International Journal of Information and Computation Technology.
ISSN 0974-2239 Volume 4, Number 5 (2014), pp. 519-526

http://www. irphouse.com /ijict.htm

Visual Approach to Role Mining with Permission Usage Cardinality Constraint

V. Thangamani and V.Uma Maheswari

Department of Information Science and Technology, College of Engineering, Guindy Campus, AnnaUniversity, Chennai.

Abstract

Role Based Access Control (RBAC) is an effective way of managing permissions assigned to a large number of users in an enterprise. This paper offers a new role engineering approach to RBAC, referred to as visual role mining. The key idea is to graphically represent user-permission assignments to enable quick analysis and elicitation of meaningful roles with constraint. There are two algorithms: ADVISER and t-SMA_R. The former is a heuristic used to best represent the user-permission assignments of a given set of roles. The latter is a heuristic algorithm that, when used in conjunction with ADVISER, allows for a visual elicitation of roles with permission usage cardinality constraint.

Index Terms: Access controls, data and knowledge visualization, mining methods, constraint.

1. Introduction

Access control is the process of mediating requests to data and services maintained by a system, determining which requests should be granted or denied [3]. Significant research has focused on providing formal representation of access control models. Among all proposed models, Role Based Access Control (RBAC) [5] has become the norm in most organizations. This success is greatly due to its simplicity: a role identifies a set of permissions; users, in turn, are assigned to roles based on their responsibilities. To implement a RBAC system, it is important to devise a complete set of roles. This design task, known as role engineering [2], has been recognized as the costliest part of a RBAC-oriented project. Recently, there has been an increasing interest in using automated role engineering techniques [7]. All of them seek to identify de facto roles embedded in existing access permissions. Since these

approaches usually resort to data mining techniques, the term role mining is often used as a synonym. Despite much work dedicated to the design of role mining algorithms, existing techniques deal with three main practical issues: meaning of elicited roles, noise within data, and correlations among roles.

To address the aforementioned issues, visual role mining is devised [1]. RBAC roles are managed as visual patterns. The rationale behind this approach is that visual representations of roles can actually amplify cognition, leading to optimal analysis results [4].Visualization of the user-permission assignments is performed in such a way to isolate the noise, allowing role engineers to focus on relevant patterns without resorting to traditional role mining tools. Further, correlations among roles are shown as overlapping patterns, hence providing an intuitive way to discover and utilize these relations. The proper representation of user-permission assignments allows role designers to gain insight, draw conclusions, and design meaningful roles from both IT and business perspectives.

Constraints are an important aspect of RBAC and sometimes argued to be the principal motivation of RBAC. Recently, the problem of defining different kind of constraints on the number and the size of the roles included in the resulting role set has been addressed. Permission Usage cardinality constraint is one of the cardinality constraint which restricts the maximum number of permissions that can be included in a role. Cardinality constraints on the number of permissions included in a role have been firstly considered in [6], and a heuristic algorithm called Constrained Role Miner (CRM) has been proposed. The remainder of this paper is organized as follows. Section II provides detailed description of visual approach with permission usage constraint. The algorithms ADVISER and t-SMA_R are discussed in this section. Section III shows the experimental results and finally, section IV provides conclusion.

2. Heuristics

This section describes two heuristics such as ADVISER (Access Data VISualizER) and t-SMA_R(t-Simple role Mining Algorithm). The overall system architecture is represented by Fig. 1.

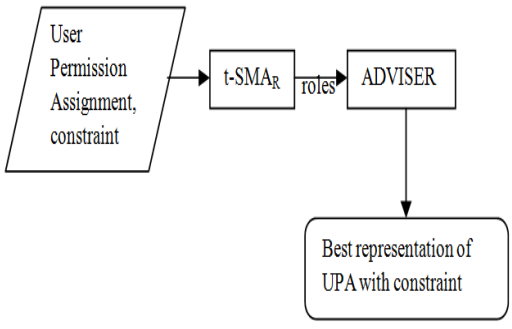


Fig. 1: System Architecture.

2.1 ADVISER

This algorithm is able to provide a compact representation of a given set of roles. In particular, it reorders rows and columns of the user-permission matrix to minimize the fragmentation of each role. Fig. 2 explains ADVISER Algorithm.

```
1: procedure ADVISER(USERS, PERMS, ROLES, UA, PA)
2:     σ_U ← SORTSET(USERS, UA, ROLES)
3:     σ_P ← SORTSET(PERMS, PA, ROLES)
4:     return σ_U, σ_P
5: end procedure

6: procedure SORTSET(ITEMS, IA, ROLES)
7:     ITEMS̄ ← {I ⊆ ITEMS | ∀i, i' ∈ I, roles(i) = roles(i')}
8:     σ ← ∅
9:     for all I ∈ ITEMS̄ sorted by descending areas of roles(I) do
10:        if |σ| < 2 then
11:            σ.append(I)
12:        else
13:            if Jacc(I, σ.first) > Jacc(I, σ.last) then
14:                p ← 1,          j ← Jacc(I, σ.first)
15:            else
16:                p ← |σ| + 1,    j ← Jacc(I, σ.last)
17:            end if
18:            for i = 2 . . . |σ| do
19:                j_prec ← Jacc(I, σ[i − 1]),   j_succ ← Jacc(I, σ[i])
20:                j_curr ← Jacc(σ[i − 1], σ[i])
21:                if max{j_prec, j_succ} > j ∧ min{j_prec, j_succ} ≥ j_curr then
22:                    p ← i,   j ← max{j_prec, j_succ}
23:                end if
24:            end for
25:            σ.insert(p, I) {between the (p − 1)th and the pth elements}
26:        end if
27:    end for
28:    return σ.expand
29: end procedure
```

Fig. 2: ADVISER Algorithm.

Detailed description follows

1. Rows and columns are sorted independently. ADVISER decomposes the optimal matrix-permutation problem into two sub problems, that is users (Line 2) and permissions (Line 3) are sorted independently. Due to this symmetry, from now on we generically refer to rows and columns as items. If some items are assigned to the same set of roles, they are put together. For this reason, the algorithm sorts groups of items, called item sets, instead of individual items. Line 7 identifies items assigned to the same roles.
2. Item set positions are decided one-by-one. In order to facilitate a better representation of large roles, item sets involving roles with larger areas are analyzed first. Line 9 implements this behavior. Let I,I` be two item sets.

assusers and assperms denotes the number of users and number of permissions respectively. I is consider before I` only if

Max |assusers(r)*assperms(r)| > Max|assusers(r`)*assperms(r`)|

r€roles(I)\roles(I`) r`€roles(I`)\roles(I`)

when roles(I)\roles(I`) or roles(I`)\roles(I`)=0 then Max=0.

3. The algorithm tries to avoid large gaps by putting item sets close to each other when they share large roles. For this Jaccard coefficient metric is used.Calculate jaccard coefficient for ranking the similarity of items.

$$Jacc(\{a\},\{b\}) = \frac{\sum_{r \in roles(a) \cap roles(b)} |m(r)|}{\sum_{r \in roles(a) \cup roles(b)} |m(r)|}$$

where m(.) is the membership function assusers(.) when we sort permissions, or the function assperms(.) when we sort users.

4. Each item set is preferentially positioned at the beginning or at the end of already sorted item sets. The idea is to avoid to "worsen" already found, high similarities. Lines 10-26 implement the item set sorting strategy by deciding a position p for the item set I in an item set permutation. The first two item sets are just inserted in the first two positions (lines 10-11). Then, subsequent item sets are inserted among already-sorted item sets only when this operation actually improves the existing sorting.
5. Item set sorting is converted to item sorting. This is the When all item sets in ITEMS have been sorted, they are "expanded" (Line 28) to return the ordering of each single item in ITEMS—instead of providing an ordering for group of items that share the same roles.

2.2. t-SMA$_R$

This algorithm takes as input the matrix UPA and returns a complete role set satisfying the cardinality constraint (i.e., at most t permissions are associated to each role).

The basic idea is to select from UPA all rows having less than t permissions in an order that will be defined below. Such rows will correspond to candidate roles that will be added to the candidate role-set. If there is no row having at most t permissions, then a row is selected and, t of the permissions included in the row is chosen. The selected permissions induce a role that is added to the candidate role-set. Then, all rows covered by the candidate row-set are removed from UPA and the procedure is iterated until the UPA matrix contains some rows. The above sketched procedure is more formally described by t-SMA$_R$ algorithm where we use the following notation. Given an a x b binary matrix M, for $1 \leq i \leq a$, with M[i], |M[i]| denotes the M`s i-th row and number of one's appearing in M[i] respectively.

The procedures numCols (M) and numRows (M), return the number of columns and rows, respectively, of the matrix M. For a set S and integer h, the procedure first(S,h) returns the first h elements listed in the set S. Given a user-permission

assignment matrix UPA, a new candidate role is generated by selecting a UPA's row having the least number of permissions with ties broken at random (Lines 6-8 of t-SMA_R). If the number of permissions associated to the selected row is at most t, then a new role is created (Line 9 of t-SMA_R). The new role, containing all permissions associated to the selected row, is then added to the candidate role-set (Line 21 of t-SMA_R). In this algorithm, the matrix uncoveredP represents the user's permissions that are not covered by the roles in candidateRoles. After discover a role to be added to the candidateRoles set, running setToZero Algorithm. All rows whose permissions are covered by the candidate roles are removed from both matrices uncoveredP and UPA removeCoveredUsers's pseudo-code is quite similar to the pseudo-code for setToZero. Algorithm halts when all UPA's rows have been removed (Line 5 of t-SMA_R). If the number of permissions exceeds the cardinality constraint, then two possible ways of selecting the role to be added to the candidate role-set have been considered. These two possibilities gave rise to two heuristics referred to as t- SMA_R-0 and t-SMA_R-1, respectively. Fig. 3 and Fig. 4 explains the t-SMA_R and SetToZero Algorithm respectively,

Algorithm 1 t-SMA_R(UPA, t, *selection*)

```
1: candidateRoles ← ∅
2: uncoveredP ← UPA
3: np ← NUMCOLS(UPA) /* np is equal to the number of permissions */
4: nr ← NUMROWS(UPA) /* nr is equal to the number of users */
5: while NUMROWS(UPA) > 0 do
6:    m ← min{|UPA[i]| : 1 ≤ i ≤ nr}
7:    candidateRows ← {i : 1 ≤ i ≤ nr and |UPA[i]| = m}
8:    selectedRow ←R candidateRows
9:    newRole ← {pj : 1 ≤ j ≤ np and UPA[selectedRow][j] = 1}
10:   if |newRole| > t and selection == 0 then
11:      newRole ← first(newRole, t)
12:   else if |newRole| > t and selection == 1 then
13:      m ← min{|uncoveredP[i]| : 1 ≤ i ≤ nr}
14:      candidateRows ← {i : 1 ≤ i ≤ nr and |uncoveredP[i]| = m}
15:      selectedRow ←R candidateRows
16:      newRole ← {pj : 1 ≤ j ≤ np and uncoveredP[selectedRow][j] = 1}
17:      if |newRole| > t then
18:         newRole ← first(newRole, t)
19:      end if
20:   end if
21:   candidateRoles ← candidateRoles ∪ {newRole}
22:   uncoveredP ← setToZero(UPA, uncoveredP, newRole)
23:   UPA ← REMOVECOVEREDUSERS(UPA, candidateRoles)
24: end while
25: return candidateRoles
```

Fig. 3: t-SMAR Algorithm.

In t-SMA$_R$-0 (i.e., when selection is set to 0 in t-SMA$_R$), the new role will simply contain the first t permissions associated to the selected row (Lines 10-11 of t-SMA$_R$). While, in t-SMA$_R$ -1 (i.e., when selection is set to 1 in t-SMA$_R$) select a row (Lines 13-16 of t-SMA$_R$) of the matrix uncoveredP having the least number of permissions, ties broken at random. In other words, selects a row (i.e, a users) having the least number of permissions still uncovered. If the selected row is associated to more than t permissions, then the new role will only include its first t permissions (Lines 17-19 of t-SMA$_R$). This algorithm returns a set of roles (i.e., rows and subsets of rows) exactly covering the UPA matrix. The following SetToZero pseudo code updates the matrix uncoveredP according to newRole.

Algorithm 2 SETTOZERO(UPA, *uncoveredP*, *newRole*)

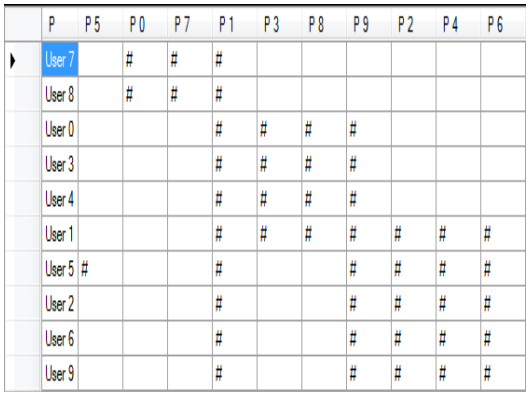
```
1: np ← NUMCOLS(UPA)
2: nr ← NUMROWS(UPA)
3: for i = 1 to nr do
4:    permissions ← {p_j : 1 ≤ j ≤ np and UPA[i][j] = 1}
5:    if newRole ⊆ permissions then
6:       for all j such that p_j ∈ newRole do
7:          uncoveredP[i][j] ← 0
8:       end for
9:    end if
10: end for
11: Remove from uncoveredP all-zeroes rows
12: return uncoveredP
```

Fig. 4: SetToZero Algorithm.

The roles produced by t-SMA$_R$ are used in conjunction with ADVISER, allows for a visual elicitation of roles with permission usage cardinality constraint.

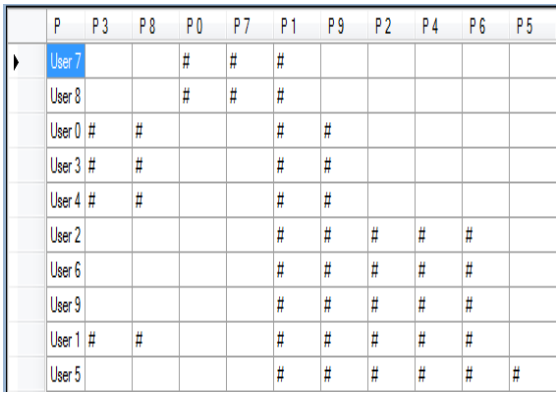
3. Experimental Results

In this section results obtained by the evaluation of ADVISER and t-SMA$_R$ are discussed. Fig. 5 shows the results obtained when using ADVISER fed with the predefined roles. Fig. 6 shows the results obtained when using ADVISER fed with the roles generated by t-SMA$_R$.



	P	P5	P0	P7	P1	P3	P8	P9	P2	P4	P6
▸	User 7		#	#	#						
	User 8		#	#	#						
	User 0				#	#	#	#			
	User 3				#	#	#	#			
	User 4				#	#	#	#			
	User 1				#	#	#	#	#	#	#
	User 5	#			#			#	#	#	#
	User 2				#			#	#	#	#
	User 6				#			#	#	#	#
	User 9				#			#	#	#	#

Fig. 5: Visual representation of predefined roles



	P	P3	P8	P0	P7	P1	P9	P2	P4	P6	P5
▸	User 7			#	#	#					
	User 8			#	#	#					
	User 0	#	#			#	#				
	User 3	#	#			#	#				
	User 4	#	#			#	#				
	User 2					#	#	#	#	#	
	User 6					#	#	#	#	#	
	User 9					#	#	#	#	#	
	User 1	#	#			#	#	#	#	#	
	User 5					#	#	#	#	#	#

Fig. 6: Visual Representation of roles with constraint

4. Conclusion

Devising a complete set of roles is necessary to implement a RBAC system. This is accomplished by bottom up approach called Role Mining. The bottom up approach starts with existing user permission assignments and attempts to derive roles from them.Visual approach to role mining simplifies the role engineering process. ADVISER algorithm is implemented to represent the user permission assignments in a better way. This representation in matrix format enables quick analysis and elicitation of meaningful roles. t-SMA_R algorithms would be implemented for permission usage cardinality constraint which prevents the overburdening of a user with a large number of permissions.

References

[1] A. Colantonio, R. Di Pietro, A. Ocello, and N.V. Verde, "Visual Role Mining:A Picture Is Worth a Thousand Roles," IEEE Trans. Knowledge andData Eng., vol. 24, no. 6, pp. 1120-1133, June. 2012.

[2] E.J. Coyne, "Role-Engineering," Proc. ACM Workshop Role-Based Access Control (RBAC '95), pp. 15-16, 1995.

[3] S. DeCapitani , S. Foresti, P. Samarati, and S. Jajodia, "Access Control Policies and Languages," Int'l J. Computational Science and Eng., vol. 3, no. 2, pp. 94-102, 2007.

[4] J.D.Fekete, J.J. Wijk, J.T. Stasko, and C. North, "The Value of Information Visualization," Information Visualization: Human- Centered Issues and Perspectives, pp. 1-18, 2008.

[5] D.Ferraiolo, R.S.Sandhu, S.Gavrila, R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," ACM Trans. Information and System Security, vol. 4, pp. 224-274,2001.

[6] R. Kumar, S. Sural, and A. Gupta, "Mining rbac roles under cardinality constraint",Proc.6th international conference on Information systems security(ICISS'10),pp.171-185, 2010

[7] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo, "Evaluating Role Mining Algorithms," Proc. 14th ACM Symposium on Access Control Models and Technologies (SACMAT '09), pp. 95-104, 2009.