each op_1 is one of the primitive operations from

Enter r into [X, Y]
Create subject X of type t_s
Create object O of type t_o
Delete r from [X, Y]
Destroy subject X
Destroy object O

Enter operation enters a right $r \in R_{right}$ into an existing cell of access matrix. If right is already present then the contents are not changed. Enter only enters the right and does not remove it from access matrix. The delete operation removes the right from the cell. The cell is treated as set, thus there will not be any effect if the cell does not have the right.

The ‘Create subject' operation introduces an empty row and column in the access matrix. It is required that the subject being created must have a unique identity. ‘Destroy subject’ removes the row and column corresponding to the subject. ‘Create object’ operation adds an empty row in the access matrix and ‘Destroy object’ remove the corresponding row.

The *protection state* of the ATAM system, which is defined as a set {SUB, OBJ, t, AM} where

- SUB is a set of subjects.
- OBJ is a set of objects.
  (SUB is a subset of OBJ)
- t is a type function that maps a subject with subject type and object with object type.
- AM is an access matrix, with a row for every subject in SUB and column for every object in OBJ.

The *protection state* is changed by means of ATAM commands. The security officer defines a finite set of ATAM commands when the system is specified. The format of these commands is explained above.

3 Role-Based Access Control Model (RBAC96)

The family of RBAC96 models [7] was recently defined by Sandhu is summarized in figure 1. The model is based on three sets of entities called users (U), roles (R) and

permissions (P). User is a human being, a role is a job function within the organization with some semantics regarding the authority and responsibility conferred on the member of the role and permission is an approval of a particular mode of access to one or more objects in the system. The user assignment (UA) is a relationship that defines the membership of user to roles. The relationship is many to many because a user can be member of many roles and a role can have many users. The relationship permission assignment (PA) defines the permission assignments to roles. The relationship is many to many because permission can be assigned to many roles and at the same time a role can have many permissions. The model also have partially ordered role hierarchies (RH) written as ≥, where x ≥ y means that role x inherits the permissions assigned to role y. The model allows multiple inheritance and that inheritance is transitive.

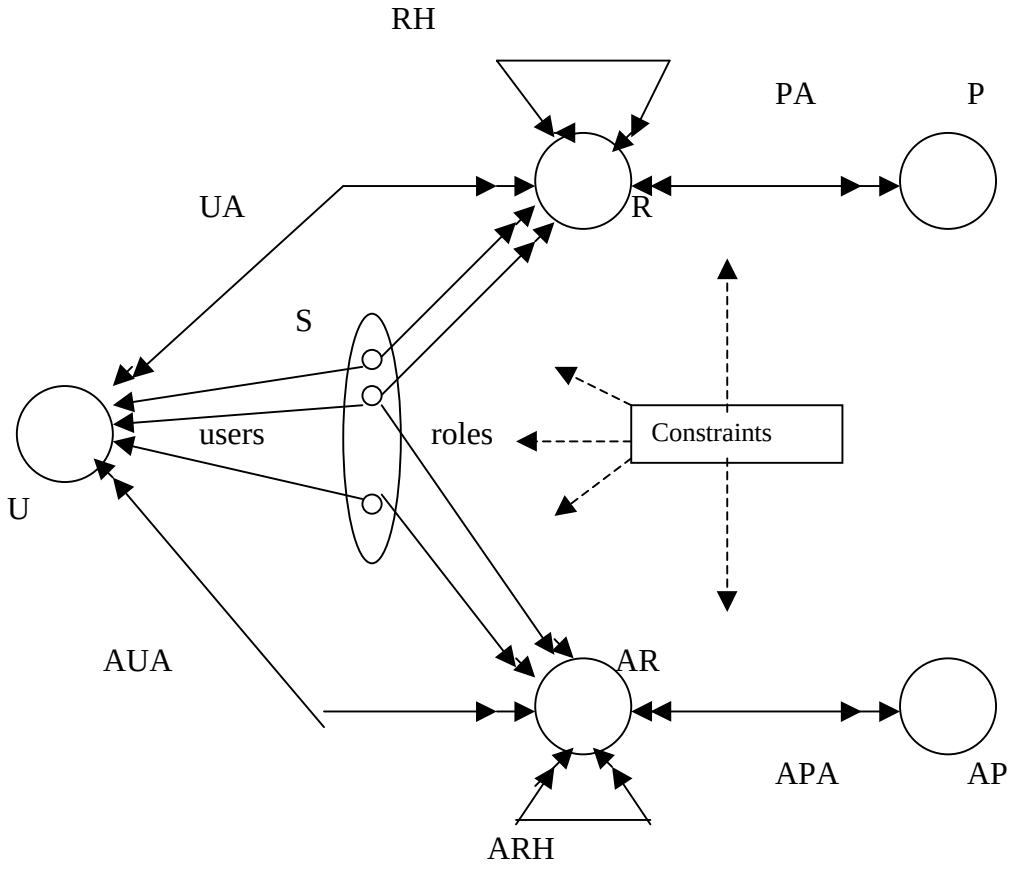

Figure 1

There is a set of sessions S. Each session relates one user to possibly many roles. A user can activate a subset of roles he/she is assigned in a session. The permissions available to user are the union of permissions assigned to the roles activated in the session. The session is related to single user and this association remains constant for the life of the session. There is a collection of constraints that can be applied to any components. An example is the mutually exclusive roles, where a user if member of one role cannot be the

member of other role or if a user activates a role in a session he cannot activate other role in that session. Following is the formal definition of the RBAC96 model.

The summary of RBAC96 framework is:

- U, a set of users
  R and AR, disjoint set of (regular) roles and administrative roles
  P and AP, disjoint set of (regular) permissions and administrative permissions
  S, a set of sessions

- $UA \subseteq U \times R$, user to role assignment relation
  $AUA \subseteq U \times AR$, User to administrative role assignment relation

- $PA \subseteq P \times R$, permission to role assignment relation
  $APA \subseteq P \times AR$, permission to administrative role assignment relation

- $RH \subseteq R \times R$, partially ordered role hierarchy
  $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
  (both hierarchies are written as $\geq$ in infix notation)

- *user* : $S \rightarrow U$, maps each session to a single user (which does not change)
  roles : $S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles
  $roles(s_I) \subseteq \{ r \mid (\exists r' \geq r)[user(s_I), r' \in UA \cup AUA]\}$ (which can change with time)
  session s_I has the permissions $\cup_{r \in roles(si)} \{p \mid \exists r' \leq r)[(p, r'') \in PA \cup APA]\}$

- There is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.

4 Simulation of ATAM in RBAC96

In this section we will describe the ATAM simulation using the framework of RBAC96.

4.1 Overview

As explained in section 2, ATAM requires that the security officer should specify the finite set of types (T), rights (R_{right}) and a set of commands as part of the system definition. The subjects and objects are created of specific types, which thereafter cannot be changed. The strong types are the principle innovation of ATAM. ATAM represents the distribution of rights by the access matrix. The access matrix has a row and column for each subject and a row for each object. Subjects are also considered as objects. The objects that have only a column in the access matrix are called pure objects. In ATAM terminology ATAM objects and denoted by symbol OBJ. If SUB is set of subjects then

Set of pure Objects = OBJ – SUB

The contents of access matrix are changed by means of commands. The usual interpretation of ATAM command is that it is initiated by the first parameter in the parameter list of the command. For each ATAM command we create an administrative permission which performs the required checks and changes in the RBAC96 components. The administrative role 'ADMN_ROLE' is created and all administrative permissions are assigned to this role. All users are made members of this administrative role. In RBAC96 we will have:

AR = {ADMN_ROLE}

In our simulation we will create a role for each type. When an object O is created of certain type t then we create self_O role senior to role that correspond to the type t. At the same time we create one role for each right along with the permissions, one for each right on the object. The permissions are assigned to the corresponding roles that cannot be changed thereafter. This means that the permission role assignments (PA) are initiated ONLY at the time of role creation. If the object is a subject (S) then a user, user(S) is also created and made member of the roles self_S and ADMN_ROLE. In this way if S is the first parameter of the command then user(S) will be able to make the checks and changes in the components of RBAC.

Deleting an object X removes the roles, permissions and user_X, if any, corresponding to the object X from the RBAC system. The removal of the user_X will require its revocation from all roles it is explicit member of.

There is no **protection states** in RBAC96. A change in the set SUB or OBJ or the change in the contents of any cell of AM changes the state. The change in SUB and OBJ changes the set of roles in the system whereas the change in the contents of the cells of AM changes the UA and PA components of the equivalent RBAC96. This suggests that we may consider the set of components {U, R, PA, UA, RH} of RBAC96 to represent the protection state of the ATAM system.

4.2 Formal description of the simulation:

In this section we formally describe the simulation of ATAM system using roles. In the RBAC and ATAM mapping we use the RBAC terminology on the left hand side and ATAM on the right hand side of the equations and expressions. Therefore R appearing on left side of expression is 'ROLE' where as on right side it is 'RIGHT'. The mapping of ATAM definition to RBAC96 constructions is as follows:

4.2.1 Administrative role ADMN_ROLE.

We create an administrative role 'ADMN_ROLE'. It is represented by RBAC96 component as:

AR = {ADMN_ROLE}

4.2.2 ATAM types are RBAC96 roles.

For each ATAM type we will have an RBAC96 role. For example: if we have the set of ATAM types as $T = \{t_1, t_2, t_3, \ldots, t_n\}$ then in RBAC96 simulation we will create n roles namely, $t_1, t_2, t_3, \ldots, t_n$. In RBAC96 we will have

$$R \supseteq \{ t \mid t \in T\}$$

4.2.3 Mapping of ATAM rights and ATAM Objects:

For each ATAM object we create

- Role self_Object senior to 'type role' this object is created of.
- Roles corresponding to the object and rights (one for each right) and
- Corresponding Permissions (one for each right).
- However of the object is a subject then we also create a user(S) and session_X.
- user_X is assigned the membership of the self_X and ADMN_ROLE.

For example, the creation of object X of type t in ATAM is equivalent to create a role self_X senior to role t and the corresponding roles for object X are $r_1\_X, r_2\_X, \ldots r_m\_X$ with corresponding permissions $Can\_r_1\_X, Can\_r_2\_X, \ldots, Can\_r_m\_X$ assuming that $R_{right} = \{ r_1, r_2, r_3, \ldots r_m\}$. Furthermore if the object is a subject then a user_X and a session_X are created. The user_X is made member of the role self_X and ADMN_ROLE. This leads to the following mapping

$$U = \{user\_X \mid X \in SUB\}$$
$$R = \{t \mid t \in T\} \cup \{r\_X \mid r \in R_{right} \wedge X \in OBJ\} \cup \{self\_X \mid X \in OBJ\}$$
$$P = \{Can\_r\_X \mid r \in R_{right} \wedge X \in OBJ\}$$
$$Sessions = \{session\_X \mid X \in SUB\}$$
$$Constraint: \ user(session\_X) = user\_X$$
$$UA = \{(user\_X, self\_X) \mid X \in SUB\}$$
$$AUA = \{(user\_X, ADMN\_ROLE) \mid X \in SUB\}$$

4.2.4 The ATAM commands:

Each ATAM command is equivalent to some changes in the components of RBAC96. This is achieved by creating an administrative permission for each command. These permissions will perform the equivalent checks and changes in the components of RBAC96. For example, for ATAM command 'create object' we will create an administrative permission 'create object' that will perform equivalent checks and changes in RBAC96 components. These administrative permissions are assigned to the role ADMN_ROLE. That is

$$AP = \{\alpha \mid \alpha \text{ is an ATAM command}\}$$
$$APA = \{(\alpha, ADMN\_ROLE) \mid \alpha \text{ is an ATAM command}\}$$

The ATAM command consists of three parts namely parameter list, condition and body. The translation of each of it is as follows:

4.2.4.1 Parameters of the command:

Formal parameters of ATAM commands are the checks of existing objects for the memberships or relationships of roles that map with the ATAM types.

The Command $\alpha(X_1 : t_1, X_2 : t_2, X_3 : t_3, \ldots, X_k : t_k)$ is simulated as follows:

1. Check if role self_ X_2 is senior to role t_2 for existing objects and
2. If X_2 is the first parameter in the parameter list then checking the membership of user_X_2 in role self_X_2 and ADMN_ROLE.

Example: Command $\alpha(X_1 : t_1, X_2 : t_2)$

If …..
Create object X_2 of type t_2
end

The above conditions for X_1: t_1 will be checked whereas the checks for X_2 will not be performed because X_2 is not an existing object. On the other hand the role self_X_2 senior to t_2 along with the roles and permissions (depending upon the set of rights) as explained in section 4.2.3.3 will be created as a result of 'Create object X_2 of type t_2' operation in the body of the command.

4.2.4.2 Condition part of the command:

- The condition part of the command is the testing for the membership and non-membership in a role. For example

(a). If $r \in [Xs_1 , Xo_1]$ will be translated as user_$Xs_1 \in rXo_1$ and

(b). If $r \notin [Xs_1 , Xo_1]$ will be translated as user_$Xs_1 \notin rXo_1$

4.2.4.3 The body of the command:

The body of the command consists of the *ATAM operations* which are translated as: (In this section the RBAC terminology is used on both right and left side of the equations and expressions.)

- **Create object/subject X of type t** is translated as creation of self_X role senior to role t, all right related roles (i.e., r_1X, r_2X,…, r_nX) and permissions (i.e., Can_r_1_X, Can_r_2_X, ….., Can_r_m_X) with respect to object X. In RBAC96 this is captured by the following changes in the components:

  $R = R \cup \{self\_X, r_1X, r_2X,\ldots,r_nX\}$ (R on right side is also a set of ROLES)

  $P = P \cup \{Can\_r_1\_X, Can\_r_2\_X, \ldots.., Can\_r_m\_X\}$

  $PA = PA \cup \{(r_1X, Can\_r_1\_X), (r_2X ,Can\_r_2\_X), \ldots.., (r_mX, Can\_r_m\_X)\}$

  $RH = RH \cup \{(self\_X > t)\}$

  If the object is a subject the we also create user_X and assign it the membership of roles self_X and ADMN_ROLE. This will be equivalent to the following changes in RBAC96 components:

  $U = U \cup \{user\_X\}$

  $UA = UA \cup \{(user\_X, self\_X), (user\_X, ADMN\_ROLE)\}$

- E**nter r into [X, Y]** is translated as making user_X member of role rY. The effect in RBAC96 is as:

UA = UA $\cup$ {(user_X, rY)}

- **Delete r from [X, Y]** is revoking the membership of user_X from role rY. The effect is:
  UA = UA – {(user_X, rY)}

- **Destroy object/subject X**: In ATAM the effect of this operation is removal of row or/and column associated with the object/subject X from access matrix. On the RBAC side this requires the deletion of all roles and permissions associated with the object X. This is depicted by the following component changes:
  R = R – {r_1X, r_2X,…,r_nX}
  P = P – {Can_r_1_X, Can_r_2_X, ….., Can_r_m_X }
  PA = PA - {(r_1X, Can_r_1_X), (r_2X ,Can_r_2_X), ….., (r_mX, Can_r_m_X)}
  RH = RH – {(self_X > t)}
  However if the object is a subject then we require that first the explicit membership of user_X be revoked from all other roles.
  U = U – {user_X}
  $\forall r \in R \mid UA = UA - (user\_X, r)$

4.3 Summary:

In this section we give the summary of the above translation of ATAM system into RBAC96. On the left side of the expression we use RBAC terminology whereas on right is ATAM terminology. Therefore R on right represents the set of ATAM rights and on left it is set of ROLES.

4.3.1 Definition of RBAC96 components

U = {user_X | X $\in$ SUB}
R = {t | t $\in$ T} $\cup$ {r_X| r $\in$ R_{right} $\wedge$ X $\in$ OBJ} $\cup$ {self_X | X $\in$ OBJ}
AR = [ADMN_ROLE]
P = {Can_r_X| r $\in$ R $\wedge$ X $\in$ OBJ}
PA = {(Can_r_X, r_X) | r $\in$ R $\wedge$ X $\in$ OBJ}
AP = {α | α is an ATAM command}
APA = {(α, ADMN_ROLE) | α is an ATAM command}
(The interpretation AP and APA is as explained in section 4.2.4)
Sessions = {session_X | X $\in$ SUB}
UA = {(user_X, self_X) | X $\in$ SUB}
AUA = {(user_X, ADMN_ROLE) | X $\in$ SUB}
RH = {self_X > t | X $\in$ OBJ}

Constraint on sessions:

- user(session_X) = user_X

- A user can have only one session that persists as long as the user remains in the system.

Constraints on PA and APA:

- The permissions are assigned to the roles only at the time of creation and cannot be changed thereafter.
- There is one administrative permission per ATAM command and they are assigned to administrative role ADNM_ROLE.

4.4 EXAMPLE:

In this subsection we demonstrate the ATAM simulation in RBAC96 using the example of Liberal DAC policy with one level grant option [5].

4.4.1 Multi-level DAC policy and its simulation

In multi-level DAC policy the owner of the object can grant authority to users who in turn can use this authority to grant access to the object. So Alice being the owner of the object O can grant access to Bob. Now Bob can grant access to Charles. But Bob cannot grant Charles the power to further grant access to Dorthy. The ATAM solution to this policy is as follows:

Types: {s, o}
Rights: {own, read, ReadwithGrant}
Commands are as follows:
(a) Command Create_Object(S:s; O:o)
Create object O of type o
Enter own in [S,O]
Enter read in [S,O]
end
(b) Command Grant_Read_ObjectWithGrant(S:s;S':s; O:o)
If own ∈ [S, O] then
Enter ReadwithGrant in [S',O]
end
(c) Command Grant_Read_Object(S:s;S':s; O:o)
If own ∈ [S, O] or ReadwithGrant ∈ [S, O] then
Enter read in [S',O]
end

For simplicity we do not consider the revoke commands in this example.

4.4.2 RBAC96 simulation of ATAM solution

In this section we will demonstrate the use of mappings described in section 4 to translate the ATAM solution into RBAC96 constructions.

1. Create an administrative role ADMN_ROLE.

2. The types are the RBAC96 roles: Thus we create two roles S_ROLE for s and O_ROLE for o type.
3. The ATAM commands can be implemented by checking the memberships and relationships. Depending upon the results of these checks, roles and permissions are created/deleted and user role and permission role assignments are done. Here we show the effects of ATAM command with respect to RBAC96:

(a) Command Create_Object(S:s; O:o)
Create object O of type o
Enter own in [S,O]
Enter read in [S,O]
end

Is equivalent an administrative permission 'Create_Object' that can perform following checks and changes:

if self_S > S_ROLE and user_S ∈ self_S
then
R = R ∪ {self_O, OWN_O, ReadWithGrant_O, READ_O} and
P = P ∪ {CanOwn_O, CanReadWithGrant_O, CanRead_O }
PA = PA ∪ {(OWN_O, CanOwn_O), (ReadWithGrant_O, CanReadWithGrant_O), (READ_O, CanRead_O) }
RH = RH ∪ {self_O > O_ROLE}
UA = UA {(user_S, OWN_O), (user_S, READ_O)}

That is, if self_S > S_ROLE then create RBAC96 roles self_O, OWN_O, READ_O, ReadWithGrant_O with the permissions CanOwn_O, CanReadWithGrant_O, CanRead_O are created. The role self_O is made senior to role O_ROLE and we also do the permission role assignments.

(b) Command Grant_Read_ObjectWithGrant(S:s; S':s; O:o)
If own ∈ [S, O] then
Enter ReadwithGrant in [S',O]
end

Is equivalent an administrative permission 'Grant_Read_ObjectWithGrant' that can perform following:

if user_S ∈ self_S and self_S > S_ROLE and
self_S' > S_ROLE and self_O > O_ROLE
then
UA = UA ∪ (user(S'), ReadWithGrant_O)

That is, if self_S, self_S' roles are senior to S_ROLE and self_O is senior to O_ROLE then user(S') are assigned the membership of ReadWithGrant_O role.

(c) Command Grant_Read_Object(S2:s;S3:s; O:o)
If own ∈ [S2, O] or ReadwithGrant ∈ [S2, O] then
Enter read in [S3,O]
end

The RBAC96 administrative permission 'Grant_Read_Object' can do following equivalent changes:

If user_S2 > S_ROLE and user_S ∈ self_S and
User_S3 > S_ROLE and
self_O > O_ROLE and
(user_S2 ∈ OWN_O or (user_S2 ∈ ReadWithGrant_O
then
UA = UA ∪ (user_S3, READ_O)

The three administrative permissions are assigned to the role ADMN_ROLE.

5. CONCLUSION

In this paper we have shown that how we can simulate the simple but highly decentralized administrative access control model ATAM [6] by configuration of various components of RBAC96 [7]. This proves that

$$ATAM \subseteq RBAC96$$

This fact is theoretically important, especially in conjunction with earlier results that MAC [4] and DAC [5] can be simulated using roles. The results of this paper show that RBAC96 can not only subsumes both traditional forms of access control but can also accommodate other administrative models.

The RBAC96 provides an open ended general framework for access control. Not only the Policies like MAC and DAC can be accommodated in RBAC96 framework but models such as HRU [3], TAM [6] and ATAM [1] can be reduced to RBAC96. Can ATAM accommodate RBAC framework? We leave this topic open for further research. It will be interesting to see if RBAC96 framework can be simulated in ATAM.

References:

1. Amman, P.E. and Sandhu R. "*Implementing Transaction Control Expressions by Checking for Absence of Access Rights*", Proc. Eighth Annual Computer Security Application Conference, San Antario, Texas, Dec. 1992.

2. Amman, P.E. and Sandhu R. "*The Extended Schematic Protection Model*." Journal of Computer Security.", Annual Computer Security Application Conference, 1992.

3. Harrison, M.H., Ruzzo W.L., and Ullman J.D. "*Protection in Operating Systems*" Communication of ACM 19(8), 1976.

4. Matunda Nyanchama and Sylvia Osborn "*Modeling mandatory access control in role-based security systems*", Database security VIII: status and Prospects. Chapman-Hall 1996.

5. Sandhu, R and Munawer, Q. "*How to do Discretionary Access Control using Roles*", Proc. of 3rd ACM Workshop on Role Based Access Control, Fairfax, Virginia, Oct. 22-23, 1998.

6. Sandhu , R. "*The Typed Access Matrix Model*" IEEE Symposium on Research in Security and Privacy. Oakland, CA. 1992.

7. Sandhu R. "*Rationale for the RBAC96 family of access control models*", Proc. Of 1st ACM Workshop on Role-based Access Control. ACM 1997.

8. Sandhu R., Coyne E.J., Feinstein H.L. and Youman C.E. "*Role-based Access Control Models*", IEEE Computer, 29(2) Feb. 1996.

9. Sandhu R. and Sanarati P. "*Access control Principles and practice*", IEEE Communications, 32(9), 1994.

10. Sandhu R. "*The Schematic protection model: Its definition and analysis for acyclic attenuating schemes.*", Journal of the ACM, 32(2), April 1988.

11. John McLean, "*Security Models*." In Encyclopedia of Software Engineering, editor John Marciniak. Wiley & Sons, Inc., 1994.