Towards Administration of a Hybrid Role Hierarchy

Suroop Mohan Chandran **James B. D. Joshi**

Dept. of Information Science and Telecommunications, University of Pittsburgh, Pittsburgh, PA
{sum22@pitt.edu; jjoshi@mail.sis.pitt.edu}

Abstract

Role-Based Access Control (RBAC) models have emerged as a leading access control approach for today's information systems. Hybrid role hierarchies introduced in the Generalized Temporal RBAC model have shown to be very desirable for capturing fine-grained access control semantics. However, its administration can become significantly complex. Efficient techniques are needed to administer such hierarchies to support the development of high performance access control systems. In this paper, we present two approaches to implementing a hybrid role hierarchy in the context of the GTRBAC model and analyze and compare their complexities.

1. Introduction

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models, which have inherent limitations [2, 3, 8, 9]. Several beneficial features such as policy neutrality, support for least privilege and efficient access control management are associated with RBAC models [1, 9].

An important feature of the the RBAC models is the role hierarchy. A hierarchical relation between two roles defines permission acquisition and role activation semantics that can be utilized for efficiently and effectively structuring functional roles of an organization. Joshi *et al.* established a clear distinction between the three role hierarchies - *permission-inheritance*-only hierarchy (*I*-hierarchy), *activation*-only hierarchy (*A*-hierarchy), and the combined *permission-inheritance* and *activation* hierarchy (*IA*-hierarchy) - within the context of the Generalized Temporal RBAC model (GTRBAC) [4]. It has been shown that such a fine-grained hierarchical semantics facilitates capturing a wide range of security requirements, including the specification of fine-grained time-based SoD constraints on the roles including those within a hierarchy, time-constrained inheritance and activation relations, and user-centric or permission-centric cardinality constraints on roles in a hierarchy [3, 7, 9].

An important issue in authorization decision making process is to know which set of roles a user is allowed to activate in a single session. Furthermore, the system should be able to determine what roles are required to be activated to avail of a certain set of permissions. The significance of such information is the following:

1. When a user requests the activation of a set of roles which may be hierarchically related in a session, the authorization system should be able to determine whether to grant that request or not.
2. By determining which roles a user can activate within a single session, the authorization system can find and even resolve possible conflicts between the role hierarchy and the SoD constraints specified.
3. As the policy evolves a hierarchy may be transformed. In such a case, the original permission inheritance and role activation semantics need to be carefully maintained. Such transformations occur when roles in a hierarchy are deleted or modified, or new roles are added to the hierarchy. In emerging applications, policy evolution is a crucial issue as policies are very dynamic and constantly evolve [6].

Administering a role hierarchy, particularly in presence of other constraints (SoD, cardinality, and dependencies), can be seen as the most challenging aspect of RBAC systems [7, 10]. Efficient techniques are needed to maintain the permission inheritance and role activation semantics to support the scenarios indicated above and to support efficient administration and management of the RBAC policies. Earlier works have addressed modeling issues related to the administration of RBAC policies as well as hierarchy transformation [10]. However, they are limited to the traditional RBAC models where only *IA*-hierarchy is applicable. Inclusion of the three different types of hierarchies makes the issue of hierarchy administration significantly challenging. Joshi *et al*. have developed a theoretical basis for the analysis of a general hybrid hierarchy [7]. This paper is built on its theoretical results and describes the algorithms for supporting hierarchy administration. To the best of our knowledge, no other works have addressed the issue of administration and management of hybrid hierarchies.

0-7803-9093-8/05/$20.00 ©2005 IEEE.

In this paper, we describe a Java-based implementation of a hybrid role hierarchy and present associated algorithms for their analysis. The implementation presented here is based on our earlier work on hybrid hierarchies within the GTRBAC framework [4, 5, 6, 7]. In particular, we present two approaches to compute *uniquely activable sets of role sets* for a user assigned to a role within a hybrid hierarchy. The first approach is based on *decomposing* a hybrid hierarchy into its component linear components and then constructing the activable set by combining those associated with these components. The second approach involves using *inference rules* that facilitate the construction of the derived relations between indirectly related pairs of hierarchical roles. For formal details of these approaches we refer the readers to [6].

The paper is organized as follows. Section 2 presents a brief background on hybrid hierarchies. Section 3 discusses implementation approaches, and algorithms used in the implementation. In Section 4, we present the complexity analysis of the approaches presented in Section 3. In section 5, we present our conclusions and some future work.

2. Background

Among several benefits of an RBAC model is the role hierarchy. A role hierarchy provides for the efficient handling of permission distribution to users [9] by virtue of the role-role relationships that define permission-acquisition and role-activation semantics. That is, if a hierarchy uses the permission inheritance semantics, anyone explicitly assigned to a role also is authorized for its junior roles. Similarly, if a hierarchy uses the role-activation semantics, anyone explicitly assigned to a role can also activate its junior roles. In both cases, users assigned to the senior role need not be explicitly assigned to the junior roles.

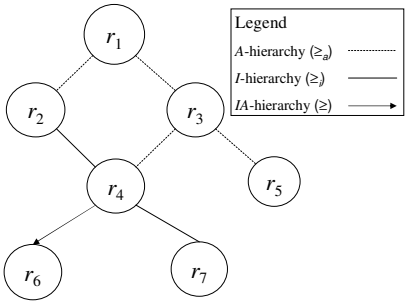

Figure 1: An example hybrid role hierarchy

2.1 Hybrid Role Hierarchy

Joshi *et al.* classify a role hierarchy into three types – *I*-hierarchy, *A*-hierarchy and *IA*-hierarchy. A hybrid hierarchy allows different hierarchical relations to coexist among roles in the hierarchy. Hybrid hierarchies have been shown to be desirable for capturing fine-grained access control requirements [4]. Figure 1 shows an example of a hybrid hierarchy.

2.2 Uniquely Activable Set (UAS)

In a hybrid hierarchy, maintaining permission acquisition and role activation semantics can become quite challenging. Joshi *et al.* introduce the concept of *Uniquely Activable Sets* (UAS), to facilitate the analysis of hybrid hierarchies and simplify the process of determining the activation and permission-acquisition sets [6]. Next we briefly introduce these concepts.

The UAS associated with a hierarchy is essentially the *set of role sets* that can be activated by a user assigned to a role of the hierarchy in a single session. Hence, UAS is mainly relevant from the perspective of the *principle of the least privilege*. We assume that H is a hierarchy with a single root (seniormost) role, S_H. UAS(H) represents the UAS associated with a user assigned to S_H. We use L to denote a monotype linear chain of hierarchically related roles and Lh to denote a hybrid linear chain of hierarchically related roles. LH will be used to refer to either L or Lh. The theorems in Table 1, presented in [7], formally characterize the UAS for a given hierarchy.

Table 1: Theorems to generate UAS [6]

Theorem 1(*Monotype Linear Hierarchy*): *For* $L = (X, <f>), <f> \in \{\geq_a \geq_i \geq\}$:

$$UAS(Lh, t) = \begin{cases} \{\{S_H\}\} & \text{if } (<f> = \geq_i) \\ 2^X \setminus \emptyset & \text{if } (<f> = \geq_a) \\ \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\} & \text{if } (<f> = \geq) \end{cases}$$

Theorem 2 (*Hybrid Linear hierarchy*):
$Lh = (L_1, LH_2)$, $L_1 = (X_1, <f_1>)$,
$LH_2 = (L_x, LH')$, $L_x = (X_x, <f_x>)$, $<f_x> \neq <f_1>$:
if $<f_1> = \geq_i$ *then* $UAS(Lh) = UAS(L_1)$
if $<f_1> = \geq_a$ *then*

$$UAS(Lh) = \begin{cases} UAS(L_1) & \text{if } (<f_x> = \geq_i) \\ UAS(L_1^{\upsilon}) \cup UAS(LH_2) & \\ \quad \cup (UAS(L_1^{\upsilon}) \otimes UAS(LH_2)) & \text{if } (<f_x> = \geq) \end{cases}$$

if $<f_1> = \geq$ *then*

$$UAS(Lh) = \begin{cases} UAS(L_1) & \text{if } (<f_x> = \geq_i) \\ UAS(L_1) \cup UAS(LH_2^{L}) & \\ \quad \cup (UAS(L_1) \otimes UAS(LH_2^{L})) & \text{if } (<f_x> = \geq) \end{cases}$$

Theorem 3 (*Hybrid Non-Linear Hierarchy*): $H = (X, [f]) = (LH_1, H_1) \exists\, x, y, z \in X,\ (x<f>y) \wedge (x<f>z)$:

- $UAS(H) = I \setminus C$, *where*
- $I = (UAS(LH_1) \cup UAS(H_1) \cup (UAS(LH_1) \setminus B \otimes UAS(H_1) \setminus B))$,
- $B = (UAS(LH_1) \coprod UAS(H_1))$, where, $X \coprod Y = \{X, Y \mid X \in UAS(LH_1), Y \in UAS(H_1)$ and $X \cap Y \neq \emptyset\}$, and
- $C = \{Z \mid Z \in I$, *such that* $\exists\, x, y \in Z, x \geq y\}$.

We use the notation LH^U to refer to the segment of the linear path component LH without the junior-most role. Similarly, we write LH^L to refer to the segment of the linear path component LH, without the senior-most element.

3. Role Hierarchy Management

In this section, we present the implementation and associated algorithms for hybrid hierarchy management.

3.1 Implementation Approaches

We present two techniques implemented to compute the UAS. The first is the *Decomposition* approach, which uses the Theorems in Table 1. The second is the *Derived Relations* approach, which uses inference rules from [6, 7]. Due to space limitations, we shall elaborate on the first approach and discuss the implementation of the second approach. We refer the readers to [6] for the details on the inference rules which have been shown to be sound and complete.

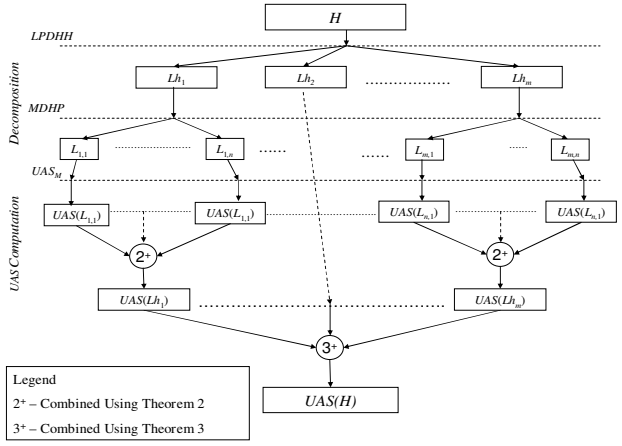

Figure 2: Decomposition approach

3.2 Decomposition Approach

The UAS is calculated in an incremental manner as shown in Figure 2. Initially, the role hierarchy is checked to determine if it is a monotype or hybrid hierarchy. If it is monotype, the UAS is not computed; instead, the hierarchy type information is stored and when the UAS is required, it is used to dynamically generate the UAS using Theorem 1 from Table 1.

If the hierarchy is hybrid, then the UAS is computed using Theorems 1, 2 and 3 from Table 1. The *Linear Path Decomposition of the Hybrid Hierarchy* (LPDHH) refers to the generation of individual linear paths for the senior-most role. Each components Lh_i, is decomposed into its linear monotype components. The UAS of each $L_{i,k}$ is calculated from Theorem 1 in Table 1. Then, the UASs of the $L_{i,k}$s are incrementally combined using Theorem 2 (denoted by $\oplus_2$), to obtain the UAS of Lh_i. Finally the UASs of the Lh_i are incrementally combined using Theorem 3 (denoted by $\oplus_3$), to obtain the UAS of the hybrid hierarchy with a root.

uasGeneration(r_1)
INITIALIZE the hierarchy H
IF *checkHybrid*() is FALSE
  $H.type = hierarchyType(H)$
ELSE
  Generate Linear Path Components $\{Lh_1, Lh_2, \ldots, Lh_m\}$
  FOR each Lhi, $1 \le i \le m$, DO
    Monotype decomposition of Lhi to obtain $\{L_{i,1}, L_{i,2}, \ldots. L_{i,n}\}$
    FOR each $L_{i,k}$, $1 \le k \le m$, DO
      $UL_{i,k} = uasLinearMonotype(L_{i,k})$
    $ULh_i = uasLinear(UL_{i,1}, UL_{i,2}, \ldots, UL_{i,n})$
  $UH = uasHierarchy(ULh_1, ULh_2, \ldots, ULh_m)$
RETURN UH

uasLinearMonotype$(L_{i,k})$
IF $L_{i,k}.type$="*I*-hierarchy", $UL_{i,k} = S_{L_{i,k}}$
IF $L_{i,k}.type$="*A*-hierarchy", $UL_{i,k} = 2^{|X|} - \varnothing$
IF $L_{i,k}.ype$="*IA*-hierarchy", $UL_{i,k} = \{r_0, r_1, \ldots.., r_n\}$, where $r_i \in L_{i,k}$, $0 \le i \le sizeof(L_{i,k})$
RETURN $UL_{i,k}$

uasLinear$(Lh_i, UL_{i,1}; UL_{i;2}, \ldots, UL_{i,n})$
FOR each pair of $UL_{i,k-1}$ and $UL_{i,k}$ $2 \le k \le n$, DO
  IF $L_{i,k-1}.type$="*I*-hierarchy" or $L_{i,k}.type$="*I*-hierarchy"
  $UL_{i,k-1} = UL_{i-k}$
  ELSE
    IF $L_{i,k-1}.type$="*A*-Hierarchy" and $L_{i,k}.type$="*IA*-Hierarchy"
      $UL_{i,k-1} = uasMonotype(UL^L_{i,k-1})$
    ELSE
      $UL_{i,k} = uasMonotype(UL^L_{i,k})$
    LET UL_{temp} be the set of all pairings of elements from $UL_{i,k-1}$ and $UL_{i,k}$
    ADD elements of UL_{temp} and $UL_{i,k}$ to $UL_{i,k-1}$
RETURN UL_1

uasHierarchy$(ULh_1, ULh_2, \ldots, ULh_m)$
FOR each pair of ULh_i and ULh_{i+1} $1 \le i \le m\text{-}1$, DO
  LET B be the set of pairings (s,t), such that s and t have no common elements.
  LET d_1 and d_2 be the sets of elements of ULh_i and ULh_{i+1}, respectively, without the elements of B.
  LET I be all pairings of elements from d_1 and d_2
  ADD elements of ULh_i and ULh_{i+1} to I
  LET C be the set of elements Z such that for all (x,y) where x and y are elements of Z and *immediateSenior*(*x, y,* "*IA*- Hierarchy")
  SET ULh_{i+1} as the set of elements of I that are not elements of C
RETURN ULh_m

Figure 3: Algorithm for UAS calculation

The algorithm for computing the UAS of a hierarchy is given in Figure 3. An important module in this implementation is the LPDHH. A depth-first-search approach is used to obtain the linear components. UAS generation is implemented as a sequential procedure, taking the role for which the UAS is to be computed as

input. The sequence of steps involved in the computation of the UAS is as follows:

1. Generate the hybrid linear path components by performing a DFS with path storing.
2. For each hybrid linear path component, decompose it into its monotype components.
3. For each monotype component apply Theorem 1 to obtain the UAS
4. Use Theorem 2 to compose the UAS for each hybrid linear path component given the UASs of its monotype components.
5. Find the composite UAS of the hierarchy given the UASs of the linear components, using Theorem 3.

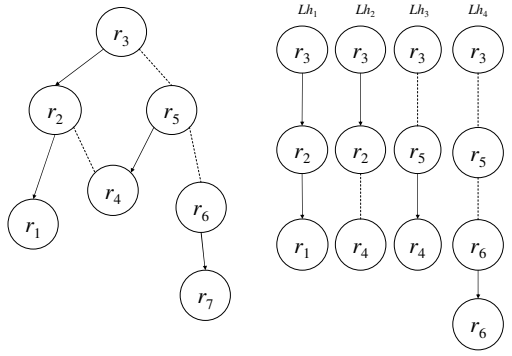

Figure 4: (a) A hybrid hierarchy, and (b) its linear component

uasLinearMonotype() to perform step 3, *uasLinear*() to perform step 4, and *uasHierarchy*() to perform step 5. The following example illustrates the decomposition approach to computing the UAS:

Example 1: Consider the hierarchy given in Figure 4. Let $S_H = r_3$. After the linear path decomposition of the hierarchy, we get the linear path components shown in the Figure 4, i.e.

H=(Lh_1, Lh_2, Lh_3, Lh_4).

Lh_1= L_{11} ; Lh_2= (L_{22}, L_{21})

`Rolesof`(L_{11})=(r_3,r_2,r_1); `Rolesof`(L_{21})=(r_2,r_4)

Lh_3= (L_{32}, L_{31}); Lh_4 = (L_{42}, L_{41})

`Rolesof` (L_{31})=(r_5,r_4); `Rolesof` (L_{32})=(r_3,r_5)

`Rolesof` (L_{41})=(r_3,r_5,r_6); `Rolesof` (L_{42})=(r_6,r_7)

1. $UAS(L_{11}) = \{\{r_3\},\{r_2\},\{r_1\}\}$
   $UAS(Lh_1) = \{\{r_3\},\{r_2\},\{r_1\}\}$
2. $UAS(L_{21}) = \{\{r_2\},\{r_4\},\{r_2,r_4\}\}$
   $UAS(L_{22}) = \{\{r_3\},\{r_2\}\}$
   $UAS(Lh_2) = \{\{\{r_4\}, \{r_2\}, \{r_3\},\{r_4,r_2\},\{r_4,r_3\}\}$
3. $UAS(L_{31}) = \{\{r_4\},\{r_5\}\}$
   $UAS(L_{32}) = \{\{r_4\},\{r_5\},\{r_4,r_5\}\}$
   $UAS(Lh_3) = \{\{r_3\}, \{r_5\},\{r_4\},\{r_5,r_3\},\{r_3, r_4\}\}$
4. $UAS(L_{41}) = \{\{r_3\},\{r_5\},\{r_3,r_5\},\{r_6\},\{r_3,r_6\}, \{r_5,r_6\},\{r_3,r_5,r_6\}\}$
   $UAS(L_{42}) = \{\{r_6\}, \{r_7\}\}$

$UAS(Lh_4) = \{\{r_3\}, \{r_5\}, \{r_6\}, \{r_7\}, \{r_3,r_5\}, \{r_3,r_6\}, \{r_3, r_7\}, \{r_5, r_6\}, \{r_5, r_7\}, \{r_3,r_5, r_6\}\}$

Thus: $UAS(H) = \{\{r_3\}, \{r_2\}, \{r_1\}, \{r_4\}, \{r_1,r_4\}, \{r_4,r_2\}, \{r_5\}, \{r_2,r_5\}, \{r_1,r_5\}, \{r_6\}, \{r_3,r_6\}, \{r_5,r_6\}, \{r_2,r_6\}, \{r_1,r_6\}, \{r_4,r_6\}, \{r_1,r_4,r_6\}, \{r_4,r_2,r_6\}\}$

derivedRelationsUAS(r_1)
Let P be the set of all possible combinations permission.
Let UAS be the set of minimal sets of role activations be.
Let permissions of r_1 be $P_1 \subseteq P$.
Add r_1 to UAS.
Start DFS
  FOR r_i such that $r_1 \geq_a r_i$, **DO**
    Let all permissions possessed by r_i be P_i.
    IF $\exists r$, $r_i \geq_i r$ or $r_i \geq r$
      Add permissions assigned to r to P_i.
      Add P_i to P, add r_i to UAS.
      FOR all $P_k \in P$, Let $P' = P_k \cup P_i$, **DO**
        IF $P' \notin P$, **THEN**
          add P' to P and add $UAS_j \cup r_i$ to UAS

Figure 5: Algorithm for derived relations approach

3.3 Derived Relations Approach

An alternate approach to resolving a "request to activate" a set of roles is to maintain a set of all possible relations that can be first derived from the hierarchy, and then to construct the UAS based on that. Joshi *et al*. introduced a set of inference rules to derive relations between any two roles of the hierarchy [7]. The resulting derived relations can be *I*, *A* or *IA*-hierarchical relations. From the definition of *A*-hierarchy, if a user u is assigned to a role say r_1, and r_1 is *A*-hierarchically related (directly or derived) to a set of roles say X', then *if* `can_activate`(u, r_1), holds, then for all r, $r \in X$', `can_activate`(u, r) also holds. For more details on this, refer to [6].

Computing the UAS from the derived relations, involves performing a DFS on the sub-hierarchy, rooted at r_1. For every $r \in X'$ encountered, a new role set is created with r, along with the permissions associated with this new role set, by virtue of their assignments to the roles. Figure 5 shows the algorithm for this approach. This procedure generates the minimal sets of roles that can be activated in *UAS*, to produce certain sets of permissions in P.

4. Complexity Analysis and Comparison

In this section, we analyze the complexities of the two approaches and compare them.

4.1 Decomposition Approach

For a monotype hierarchy, we shall not calculate the time complexity as we do not pre-calculate the UAS. Every

time an activation request is to be processed, a DFS is performed on the hierarchy to determine whether the roles can be allowed to be activated for a single session. The complexity of the DFS is be O(|X|+|H_e|), where |X| represents the number of roles in the hierarchy, and H_e represents the set of the hierarchical relations. *uasGeneration*() is the main function that initiates the computation of the UAS for a given role.

The function to generate the linear path decomposition of a hybrid hierarchy uses a DFS traversal on the hierarchy, given the set of hierarchical relations H_e. The complexity of a DFS operation on a graph $G = \{V,E\}$, where V is the set of nodes and E is the set of edges, is given as O(|V| +|E|). Thus the complexity of the decomposition step is:

$$O(|X|+|H_e|).$$

The function to generate the monotype components of a linear path, LH_i, performs a linear search on the roles in the component to generate its monotype components. Hence, the complexity of this function is O(|`Rolesof`(LH_i)|).

The function *uasLinearMonotype*($L_{i,k}$) generates the UAS for the monotype component $L_{i,k}$, which is dependant on the type of the hierarchy and the number of roles. The complexity of this function is given as: $uLM_{i,k} = O(2^{|L_{i,k}|})$

Function *uasLinear*(LH_i) generates the UAS for the linear component LH_i, by combining those of its monotype components ($L_{i,1}$, $L_{i,2}$,….$L_{i,n}$). The complexity of this function is dependant on the sizes of the UASs of its monotype components and is given as follows:

$$uL_i = O(uLM_{i,1} \times uLM_{i,2} \times .... \times uLM_{i,n})$$

Where n represents the number of linear monotype components obtained from the decomposition of LH_i. Function *uasHierarchy* (LH_i) generates the UAS of the hierarchy H, by combining those of its linear components (LH_1, LH_2,….LH_m). The complexity of this function depends on the sizes of the UASs of each of the linear components and is given as:

$$uH = O(uL_1 \times uL_2 \times ..... \times uL_m)$$

Where m represents the number of linear path components of H. From the algorithm, we see that the overall complexity of computing the UAS is the sum of the complexities of the LPDHH, the monotype decomposition of the linear paths and the complexity of *uasHierarchy*(). Thus we have

$$O(|X|+|H_e|+\sum_{i=1}^{n}\left|\texttt{Rolesof}(LH_i)\right| + uH \qquad (1)$$

When a request for activation is to be processed, a simple linear search of the stored UASs for the role r_i assigned to the user and a second linear search on UAS(r_i), will yield a result. Thus the complexity of processing the request is:

$$O(|X| + |UAS(r_i)|)$$

Figure 6a presents sample complexity plots for some specialized hierarchies. The graphs are "*log*(*Complexity*) *vs. No of Roles*" plots. Plots were obtained from random hierarchy types with the following structures:

- A monotype hierarchy of *A*-Hierarchy type
- A simple hybrid hierarchy, where the hybrid hierarchy has linear path components that are monotype.
- A general hybrid hierarchy that has hybrid linear path components. For the plot, we have assumed that each linear path component has no more than 3 linear monotype segments.

The complexities are calculated from (1).

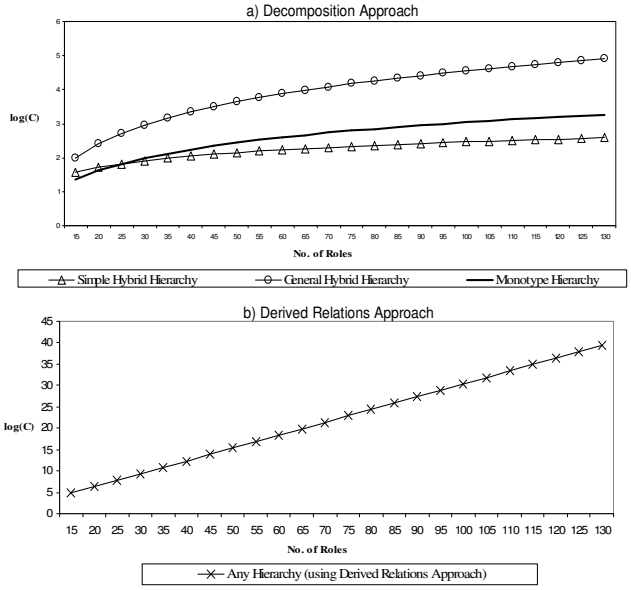

Figure 4: Complexities of computing the UAS

4.2 Derived Relations Based Approach

Determining all the direct and derived relations in a hierarchy is equivalent to a single DFS, which has a complexity of O(|X| + |H_e|). The complexity of computing the UAS from the derived relations is:

$$O((\sum_{i=1}^{|X'|} 2^i - 1) \times |X|) + |H_e|).$$

So the overall complexity of determining the UAS is

$$O((|X| + |H_e|) + ((\sum_{i=1}^{|X'|} 2^i - 1) \times |X|) + |H_e|) \qquad (2)$$

Figure 6b shows the complexity of the derived relation based approach. This approach has greater complexity with high values of log(C) and is independent of the hierarchy type.

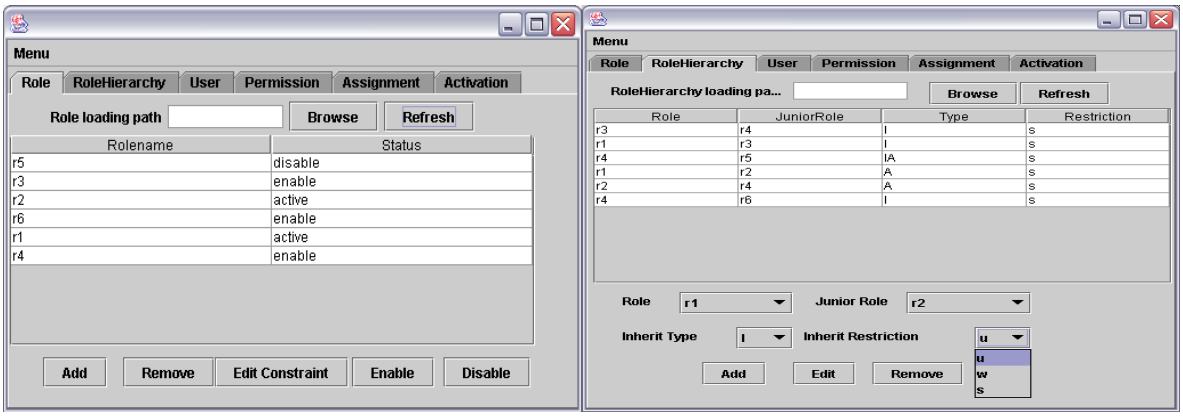

Figure 7: Snapshot of our administration module

4.3 Discussion

From the plots, we see that the complexity of calculating the UAS of a monotype hierarchy dynamically is significantly lower than that for a hybrid hierarchy. Furthermore, the complexity of computing the UAS for the simple hybrid hierarchy is also much lesser than that for the general hybrid hierarchy. This means if the hierarchy could be segmented into multiple monotype hierarchies or even simple hybrid hierarchies, then the complexity of computing the UAS for each and combining them together would be much lesser than that of computing the UAS for the general hybrid hierarchy. We are pursuing this approach currently. We can see that the decomposition approach is dependant on the sizes of the linear monotype segments, whereas the derived relations based approach is dependant on the size of the entire sub-hierarchy rooted at a role. This makes the derived relations approach more complex in terms of the processing time. The two approaches were tested in the implementation of our GTRBAC framework, adopted from [4]. The implementation in [4] does not consider hybrid hierarchies. Figure 8 shows the snapshots of hierarchy administration component in our implementation.

6. Conclusions and Future Work

In this paper, we have shown two techniques for computing the UAS of a hybrid hierarchy, the decomposition based and the derived relations based approaches. The decomposition approach is better in terms of complexity than the derived relations approach. As a future work, we plan to increase the efficiency of this approach by using segmentation of the hybrid hierarchy into component monotype hierarchies. Furthermore, the current implementation has not addressed the temporal constraints on hierarchically related roles. This work is also motivated by the need to support hierarchy transformations as the policy evolves. To provide support for evolution management, both UAS and derived relations need to be incrementally handled for a hierarchy. We are currently extending these algorithms to support such incremental updates.

References

[1] D. F. Ferraiolo, D. M. Gilbert, N Lynch. "*An Examination of Federal and Commercial Access Control Policy Needs*", Proceedings of NISTNCSC National Computer Security Conference, pages 107-116, September 20-23 1993.

[2] Luigi Giuri, *"Role-Based Access Control in Java™* ", In Proceedings of the third ACM Workshop on Role-based access control, 91-100, 1998

[3] J. B.D. Joshi, W. G. Aref, A. Ghafoor, E. H. Spafford. "*Security Models for Web-based Applications*". Communications of the ACM, 44(2), 38-72, 2001.

[4] James B D Joshi, Elisa Bertino, Arif Ghafoor, *"Temporal hierarchies and inheritance semantics for GTRBAC"*, Proceedings of the seventh ACM SACMAT, Monterey, California, USA, pages 74 – 83, 2002

[5] James B. D. Joshi, Basit Shafiq, Arif Ghafoor, Elisa Bertino, *"Dependencies and separation of duty constraints in GTRBAC"*, Proceedings of the eighth ACM SACMAT, Como, Italy, page 51 – 64, 2003

[6] James B D Joshi, Elisa Bertino, Arif Ghafoor, "*Formal Foundations for Hybrid Role Hierarchies in GTRBAC*", Submitted to ACM Transactions on Information and System Security.

[7] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor, *"A Generalised Temporal Role Based Access Control Model*". IEEE Transactions on Knowledge and Data Engineering, 17(1):4-23, Jan. 2005

[8] S. Osborn, R. Sandhu, Q. Munawer. "*Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies*". ACM Transactions on Information and System Security, 3(2):85-106, May 2000.

[9] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. "*Role-Based Access Control Models*". IEEE Computer 29(2): 38-47, IEEE Press, 1996

[10] Jason Crampton, and George Loizou, "*Administrative Scope: A Foundation for Administrative Models*", ACM Transactions on Information and System Security, 6(2), 201–231, 2003.