# Role Mining with ORCA

Jürgen Schlegelmilch
OFFIS e. V.
Escherweg 2
26121 Oldenburg, Germany
juergen.schlegelmilch@offis.de

Ulrike Steffens
OFFIS e. V.
Escherweg 2
26121 Oldenburg, Germany
ulrike.steffens@offis.de

## ABSTRACT

With continuously growing numbers of applications, enterprises face the problem of efficiently managing the assignment of access permissions to their users. On the one hand, security demands a tight regime on permissions; on the other hand, users need permissions to perform their tasks. Role-based access control (RBAC) has proven to be a solution to this problem but relies on a well-defined set of role definitions, a role concept for the enterprise in question. The definition of a role concept (role engineering) is a difficult task traditionally performed via interviews and workshops. However, often users already have the permissions that they need to do their jobs, and roles can be derived from these permission assignments using data mining technology, thus giving the process of role concept definition a head-start.

In this paper, we present the ORCA role mining tool and its algorithm. The algorithm performs a cluster analysis on permission assignments to build a hierarchy of permission clusters and presents the results to the user in graphical form. It allows the user to interactively add expert knowledge to guide the clustering algorithm. The tool provides valuable insights into the permission structures of an enterprise and delivers an initial role hierarchy for the definition of an enterprise role concept using a bottom-up approach.

## Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Access controls*; H.1.2 [**Information Systems**]: Models and Principles—*User/Machine Systems, Human factors*; K.6.5 [**Computing Milieux**]: Security and Protection

## General Terms

Security, Management, Algorithms, Human Factors

## Keywords

Role-based access control, role definition, role hierarchy, role engineering, role mining, data mining, cluster analysis


*SACMAT'05,* June 1–3, 2005, Stockholm, Sweden.


## 1. INTRODUCTION

Since the early 1990s [7], Role-Based Access Control has become more popular. There are standard models (RBAC96 [23], NIST2001 [4]) and active research on both extensions [23, 22, 15, 10, 11] and methodologies [5, 17, 6]. Mainly enterprises with many users and high security demands like banking companies are now considering RBAC and role concepts, and therefore facing the problem of how to define roles. This has prompted new approaches in role engineering. The traditional top-down approach [17, 6] has been complemented with bottom-up approaches using data mining techniques [24]. This application of data mining to identify roles from existing data is called role mining.

The few proposals for role mining differ in the choice of algorithm as well as source data. In this paper, we discuss role mining based on existing permission assignments and present ORCA, the OFFIS Role mining tool with Cluster Analysis, and its algorithm. This algorithm builds a hierarchy of permission clusters using a bottom-up cluster analysis. The tool provides the user with valuable insights into the existing permission structures and allows to iteratively transform the cluster hierarchy into an initial role hierarchy.

The remainder of this document is organised as follows: Section 2 describes role mining along with its benefits and pitfalls. Section 3 presents a data mining algorithm tailored for role mining. The algorithm has been implemented within the ORCA tool which is introduced in Section 4. We discuss some related work in Section 5 before closing with a conclusion and a prospect on future work in Section 6.

## 2. THE CONCEPT OF ROLE MINING

Role engineering is a tedious, error-prone and politically difficult task. Once role definitions are established, there is no more slack to shift responsibilities or to demand additional permissions. So, people are hesitant to specify roles and reluctant to co-operate. Any tool supporting the process of role engineering with objective data helps putting discussions on a firm base and avoids intentional as well as accidental errors, thereby accelerating the process.

There are generally two approaches to role engineering [6]: Either working top-down from an initial description to roles and permissions or else aggregating permissions bottom-up into roles. The first approach starts with business process definitions or scenarios, extracts role candidates from these descriptions, and then transforms them into an enterprise role concept [5, 17, 18]. The roles are then fitted with the necessary permissions. This approach is time-consuming and may deliver process descriptions as an unwanted side-

product (unwanted because of their potential for process optimization and hence reorganisation). It also requires early co-operation of employees and/or experts and is harder to support with a tool: the necessary knowledge is in people's minds and has to be externalised first.

The second approach starts bottom-up by analysing artifacts of roles [12] and then transforms and aggregates them into the roles themselves. The idea here is that roles are already implicitly in use [23] and have to be identified rather than defined. Roles describe tasks within a process and in modern enterprises, these tasks often involve applications. So, to act in a role a person has to access applications and therefore has to have permissions. In short, roles leave patterns in the permission assignments and it is possible to find those patterns using data mining technology. This provides the role engineering process with an impartial and reliable base for discussions and refinements.

*Caveats of Role Mining.* The usual precautions of data mining apply to role mining, too: The more permissions are needed for a role, and the more specialised they are, the better will be the mining result. For example, the permission to use Microsoft Word is not very specific and will therefore not distinguish any role. On the other hand, access rights for a geo-information system, a database of telecommunication lines, and certain transactions in an ERP system may well identify a specific engineering role in a Tele-Communications company.

Of course, data mining in permission assignments will not automagically deliver a complete and error-free role concept. Not all roles have an impact on permission assignments, and even then, there are a number of considerations to make.

- First of all, unless a permission management system is used – which is likely to use roles anyway – there will be noise in the data, for example access rights that have been granted erroneously or exceptionally. So, prior to role mining data cleansing [8] must be applied to weed out anomalies.
- Second, even single functions in typical enterprise applications may support more than one role, and associated permissions will hence be ambiguous. The effect is worse if only few systems and their access control can be considered, as motivated above.
- Third, employees in an enterprise usually have more than one role, so that associated permissions will be mixed up and hard to group by role. Some combinations of permissions may be entirely accidental, or due to effects like people retaining permissions despite changed responsibilities.
- Fourth, one person may have several digital identities. This may hide useful combinations of permissions if they may belong to different identities of one person. Such multiple identities conflict with RBAC [23] in general and should be replaced with multiple roles for one identity.
- Fifth, a combination of permissions does not provide a name or semantics for the role it may represent.

Reviewing of the mining result with experts is therefore indispensable. However, we believe the mining result to still provide a more reliable base for decisions than the early rounds of approaches based on interviews and expert judgement alone, and thus to deliver better role concepts, giving the bottom-up approach a head-start into the whole role definition process.

# 3. DISCOVERING ROLE PATTERNS

Patterns of roles can be found in the access control of various application systems in a company. Any person acting in a role has to have permissions for some applications and the combination of these permissions hints at the role. There may be other roles that need the very same combination, so there is no one-to-one correspondence. However, the more applications and permissions we consider, the better can we identify roles with certain combinations.

With data mining, we are able to retrace this relationship and look for combinations of permissions. If we find a significant set of persons sharing the same set of permissions, we can assume that those persons share – at least part of their time – a role, namely one that requires those permissions. So we scan the permission assignments for combinations shared by many persons and propose them as roles. This technique is known as cluster analysis: We put permissions into a cluster iff a significant number of persons share them. A cluster $c$ has two characteristics:

- a set of permissions: $\mathit{rights}(c)$
- a set of persons, namely those that have all the permissions: $\mathit{members}(c)$

So, the cluster is defined by the permissions that it combines. The clustering algorithm groups permissions into clusters, associates any person with the cluster that has all the permissions in the cluster, and keeps those clusters with a significant set of persons. To avoid the combinatorial explosion of all permissions we chose a hierarchical bottom-up algorithm (a Single Linkage variant [9]) and consider not all sets of permissions but only those defined by combinations of clusters.

Let $\mathsf{Persons}$ be the (given) set of all persons, $\mathsf{Clusters}$ the set of all clusters, and $\prec \subseteq \mathsf{Clusters} \times \mathsf{Clusters}$ a partial order on clusters.

1. Initialise the variables:
$$\mathsf{Clusters} := \emptyset$$
$$\prec := \emptyset$$
2. For all single permissions $r$, define clusters $c_r$ with
$$\mathit{rights}(c_r) = \{r\}$$
$$\mathit{members}(c_r) = \{p \in \mathsf{Persons} : p \text{ has permission } r\}$$
and add the new clusters $c_r$ to the set of all clusters:
$$\mathsf{Clusters} := \mathsf{Clusters} \cup \{c_r\}$$
3. Now, find the pairs of clusters with a maximal overlap among their members. For this, we only consider clusters that have not already been placed in the hierarchy, namely the maximal elements of $\prec$:
$$\top_\prec = \{c \in \mathsf{Clusters} : \neg\exists d \in \mathsf{Clusters} : c \prec d\}$$
For any pair $\langle c, d\rangle$ of clusters $c, d \in \top_\prec$ define
$$\mathit{members}(\langle c, d\rangle) = \mathit{members}(c) \cap \mathit{members}(d)$$
$$\mathit{rights}(\langle c, d\rangle) = \mathit{rights}(c) \cup \mathit{rights}(d)$$

**Table 1: Activities and permissions**

| | *rAcc* | *wAcc* | *cdAcc* | *cTrans* | *rTrans* |
|---|---|---|---|---|---|
| CH | × | × | | | |
| MT | × | | | × | |
| AT | × | | | | × |
| MA | | | × | | |

The maximum member overlap between clusters is defined as

$$m = \max\{|members(\langle c, d\rangle)| : c, d \in \top_{\prec}\}$$

So, these are the pairs with maximal overlap:

$$S = \{\langle c, d\rangle : |members(\langle c, d\rangle)| = m \wedge c, d \in \top_{\prec}\}$$

And with $r = \max\{|rights(\langle c, d\rangle)| : \langle c, d\rangle \in S\}$, pick from $S$ the tuples with the largest set of rights:

$$E = \{\langle c, d\rangle : |rights(\langle c, d\rangle)| = r \wedge \langle c, d\rangle \in S\}$$

Now, randomly choose a pair $\langle c, d\rangle$ from $E$ and define the new cluster $e$ with

$$rights(e) = rights(c) \cup rights(d)$$
$$members(e) = members(c) \cap members(d)$$

In the cluster hierarchy, $e$ is placed as a super-cluster above both $c$ and $d$ ($c$ and $d$ are sub-clusters of $e$):

$$\mathsf{Clusters} := \mathsf{Clusters} \cup \{e\}$$
$$\prec := \prec \cup \{\langle c, e\rangle, \langle d, e\rangle\}$$

This excludes both $c$ and $d$ from further iterations of this step since they are no longer elements of $\top_{\prec}$.

4. Repeat Step 3 until Clusters is stable.

The algorithm works its way up in the set hierarchy of permissions and stops when there are no suitable pairs of clusters with overlapping member sets.

### *An example*

Imagine a bank where employees perform tasks like handling cash payments (deposits or withdrawals), prepare or approve money transfers, or manage accounts. Let us assume that

- Cash handling (CH) requires read and write access to customer accounts (*rAcc*, *wAcc*).
- Making transfers (MT) requires read access to customer accounts (*rAcc*) and create access to internal transfer transactions (*cTrans*).
- Approval of transfers (AT) requires read access to customer accounts (*rAcc*), plus read access to internal transfer transactions (*rTrans*).
- Managing accounts (MA) requires create/delete access to customer accounts. (*cdAcc*)

These assumptions are summarised in Table 1.

Our sample bank has 6 employees with permission assigments as shown in Table 2.

**Table 2: Permission assignments**

| | *rAcc* | *wAcc* | *cdAcc* | *cTrans* | *rTrans* |
|---|---|---|---|---|---|
| Ann | × | × | | × | |
| Bob | × | × | | × | |
| Carl | × | | × | | × |
| Doro | × | | × | | × |
| Ed | × | × | | × | |
| Fay | × | × | × | × | × |

### *Constructing the cluster hierarchy.*

Now we apply the algorithm to construct the cluster hierarchy.

1. In the first two steps, variables are initialised and single clusters for all five permissions are created:

   $$\mathsf{Clusters} := \{c_{rAcc}, c_{wAcc}, c_{cdAcc}, c_{cTrans}, c_{rTrans}\}$$

2. Now, Step 3 builds pairs of clusters and compares their amount of member overlap. There are

   - 3 pairs with overlap 4:
     $\langle c_{rAcc}, c_{wAcc}\rangle$, $\langle c_{rAcc}, c_{cTrans}\rangle$, $\langle c_{wAcc}, c_{cTrans}\rangle$
   - 3 pairs with overlap 3:
     $\langle c_{rAcc}, c_{cdAcc}\rangle$, $\langle c_{rAcc}, c_{rTrans}\rangle$, $\langle c_{cdAcc}, c_{rTrans}\rangle$
   - 3 pairs with overlap 1:
     $\langle c_{wAcc}, c_{rTrans}\rangle$, $\langle c_{wAcc}, c_{cdAcc}\rangle$, $\langle c_{cTrans}, c_{rTrans}\rangle$

   The algorithm randomly picks a pair from the list for overlap 4 and creates the cluster $c_1$ with

   $$rights(c_1) = \{rAcc, wAcc\}$$
   $$members(c_1) = \{\text{Ann}, \text{Bob}, \text{Ed}, \text{Fay}\}$$

   making it a super-cluster of the constituent clusters:

   $$\prec := \prec \cup \{\langle c_{rAcc}, c_1\rangle, \langle c_{wAcc}, c_1\rangle\}$$

   Hence we get $\top_{\prec} = \{c_{cdAcc}, c_{cTrans}, c_{rTrans}, c_1\}$.

3. The next iteration of Step 3 finds

   - 1 pair with overlap 4: $\langle c_{cTrans}, c_1\rangle$
   - 1 pair with overlap 3: $\langle c_{cdAcc}, c_{rTrans}\rangle$
   - 3 pairs with overlap 1:
     $\langle c_{cTrans}, c_{rTrans}\rangle$, $\langle c_1, c_{rTrans}\rangle$, $\langle c_1, C_{cdAcc}\rangle$

   So, the cluster $c_2$ is created with

   $$rights(c_2) = \{rAcc, wAcc, cTrans\}$$
   $$members(c_2) = \{\text{Ann}, \text{Bob}, \text{Ed}, \text{Fay}\}$$

   This cluster is a super-cluster of $c_{cTrans}$ and $c_1$:

   $$\prec := \prec \cup \{\langle c_{cTrans}, c_2\rangle, \langle c_1, c_2\rangle\}$$

   We now have $\top_{\prec} = \{c_{cdAcc}, c_{rTrans}, c_2\}$

4. In the next iteration of Step 3, we get:

   - 1 pair with overlap 3: $\langle c_{cdAcc}, c_{rTrans}\rangle$
   - 2 pairs with overlap 1: $\langle c_2, c_{cdAcc}\rangle$, $\langle c_2, c_{rTrans}\rangle$

   Now the algorithm creates the cluster $c_3$ with

   $$rights(c_3) = \{cdAcc, rTrans\}$$
   $$members(c_3) = \{\text{Carl}, \text{Doro}, \text{Fay}\}$$

   and makes it a super-cluster of $c_{cdAcc}$ and $c_{rTrans}$:

   $$\prec := \prec \cup \{\langle c_{cdAcc}, c_3\rangle, \langle c_{rTrans}, c_3\rangle\}$$

   $\top_{\prec}$ is now $\{c_2, c_3\}$

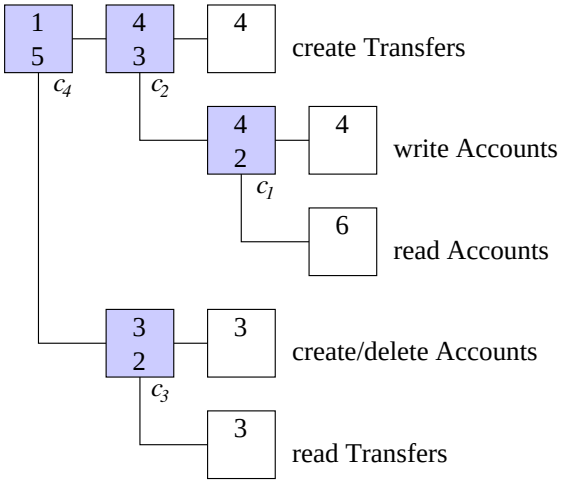


**Figure 1: Cluster hierarchy of the example**

5. The next iteration of Step 3 yields just one pair, namely $\{c_2, c_3\}$. So we get a new cluster $c_4$ with

$$\begin{aligned} \mathit{rights}(c_4) &= \mathit{rights}(c_2) \cup \mathit{rights}(c_3) \\ &= \{\mathit{rAcc}, \mathit{wAcc}, \mathit{cdAcc}, \mathit{cTrans}, \mathit{rTrans}\} \\ \mathit{members}(c_4) &= \mathit{members}(c_2) \cap \mathit{members}(c_3) \\ &= \{\text{Fay}\} \end{aligned}$$

   with $\prec := \prec \cup \{\langle c_2, c_4\rangle, \langle c_3, c_4\rangle\}$ and $\top_\prec = \{c_4\}$.

6. Since $\top_\prec$ has now only one element, no candidate pairs can be formed, hence no further clusters created. So, Clusters is stable and the algorithm stops.

The resulting cluster hierarchy is shown in Figure 1. Each rectangular node represents a cluster, with the number of members $|\mathit{members}(c)|$ and below that, for inner nodes, the number of permissions $|\mathit{rights}(c)|$. The partial order among the clusters is shown as connecting lines, with super-clusters left of sub-clusters. Next to trivial clusters, the name of their only permission is given.

*Deriving the role hierarchy.* Now that we have a cluster hierarchy, we can try to derive a role hierarchy from it. Each of the clusters might represent a role, with the leaf clusters being unlikely candidates. The hierarchy in Figure 1 has four non-trivial clusters which may represent roles:

- The cluster aggregating the *rAcc* and *wAcc* permissions has the same members as its parent, so its only information value is the set of rights it contributes to its parent. We consider it to be an intermediate cluster and remove it from the hierarchy, by redirecting its children to its parent. The hierarchy tree (or forest, in general) is then no longer binary.

- The cluster aggregating the *cdAcc* and *rTrans* permissions has three members, just as the respective trivial clusters. This proves that both permissions are always granted together, a good indicator for them representing a role. From the semantics of the aggregated permissions, a banking expert could derive that this is some kind of *JuniorManager* role.

- The cluster for the *rAcc*, *wAcc*, and *cTrans* permissions has four members, which are identical with the members of the trivial clusters for the *cTrans* and *wAcc* permissions; the *rAcc* cluster has two more members. The aggregated permissions suggest that this cluster represents the role *Clerk*. The two extra members of the *rAcc* cluster hint at either inconsistencies in the permission assignments (see Section 3.1), or an alternative clustering (see Section 3.2).

- The root of the cluster hierarchy combines all permissions and has only one member. These are both unlikely properties for a role and may for example indicate an accidental combination of permissions.

The analysis has produced two roles that are not in hierarchical order. If we accepted the root cluster as a role, it would be a sub-role (see Section 3.1) of the *Clerk* and *JuniorManager* since it has more responsibilities.

Note that the tasks introduced in the beginning are not needed to interpret the clustering result. We only used them to motivate the permission assignment, and the expert may use them to guide her judgement about the validity and semantics of clusters.

## 3.1 The cluster hierarchy

We have shown in Section 3 that along with the clusters, the algorithm automatically constructs a partial order for them: A cluster subsumes all permissions of its sub-clusters, and the resulting hierarchy is therefore called a set hierarchy or subsumption hierarchy. This hierarchy may correspond to a hierarchy of roles.

The basic idea here is that a sub-role has more responsibilities or tasks than its super-roles, and additional permissions are necessary to perform them [21]; with this definition of a role hierarchy, a role inherits all the permissions of all its super-roles. Therefore, the cluster corresponding to the sub-role has to include those permissions and will consequently be formed later than the cluster representing the super-role.

Note that the clusters and their corresponding roles are in reverse hierarchical order—i. e., the higher up in the cluster hierarchy, the lower in the role hierarchy, respectively. Also note that the cluster hierarchy may form a forest instead of a single tree.

Although it seems that the cluster hierarchy and the role hierarchy should match very well, there a two problems. One is accumulation (gathering permissions by promotion), and the second is the definition of the role hierarchy itself. We discuss both problems in the following paragraphs.

*Accumulation.* Persons in enterprises tend to accumulate permissions as they move within the enterprise, working in new departments and roles. Often, permissions are not revoked, once they have been granted. Over the long term, a single person may act in many roles sequentially but not at the same time, but still retain all the permissionsThis accumulation of roles and permissions over time shows up in the cluster hierarchy as chains of sub-/super-roles not unlike those originating from a role hierarchy.

In the example cluster hierarchy in Figure 1, the root cluster may be interpreted as the result of accumulation. Its only member may have started as a *Clerk* and later been promoted to *JuniorManager* without giving up the permissions of a *Clerk*. To explain why the root cluster cannot

correspond to a role, a banking domain expert might reason that it is very unlikely for a bank employee to be both a *Clerk* and the *JuniorManager* that may approve the *Clerk*'s transfers.

In general, both the role hierarchy and the accumulation hierarchy are unrelated. They correspond only if people often get promoted along the role hierarchy; if cross-promotion prevails, the hierarchies may differ significantly, making it difficult to extract the role hierarchy from the cluster hierarchy.

*Alternative Role Hierarchy Definitions.* Up to now, we have assumed a specific kind of role hierarchy, namely one where sub-roles inherit all permissions of all their super-roles; this is called a generalisation hierarchy in [13]. There may be other definitions of a role hierarchy where sub-roles have incomparable sets of responsibilities. These hierarchies cannot be reconstructed using hierarchical data mining algorithms based on permissions alone but need additional knowledge, for example the organisational structure of the enterprise in question or user attributes. Of course, data mining techniques can be applied to those data sets as well, in order to construct hierarchies.

[13] specifically discusses the problems of inheritance of permissions in the light of enterprise control principles like separation of duties, delegation, supervision and review, and in [14], alternative role hierarchies and their consequences for access control are presented. We consider the arguments of [14] in Section 5. In general, one should contemplate whether hierarchies for managing permissions and hierarchies for organising responsibilities should coincide. The focus of ORCA is primarily on the efficient analysis of permissions.

## 3.2 The matching criterion

In Step 3 of the algorithm, clusters are formed from permissions that share the maximal number of members. The maximality condition ensures that each cluster gets at most one super-clusterand results in a tree with single permissions as leafs and sets of permissions as inner nodes. Consequently, single permissions show up in exactly one path of clusters from the leaf to the root of the tree in which it is enclosed. The permission is therefore treated as if it is used exclusively in one role and all its super-roles.

This assumption may not be the best choice from a semantical point of view: Permissions are seldom used by one role only and may be necessary for incomparable roles. Translated into the cluster hierarchy, this means that corresponding clusters would have to have permissions in common. However, the chosen algorithm does not yet support this.

*Relaxing Exclusiveness.* It is possible to let the algorithm define clusters for the $n$ best combinations of permissions, instead of for the optimal match alone, or for all combinations down to a limit of shared users.Then, permissions would not be locked exclusively into one cluster (and its ancestors) but may show up in several clusters.

This has consequences for the construction of the cluster hierarchy, namely the improved algorithm will deliver a directed acyclic graph (DAG) rather than a tree. Since there is no single hierarchy in a DAG, it is more challenging to transform the cluster hierarchy into a role hierarchy.

On the other hand, the cluster hierarchy will be more stable with respect to noise in the data. With exclusiveness, few accidentally granted permissions can influence large parts of the cluster hierarchy and lead to an erroneous clustering.

For example, if the difference between the optimal and the second best match for a cluster is very small, then the clustering might be wrong due to minor inconsistencies in the permission assignment. In the cluster hierarchy in Figure 1, the *rAcc* permission shows up only in the upper part of the hierarchy. However, we know that it also occurs almost as often with the *rTrans* and *cdAcc* permissions aggregated in the lower part. One person makes the difference here, and we have already argued in Section 3.1 that this assignment might be due to accumulation, meaning that it should be revoked anyway.

With the relaxed clustering criterion, alternative clusters will be constructed, too, thus placing the burden of choice on the analyst instead of letting the algorithm make a narrow decision which might be based on dirty data.

*Avoiding Coincidence.* The algorithm has yet no means to check the semantics of permissions and may build clusters that have no semantic counterpart in roles, e. g., due to a person combining several roles, or accumulation (see Section 3.1). Since it requires expert knowledge to identify these invalid clusters, the algorithm has been adapted to respect these judgements in its matching criterion. To this end, clusters can interactively be marked as invalid, which will put the corresponding combination of permissions into a list. The algorithm ignores all permission combinations from the list. This list of invalid combinations of permissions is stored persistently with the cluster data and over time accumulates the expert knowledge.

# 4. THE ORCA TOOL

ORCA is a Java-based tool intended as an instrument to visualise the hierarchy of existing permissions and to support the transformation of the cluster hierarchy into an enterprise role concept. ORCA is based on the clustering algorithm described in Section 3. It displays the result of the clustering in a so-called cluster hierarchy view and supports the analysis of this hierarchy by highlighting clusters according to different criteria. Besides the permission assignment matrix, ORCA can use detail data about persons and organisational units of the enterprise. This data is shown on demand for the members of any cluster.

## 4.1 Visualising the cluster hierarchy

The main view of ORCA is the cluster hierarchy view which shows the result of the cluster algorithm of Section 3 giving the user an overview over the existing permission assignments. Figure 2 shows an excerpt from the graphical overview provided by ORCA[1].

*The nodes.* Each node in the graph represents a cluster as a rectangle with the number of members $|members(c)|$ in the upper half and, for inner nodes, the number of aggregated rights $|rights(c)|$ in the lower half. For leaf nodes, the

[1]The data shown in the examples comes from a real company and covers about 4 800 user accounts and more than 1 600 permissions. All names have been anonymised for the presentation, leaving the structures intact.

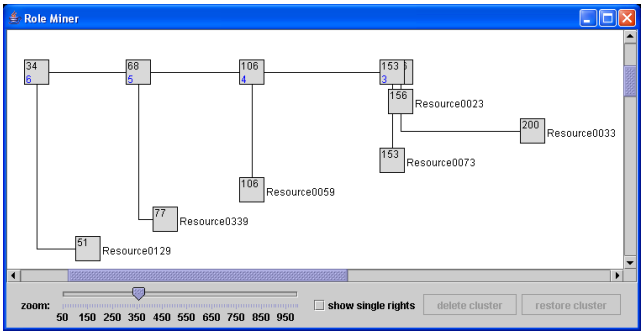


**Figure 2: The cluster hierarchy view in ORCA**

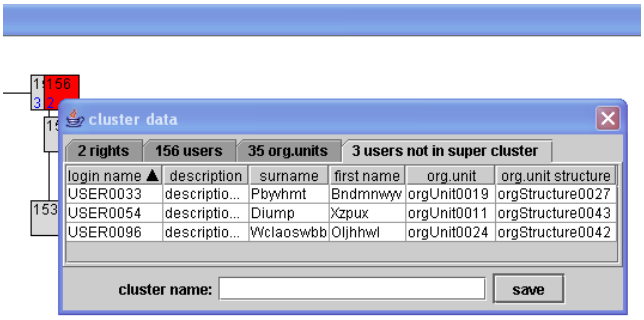


**Figure 3: The cluster information dialog**

number of rights is always 1, so it is not shown in the node; instead, the access right itself is shown next to the node. In the example in Figure 2, the root node of the tree combines 6 permissions which are held by a total of 34 users. The leaf nodes hold permissions named 'Resource$X$'.

Sub-clusters and super-clusters are connected by lines, where the super-cluster is shown left of the sub-cluster. The cluster hierarchy thus resembles a tree or forest, with root clusters on the left and trivial clusters as leaf nodes stretching out to the right. While the algorithm in Section 3 constructs a binary tree, ORCA removes all intermediate nodes (cf. Section 3) so the tree is no longer binary.

Besides normal cluster trees, the cluster hierarchy may also hold degenerated trees consisting of single nodes, which appear as unconnected leaf nodes. These are shown on demand below the other trees.

*The co-ordinate system.* The position of a cluster node along the x-axis of the graph corresponds directly with the number of members of the node. The further to the right a cluster appears the more persons share the permissions represented by it. Consequently, clusters at about the same vertical line have about the same number of members, and if they are connected, they even share most of those members.

In Figure 2 for example, the cluster 'Resource0023' and the cluster 'Resource0073' are placed close to each other because their memberships differ only by 3 persons. Since their closest common super-cluster has the same x-position as 'Resource0073', both have all members of that cluster in common.

Since super-clusters aggregate all permissions of their sub-clusters, nodes on the left side of the graph represent larger sets of access rights and therefore smaller groups of persons sharing them. However, there is no direct relationship between number of members and number of permissions; even leaf nodes may have a very small set of members if they represent very special permissions.

Along the y-axis, the trees are sorted in descending order of the number of members of their root clusters and shown one below the other.

## 4.2 Cluster detail information

To adapt existing permission assignments or to make decisions with respect to a role definition, details about the clustered rights and the associated users are needed. ORCA offers this information in cluster detail dialogs which are non-modal and can thus stay open for as many clusters as needed. This enables the detailed comparison of clusters.

Figure 3 shows the cluster detail dialog with its four tabs.

- With the input line at the bottom of the dialog, a cluster may be given a name. This name is then shown as a tool tip in the cluster hierarchy view (see Section 4.1). The name should be chosen to reflect the semantics of the cluster which is most likely the role that it corresponds to. Thus, the ORCA user can use cluster naming to identify those clusters that are part of the role hierarchy under development.

- The first tab shows the list of the access rights that are aggregated in the cluster.

- The second tab holds the list of members with their personal details like login name, full name, description, and the organisational unit to which the user belongs.

- The third tab shows all the organisational units of cluster members, together with the number of cluster members belonging to that unit. This information gives an idea of the degree of congruence between the cluster hierarchy (and possibly the role hierarchy) with the organisational hierarchy.

- The fourth tab is only shown for non-root clusters. Here, ORCA lists all those members of the cluster that are not members of its immediate super-cluster

  As shown in Figure 2, the clusters 'Resource0023' and 'Resource0073' share 153 members with their closest common super-cluster. Since cluster 'Resource0023' has 156 members, there are 3 persons (shown in Figure 3) that do not have access to 'Resource0073'.

Each tab shows the number of list elements in its name for quick reference. All lists can be sorted in both directions and incrementally by arbitrary columns. The person lists can be customized by hiding arbitrary columns.

All in all, the cluster information can help the ORCA user to understand how a clustering has been accomplished and form a basis for further decisions with respect to the role concept and the management of access rights.

## 4.3 Highlighting clusters

In a cluster hierarchy, there can be certain patterns which can either be a clue for how to form organisational roles or reveal weaknesses in the current permission assignments. ORCA offers ways to help users spot such patterns in the cluster hierarchy.

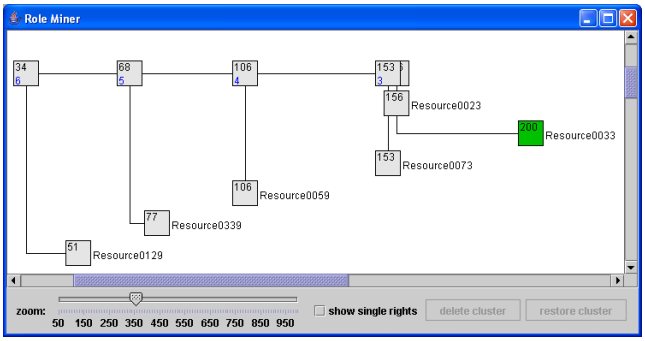


**Figure 4: Clusters marked by organisational unit**

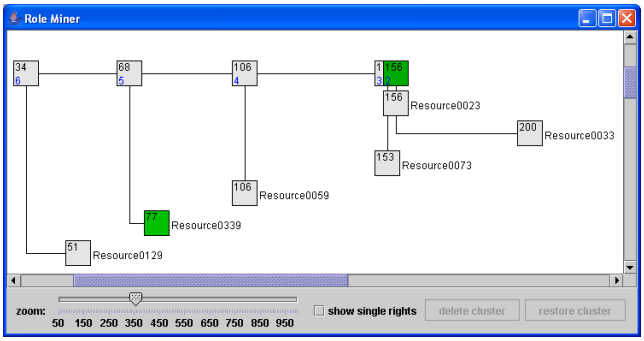


**Figure 5: Clusters marked by member difference**

Man is the proverbial visual animal and recognizes patterns best when they show up visually. To support the discovery of patterns in the cluster hierarchy, ORCA therefore does not have a search function that sequentially directs the user's attention to single clusters but rather offers marking algorithms that take a criterion and mark all clusters that match the criterion, by changing their colour. The colour of a node is gradually intensified according to the degree of conformance to the given criterion. This way, potential patterns show up in the cluster hierarchy view and provide entry points for closer examination.

Up to date, ORCA implements two marking algorithms: one for marking organisational unit quotas in clusters and the other marking deviations in memberships between clusters. Both are discussed in the following paragraphs.

*Marking organisational unit quotas.* To get an overview over the involvement of organisational units in the clusters of a hierarchy, ORCA offers the organisation unit quota marking algorithm. It lets the user select some organisational units from a list and then marks clusters containing persons belonging to one of these units in linear relation to the share of these members. The higher the percentage of members belonging to one of the selected units, the more intense the node colour. With this marking algorithm, users can easily find all clusters with members from certain departments, or check the correlation between the cluster hierarchy and the organisational structure of the enterprise.

Figure 4 shows an example where the marking algorithm has been employed to find members of the 'orgUnit0063' department. Since the leaf node 'Resource0033' has been marked with intense colour, we know that a large share of the employees with permission 'Resource0033' belongs to this unit. The figure also reveals that no other cluster in the depicted tree has members from that department. This pattern makes the marked cluster worth analysing to find the reason for this exceptional state of cluster 'Resource0033' with respect to department 'orgUnit0063'.

A single node marked in a whole cluster tree is only one example of patterns that might emerge when marking clusters according to their organisational unit share. There may be other patterns that might warrant closer examination of a cluster with respect to organisational units, too.

*Marking membership difference.* In Section 4.2, we have already motivated the significance of small deviations in the membership of neighbouring clusters: If only few persons differ in their permission assignments from a larger group, this may be due to inconsistencies in the assignments. It is however too time-consuming to find critical clusters by checking the fourth tab of their cluster info dialog. To get a quick overview over all clusters concerned ORCA offers the membership difference marking algorithm. This algorithm marks clusters according to their membership distance to their direct super-cluster[2]. Here, membership distance is defined to be the cardinality of the set of persons that are members of the sub-cluster but not the super-cluster—i. e. $|members(c) \setminus members(d)|$ for a cluster $c$ and its immediate super-cluster $d$. If this distance is below an user-adjustable upper limit, the sub-cluster will be highlighted, and the smaller the distance, the more intense the node colour.

The upper limit on membership difference to indicate up to which amount the deviation between two adjacent clusters might be questionable must be chosen by the user. It depends on the size of the organisation as well as on the granularity of the permissions. Deviations larger than the limit are considered too substantial to be caused by mistakes and therefore indicate real differences between clusters.

The lower limit shows a difference in the analysis of membership deviation for role concept design on the one hand and for permission assignment inconsistencies on the other hand. For role concept design, clusters with no membership difference are interesting and should be marked. In contrast, in permission assignment analysis a membership difference of 0 does not indicate any problem but perfectly consistent assignments. Thus, the respective cluster should not be marked, and the lower limit for the algorithm should be 1. ORCA offers both alternatives and leaves it to user to select which lower limit to use.

In Figure 5 we see the result of applying the difference algorithm with an upper limit of 10 and a lower limit of 1. The cluster 'Resource0077' is marked lightly because the difference of 9 between its 77 members and the 68 members in the direct super-cluster is slightly under the limit. The cluster representing the combination of 'Resource0023' and 'Resource0033' is marked more intensely because its membership of 156 persons overlaps much more with that of its super-cluster which has 153 members.

The interpretation of small membership differences among clusters depends on the focus of the user. In role concept design, clusters differing only by a few persons might repre-

[2]If we relax the matching criterion as described in Section 3.2 we'll have to use the minimal distance since there may then be more than one direct super-cluster.

sent rather one role than two. In the analysis of permission assignments, a low deviation might indicate that persons who are only contained in the sub-cluster are accidentally deprived of the access rights added by the super-cluster.

# 5. RELATED WORK

The growing interest in Role-Based Access Control as a means of efficient and secure permissions management in larger enterprises has stipulated work on role engineering [5, 17]. Some publications discuss various approaches for the role definition process, while others concentrate on the qualities of role hierarchies. We now look at related work for each aspect in turn.

*Data Mining Techniques in Role Engineering.* The traditional top-down approach to role engineering [18, 6, 16] starts from process or scenario descriptions and extracts roles, or starts from initial 'job functions' defined by experts and refines those into roles. The roles are then decomposed into suitable sets of permissions. Since permissions enter late into this process, they are not analysed to help the role definition process.

The top-down approaches have been complemented with bottom-up approaches which start from the actual permission assignments and/or user details, and aggregate them into roles [24, 6]. Only few publications discuss the use of data mining algorithms to support this aggregation process, however there are two products, namely Eurekify Sage and betasystems' SAM Role Miner.

To our knowledge, the term 'role mining' has been coined by [24]. The author is closely related to the SAM Role Miner product. [12] presents some details about the algorithm used by SAM Role Miner. It starts with a set of sample users given by an expert and iteratively lets the user apply two techniques called Association, to define rules, and Demographic Clustering, to define a given number of clusters; the latter sounds like k-means clustering. Applying demographic clustering repeatedly allows then to build up a role hierarchy but the authors list this as an open question which they intend to solve using association. In contrast, ORCAs algorithm does not require pre-defined samples or a given number of clusters, and automatically delivers a role hierarchy. It therefore seems better suited to the job at hand. Furthermore, SAM Role Miner uses clustering only to find so-called organisational roles, as opposed to functional roles for which it uses association; these functional roles do not cross system boundaries. This seems an unfortunate decision since the combination of data from several systems should enhance the clustering results. Parameterisation of roles is an open question for SAM Role Miner as well as for ORCA.

There are no scientific publications about Eurekify Sage but the founder of Eurekify, Dr. R. Rymon, has published papers about Set Enumeration [19] which is one technique used by Sage. According to white papers [20, 25], Sage finds role hierarchies just like ORCA does but uses its own, patented algorithms. However, a fundamental difference is the target of clustering: Sage builds clusters of users based on their access rights, while ORCA build clusters of access rights based on the users having those rights. Since we believe (combinations of) permissions to be more promising indicators for roles than groups of persons, we expect ORCA to perform better than Sage in defining role hierarchies.

*Role hierarchies.* [13, 14] discuss the influence of different types of role hierarchies on access control, specifically the use of inheritance in those hierarchies. They argue that enterprise control principles favor role hierarchies where inheritance of permissions might be dangerous. However, inheritance of permissions is automatic iff users may have more than one role, as is usually the case: A user has all permissions of all his or her roles, and if two of the roles are related hierarchically, the effect is that of permission inheritance. So, rather than blaming permission inheritance, one could question role hierarchies based on the subset relationship of members, that is where members of sub-roles are also automatically members of super-roles.

# 6. CONCLUSION AND FUTURE WORK

Role Mining in combination with the bottom-up approach to role concept definition as required for example for RBAC can be a viable alternative to the top-down approach of role engineering. It can be implemented by applying data mining algorithms to permission assignments within an enterprise. In this paper, we have described one such clustering algorithm in detail, together with ORCA as a prototypical implementation of a tool for the visualisation and analysis of permission cluster hierarchies.

Right now, we are about to evaluate ORCA in an enterprise with about 3 500 employees, more than 4 800 user accounts and more than 1 600 permissions. The focus here is on the visualisation of permission structures, but a subsequent evaluation regarding its support for role definition is intended.

*Possible improvements.* Beyond an evaluation of the current ORCA prototype, we are going to investigate some possible improvements.

- We want to examine other data mining algorithms with respect to their applicability for role mining. One drawback of the current algorithm is its strict partitioning of the permissions set (see Section 3.2): Once a permission is part of a tree in the forest of permission hierarchies, it at cannot be clustered elsewhere again. This runs contrary to the fact that one permission may very well be connected to a number of different roles within a role concept. Alternative, non-partitioning algorithms which allow the repeated appearance of permissions in different clusters in the first place may remedy this problem.

- From the angle of usability, rules are better suited to support users in identifying invalid permission combinations than manual tagging. Instead of having users mark the invalid clusters one by one, these clusters could also be described by rules evaluated by ORCA for automatically excluding invalid combinations from the clustering.

As can be seen, there are still several opportunities to improve both the ORCA tool as well as the algorithm to better support role mining.

# 7. REFERENCES

[1] ACM. *Proceedings of the 3rd ACM Workshop on Role-Based Access Control (RBAC 1998)*. ACM Press, 1998.

[2] ACM. *Proceedings of the 4th ACM workshop on Role-Based Access Control (RBAC 1999)*. ACM Press, 1999.

[3] ACM. *Proceedings of the 5th ACM workshop on Role-Based Access Control (RBAC 2000)*. ACM Press, 2000.

[4] *Information Technology – Role Based Access Control*. Number ANSI/INCITS 359-2004. InterNational Committee for Information Technology Standards, 2004.

[5] E. J. Coyne. Role engineering. In *RBAC'95: Proceedings of the 1st ACM Workshop on Role-Based Access Control*, page 4. ACM Press, 1996.

[6] P. Epstein and R. S. Sandhu. Engineering of role/permission assignments. In *17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 127–136. IEEE Computer Society, Dec. 2001.

[7] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[8] U. Grimmer and H. Hinrichs. A methodological approach to data quality management supported by data mining. In E. M. Pierce and R. Katz-Haas, editors, *6th Conference on Information Quality (IQ 2001)*, pages 217–232. MIT, 2001.

[9] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[10] J. Joshi, E. Bertino, and A. Ghafoor. Hybrid role hierarchy for generalized temporal role based access control model. In *26th International Computer Software and Applications Conference (COMPSAC 2002)*, pages 951–956. IEEE Computer Society, Aug. 2002.

[11] A. Kern. Advanced features for enterprise-wide role-based access control. In *18th Annual Computer Security Applications Conference (ACSAC 2002)*, pages 333–342. IEEE Computer Society, Dec. 2002.

[12] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *SACMAT 2003: Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, pages 179–186. ACM Press, 2003.

[13] J. D. Moffett. Control principles and role hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control (RBAC 1998)* [1], pages 63–69.

[14] J. D. Moffett and E. Lupu. The uses of role hierarchies in access control. In *Proceedings of the 4th ACM workshop on Role-Based Access Control (RBAC 1999)* [2], pages 153–160.

[15] M. J. Moyer and M. Ahamad. Generalized role-based access control. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, pages 391–398. IEEE Computer Society, Apr. 2001.

[16] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *SACMAT 2002: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 33–42. ACM Press, 2002.

[17] H. Roeckle. Role-finding/role-engineering (panel session). In *Proceedings of the 5th ACM workshop on Role-Based Access Control (RBAC 2000)* [3], page 68.

[18] H. Roeckle, G. Schimpf, and R. Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In *Proceedings of the 5th ACM workshop on Role-Based Access Control (RBAC 2000)* [3], pages 103–110.

[19] R. Rymon. SE-trees outperform decision trees in noisy domains. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 331–334. AAAI Press, 1996.

[20] R. Rymon. Sage – Enabling Role-Based User Management. Presentation slides, Eurekify, Dec. 2002.

[21] R. S. Sandhu. Role activation hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control (RBAC 1998)* [1], pages 33–40.

[22] R. S. Sandhu, V. Bhamidipati, E. J. Coyne, S. Canta, and C. E. Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of the 2nd Workshop on Role-Based Access Control (RBAC 1997)*, pages 41–54, 1997.

[23] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[24] G. Schimpf. Role-engineering: Critical success factors for enterprise security administration. Position paper for [17], Dec. 2000.

[25] M. Sel. RBAC & Role Mining. Technical report, COSIC, 2004.