ENGINEERING OF ROLE/PERMISSION ASSIGNMENTS

Pete Epstein
George Mason University Student
Manassas, VA 20111
<mepstein@tidalwave.net>

Ravi Sandhu
Department of ISE
George Mason University, Fairfax, VA 22030
<sandhu@gmu.edu>

Abstract

In this paper, we develop a model for engineering role-permission assignment. Our model builds upon the well-known RBAC96 model [SCFY96]. Assigning permissions to roles is considered too complex an activity to accomplish directly. Instead we advocate breaking down this process into a number of steps. We specifically introduce the concept of Jobs, Work-patterns, and Tasks to facilitate role-permission assignment into a series of smaller steps. We describe methodologies for using this model in two different ways. In a top-down approach, roles are decomposed into permissions, whereas in a bottom-up approach, permissions are aggregated into roles.

1. Introduction

With industry's increased awareness to protect the confidentiality and integrity of applications and its data, system administrators' are continuing to implement access control mechanisms. Historically, user access has been granted by adding the necessary permissions to each individual application. Administering accesses to many users for several different applications quickly becomes tedious and error prone; this is particularly true when the user changes positions and requires a different set of accesses.

An alternative is not to directly assign users to permissions for each application; instead, users are assigned to roles and the roles are mapped to permissions for each application. If the user's needs change, the administrator simply assigns another role containing appropriate permissions, rather than updating the authorization on each user application.

The well-known RBAC96 model's [SCFY96] Permission Assignment provides the efficiency of allowing the administrator to assign users to roles rather than directly to permissions. The RBAC96 model directly assigns a role to permissions. Without knowing the details of the roles, we cannot simply say that an arbitrary role, such as a doctor, requires access to the patient's medical records, patient's x-rays, and the research database. Assignment of permissions to roles can itself be a complex undertaking.

We need an approach to assist us in determining a role's permission. One solution is to further define the details of a role by studying the work that is being conducted by that role. If we consider that an individual agent performing a role is required to perform more than one job (i.e., multiple responsibilities required by one role), we can define the tasks that the role must follow to complete the desired job. If an access to an object is required, then we must assign the necessary permissions to the task so we can complete the job. After identifying all of the tasks and the required permissions, we can collect all the permissions and assign them to a role. This is a collection of all the permissions that are required for the individual agent to perform the responsibilities of the role.

Using this approach, we can perform the role/permission assignment by either considering the top-down approach of decomposing roles to permissions or the bottom-up approach of aggregating permissions to roles.

There has been related work. Chandramouli [C99] discusses an approach for identifying roles in a healthcare information system. Thomsen [TOP99] presents a layered methodology called Napolean. Roeckle [RSW00] describes their experience in role-permission engineering in a large corporate environment.

The motivation for this paper is to define a methodology to decompose the functionality of the roles and to logically assign their components to permissions without ignoring any required accesses. We define a model that contains the layers that will assist us in designing decomposition of roles to permissions or aggregation of permissions to roles. In addition, we define properties to optimize the assignment of roles to permissions. As presented in this paper a single model to decompose roles to permission, aggregate

permissions to roles and optimization properties enhances the previous role engineering work.

This paper will discuss the decomposition and aggregation of roles and permissions by discussing the extension of the RBAC96 model, the decomposition and aggregation approaches, the tools that are needed to perform the approaches, and an example showing the use of the decomposition approach.

2. RBAC96 Model Review

The model developed in this paper is constructed by extending the RBAC96 model [SCFY96]. The RBAC96 model is a comprised of four models: $RBAC_0$, $RBAC_1$, $RBAC_2$, and $RBAC_3$. $RBAC_0$ is the base model. $RBAC_1$ and $RBAC_2$ added role hierarchies and constraints, respectively. $RBAC_3$ is the consolidated model. RBAC96 also makes a distinction between User and Administrative roles.

There are three components of RBAC96 that we are interested in using for the extension of the model: users (U), roles (R), and permissions (P).

The model defines the components of role/permission assignment by PA. It also defines the role hierarchy RH. It does not, however, state how to engineer the role/permission assignments (See Figure 1).

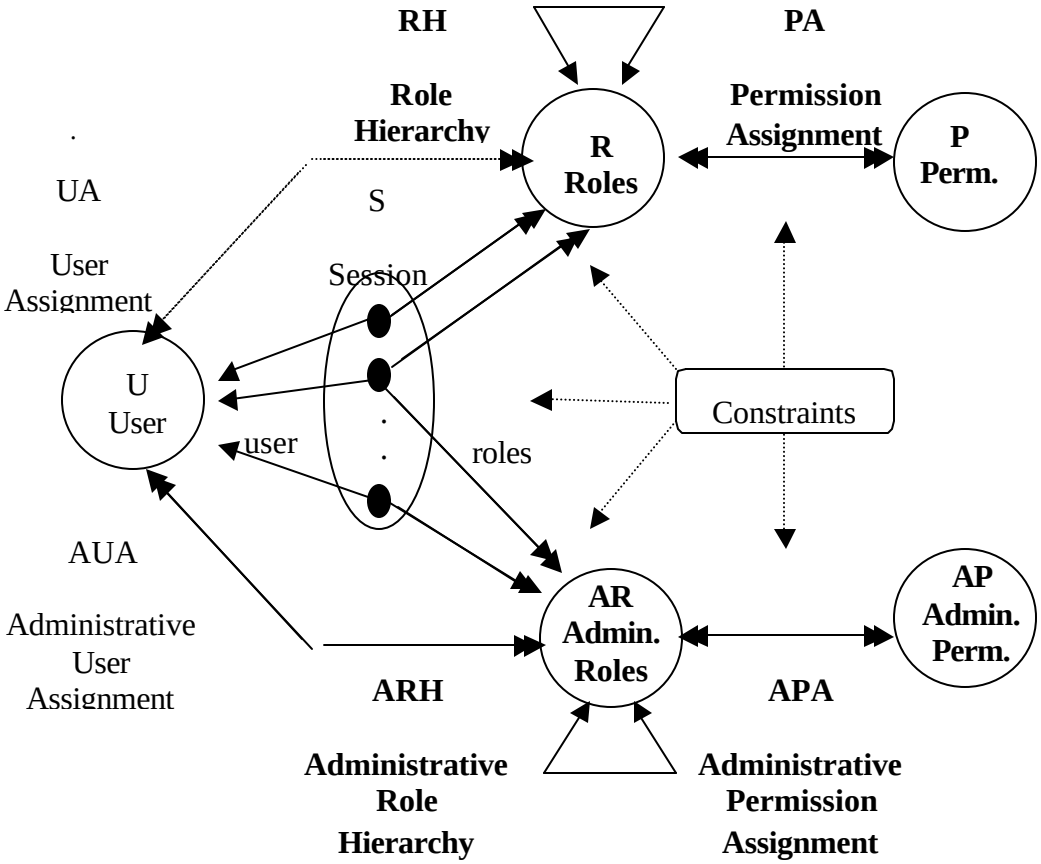

Figure 1: RBAC96 Model

Specifically, we will be studying PA, P, and R. This subset of the model is displayed in Figure 2.

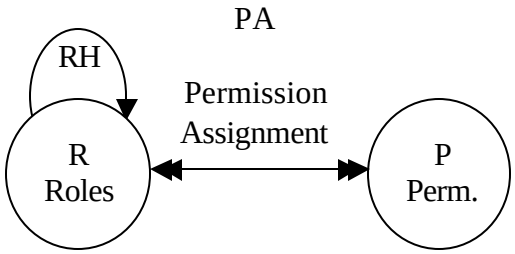

Figure 2: RBAC96 Model's Permission/Role Assignment

3. RBAC Extension

We extended the RBAC96 model by including three additional layers between the roles and permissions.

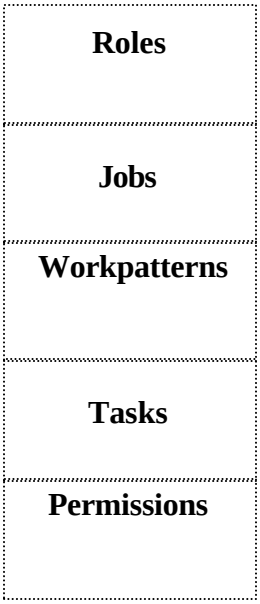

Figure 3: Role/Permission Assignment Model

As shown in Figure 3, we add three additional sets: Jobs, Workpatterns, and Tasks.

- J is a set of Jobs;
- T is a set of Tasks; and
- W is a set of Workpatterns, $W \subseteq 2^T$.

We consider that a role may perform more than one type of work. A role is responsible for all the activities that are required to perform the work. We define each type of work as a job. The jobs need not be in any sequence; but for organizational purpose, we group activities into a set we call a workpattern. Each workpattern is composed of a set of steps that are required by a single agent to complete the work of the job. These workpatterns can be part of a workflow that is being completed by more than one distinct agent. Each step of the workpattern is assigned to a task. Later, we show that the tasks requiring access to applications will be mapped to the permissions granted the desired access.

An example of the decomposition of roles can be a professor role that will perform the jobs of teaching and researching. For simplicity, the teaching role has a workpattern that requires the steps to creating a presentation, creating exams, recording results, and e-mailing results; whereas the researching job has the workpattern of creating a theory, testing the theory, documenting the results, and e-mailing results. These steps are assigned to a task. The tasks are "presentation, exam, record, e-mail, theorize, test, and document." We do not need to list the second e-mail task for research because we can "re-use" the e-mail task identified by the teaching job. The tasks are mapped to permissions that grant access to perform the work required by each task.

In Figure 4, we show the relationships between each set. The double-headed line means "many." A single headed line represents "one." If we have a double-headed arrow pointing to two separate sets, we will have a many-to-many relation. In the case of jobs-to-workpatterns, we have a double-headed arrow pointing to Jobs and a single-headed arrow pointing to workpatterns, so we have a many-to-one relation. Many Jobs can map to the same workpattern, but we can only have one workpattern map to a Job.

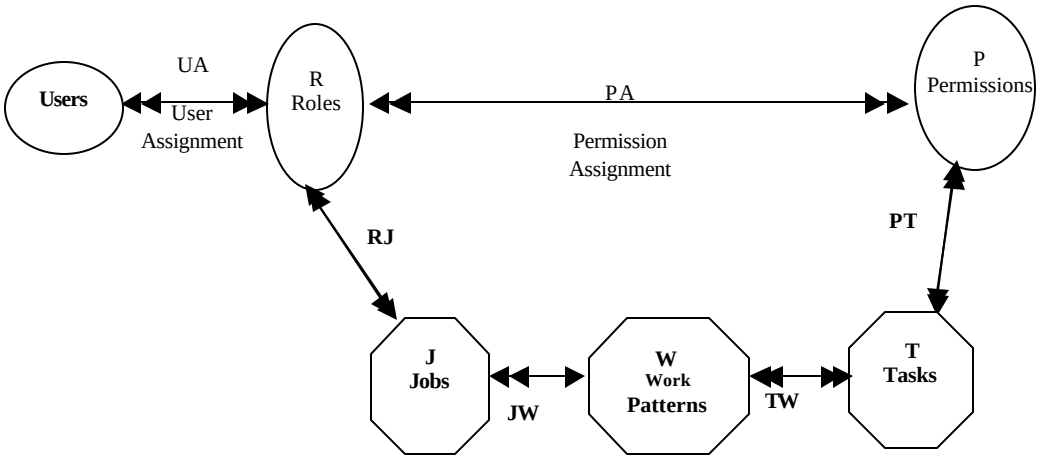

Figure 4: RBAC 96 Extended

4. The approaches

We have defined the layers and the relationship between the layers. The next step is to define the process of using these layers to engineer role-permission relationships. The complete description of this process is beyond the scope of this paper. Instead we will identify the major challenges for the approaches, decomposition and aggregation, and state how they can be overcome.

4.1 Decomposition Challenges

.

As seen from the layer mapping in Figure 5, decomposing roles to permissions can be complex. To aid us in this challenge we addressed the following issues:

1. Focus the decision based on a criteria;
2. Define the work required by the role;
3. Define the logical order of the work; and
4. Define properties to optimize this approach.

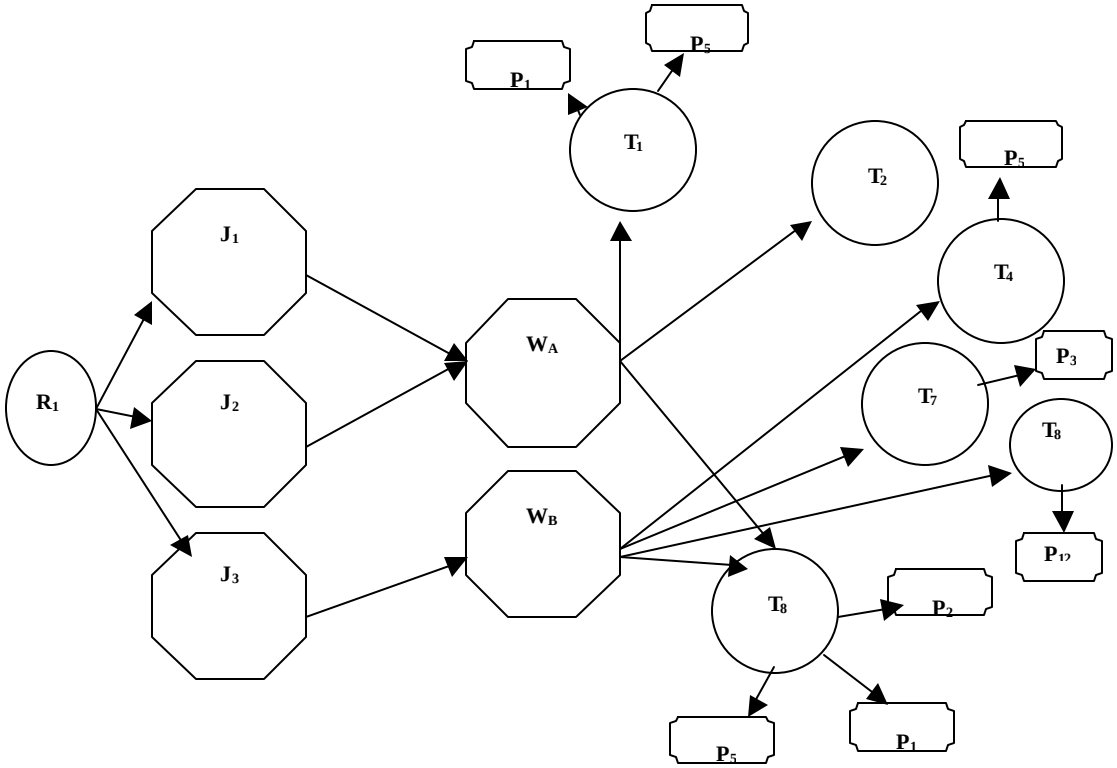

Figure 5: Role-Permission Mapping

4.2 Additional Aggregation Challenges

Aggregation requires the use of the four issues identified under the decomposition approach. In addition, the aggregation approach requires the grouping of a discrete set of permissions based on some type of organization into larger sets that will be eventually be assigned to roles. This additional challenge has been resolved by using the concept of "bucketing."

4.3 Focus Concept

Within each layer there can be more than one approach (i.e., engineering based on a design criterion). We consider the following approaches: 1) Role-focus, 2) Application-focus, and 3) Permission-focus.

Permission-focus requires decomposing the permissions based on attributes of the target permissions; application-focus uses the target application's attributes; and role-focus uses the attributes of the source roles. From these approaches, we use the attributes as the criteria to decide how to decompose or aggregate the current values to the next layer.

The focus challenge is to determine the decomposing approach (e.g., role-focus, application-focus, or permission-focus) and the subsequent criterion that will be used to engineer the role to permission assignments. The following concept can be used to engineer decomposition as well as aggregation. For simplicity, we do not consider hybrid approaches.

For each approach, we need to list the criterion. These criteria are used to assist in the engineering of the layers. For example, we can use:

- Role Attributes based on: Skill sets, Educational Level, Location, Experience;
- Application Attributes based on: Functionality, Manageability, Interoperability; and
- Permission Attributes based on: Platform, Access type, Type of target Application, and Capability.

4.4 Defining Role's Responsibilities

Next, we identify the role's responsibilities. For our research, we categorize the roles into the following groups:

1. The role responsibilities that have been documented,
2. The role responsibilities that have not been documented but where the role has been defined, and,
3. Neither the role nor the responsibilities have been defined.

By using our knowledge of the chosen approach and its attributes for the first group, we will divide the role's responsibilities into categories. Related responsibilities that can be part of the same job set will be merged into "like job" categories, J_1, … J_n. For simplicity, jobs will be reused and each J_i will be unique. For example, the duties of the Office Administrator are: 1) Maintain the records for all Ph.D. students, 2) Maintain the Calendar for the Dean, and 3) Schedule meetings with the professor.

In the second group, each role exists but has not been documented. An extra step is required to glean the responsibilities from the undocumented roles. This is accomplished by monitoring and then by documenting the activities performed when a user has activated the role. From the documentation and our knowledge of the chosen approach and its attributes, we analyze the responsibilities required to perform the role's activities. Similar to the earlier option, related responsibilities that can be part of the same job set will be merged into like job categories, J_1, … J_n.

For example, the role may be a professor. The Computer Administrator has not had time to create her list of responsibilities; however, a role-engineer can follow the administrator while she performs her role. The approach is application-focused and the attributes are functionality and manageability. We know that the jobs are based on the attributes of application functionality and manageability. Through observation, we determine that the administrator performs archiving, software maintenance, and password access control of application servers. Subsequently, we map the Computer Administrator role to the jobs of: Application Server Archiving, Application Server Software Maintenance, and Application Server Password Access Control. (Note: A job that already exists does not have to be redefined but can be reused.)

The final group is for a role that has been identified, but has not yet been defined nor documented. The responsibility of the role is only conceptual and cannot be verified against

existing activities. We need to deduce the role's expected responsibilities by interviewing the designer of the organization. We then document each role's responsibility within the framework of the approach and it's associated values. Related responsibilities that can be part of the same job set will be merged into like job categories, J_1, … J_n. For example, if the government agency ABC.com has a new position of Chief Information Officer (CIO) that is required by a new government law, we need to provide the role with the needed access to perform the work of the role. Although the role exists in industry, it is a new role for this government agency. Thus, we need to determine from management what the responsibilities of the role are. The approach is role-focused and the attributes are skill sets and experience. After talking with the Chief Financial Officer, Chief of the Agency, and the Chief of Operations, we determine that the job skill sets are Program Oversight, Technical Management, and Budget Reviewer. Although the role has the skill set to understand technological information, it does not have the experience to perform in-depth technical reviews. Fortunately, the CIO can hire a person to perform the role that contains the job of in-depth technical reviewer.

4.5 Workpattern Order

We need to identify all of the steps that the workpattern requires to perform the work of each job. These steps do not have to be followed sequentially, but each step is required to define the work of the job. Ambiguity of jobs increases the difficulty of defining all of the steps; however, if we can identify the steps logical ordering as a process, we will have an engineering aid to reduce the complexity of step definition. We begin by categorizing the job in one of three groups:

1. The steps are part of a single process that is entirely defined within one workpattern. All of the steps can finish without waiting for another step outside of the workpattern to finish.
2. The steps are part of at least one process that is outside of the workpattern. At least one step must wait for another step that is outside of the workpattern to finish.
3. The steps cannot be defined as a process.

Group 1 is a set of steps that must be derived from the job. We know that the steps are formulated as a process, so that there will be some semblance of order. The order need not be sequential; but there is a series of steps that need to be performed to satisfy the work of the role. We define the process within the criteria of the focus approaches. For example if there is a role-focus approach for the role of Professor for the job of Teaching within the criteria of Educational Level and Skill Sets, we determine the process steps that are required to satisfy the work of the role are:

- Investigate Information,
- Prepare Lectures,
- Lecture,
- Prepare Exam,
- Administer Exam,
- Grade Exam, and
- Record Exam.

All of these steps are defined and controlled within a workpattern. For each step S_{ij}, where i is the process and j is the step number, a workpattern W in group 1 will contain a set of S_{ij}, where all i's are equal, and for all j's, S_{ij} is contained in W.

Within the second group, we need to identify the steps within the external process that the workpattern will satisfy. Ideally, the master process has been created and the steps have been defined. Thus, we determine the job that performs the work, and then include the steps as part of the workpattern. If the steps of the master process are not known, but we are aware that a job is part of the external process, then we need to define the steps. As with group 1, we know that the steps are formulated as a process, so there will be some semblance of order, but it does not have to be sequential, although there is a series of steps that will be done to satisfy the work of the role. We define the process within the guides of the focus approaches discussed earlier.

For example, if the job is for a mortgage collection-clearing house, we need to understand that the role is part of a larger process that includes other roles such as: the mortgagee (the person paying the mortgage) and the mortgagor (the company receiving the money). We determine from the information that we obtained when we defined the role that the steps are: Send out list of mortgagee (mortgagor), send out notice (clearing house), send out payment (mortgagee), post payment (clearing house), pay bank (clearing house), and send out notice of payment receipt (mortgagor).

All of these steps are defined and controlled within a workpattern. Thus, for each step S_{ij}, where i is the process and j is the step number, a workpattern W in group 1 will contain a set of S_{ij}, where all i's are not equal and for at least one j, S_{ij} is not contained in W.

The last suggested method is an ad-hoc set of steps that may not be related. We cannot use the aid of a process to logically define the steps. All that is known is that there is a job that has been created as part of the approaches defined earlier. We must deduce from the present information what steps are required by the workpattern. For example, we may glean from the documented role of an administrator that a set of responsibilities did not fit into another job. They were combined into a job of an office manager and require the steps: 1) update employee payroll, 2) add employees to the company gym, and 3) obtain parking permits.

Thus, for each step S_{ij}, where i is the process and j is the step number, a workpattern W in group 1 will contain a set of S_{ij}, where all i's may be equal, and for some j's, S_{ij} is contained in W.

4.6 Concept of Buckets

When we aggregate from one layer to the next, we need the ability to combine like elements into the same group. A bucket is a grouping of like elements into the same group. We group elements into a bucket according to like capabilities that are based on the focus attributes. To ensure completeness (see properties), each permission has to be a member of at least one bucket; however, each permission can be a member of more than one bucket. Large groups may need to be further categorized into smaller groups. A large bucket that will represent multiple groups must be subdivided into additional buckets. For example, all the data file updates are defined into one group. The group contains over 200 permissions. We further categorize the group by data type (e.g., database files, network configurations, word files,...). Now we have a more definitive set of buckets that group more specific-like elements.

4.7 Defining Properties

As part of the role-permission extension, there are properties that can be applied to the layer and the mapping between the layers. These properties are: Uniqueness, Equivalence, Minimization, Reuse, and Completeness. The formal definitions follow and use the following symbols, along with the previous definitions of permissions.

We strive to minimize the number of elements that will be used to perform the role/permissions assignments. Ideally, each element is unique and therefore, each set will not contain duplicate entries. We can determine if an element is unique if there is not another element that is equivalent to that element. Our real interest in equivalence is that, when we finish mapping the layers to permissions, we want to know if the layered elements map to the same set of permissions. If the elements are equivalent, then the element will grant the same accesses to the functions of the application and there may be no benefit to have more than one element that maps to the same set of permissions. To continue with this line of thought, we may not need to define another element if we can reuse an existing element that can provide the same set of accesses. Once we finish with the approach, we verify that all the pre-defined elements (i.e., roles and permissions) have been mapped. We check the completeness of the assignments by mapping each role to at least one permission and each permission to at least one role. As we stated earlier, we strive to minimize the number of elements; however, there may be a benefit not to eliminate duplicate unique elements. We will discuss these potential benefits later in this dissertation.

One of the goals of this dissertation is to detail an optimized approach to increase efficiency when performing the Role/Permission assignments. The key phrase is "an optimized approach." This means the optimization of each phase of the approach. Each phase has two major portions: the definitions of the elements within a layer, and the mapping of the elements from one layer to the next. Hence, we can increase efficiency by either reducing the number of elements within a layer, or reducing the number of mappings from one layer to the next.

We can reduce the number of elements by either eliminating duplicate elements or not defining new elements by reusing existing elements. To work towards these goals, we introduce properties that can be applied to the layer and the mapping between layers. These properties are: Uniqueness, Equivalence, Minimization, Reuse, and Completeness. They will be defined in greater detail below.

Uniqueness ensures that there are no two elements that contain the exact same values

within a layer. For example, the Information Technology and Psychology Departments require the same set of accesses to logon into the University registration applications. Both departments do not need to create their own version of a logon task; one unique task can be used for both departments. We work towards uniqueness when we: 1) eliminate or merge duplicates by minimizing elements or 2) reuse a unique element rather than define another element. We need to be careful that we do not eliminate an element that is needed for permission completeness. Permission completeness will be defined later in this paper.

Two sets, within the same layer, are equivalent if they contain the exact same elements. Permission equivalence is a special case of equivalence and is defined as two sets that permit access to the same applications but may not contain the same elements. Permission equivalent sets need not be identical, but equivalent sets are permission equivalent. For example, workpattern A may require three tasks: a task to logon to the computer, another to make a phone call, and a third to check e-mail. Workpattern B will perform the same tasks as accessing a computer and checking e-mail, and require a third task of faxing documents. The two tasks making a phone call and faxing documents do not require special permissions. Workpatterns A and B map to the same permissions, even though they contain slightly different tasks; consequently, workpatterns A and B are permission equivalent workpatterns.

The minimization property eliminates equivalent elements within a layer. Minimization can be performed on jobs, workpatterns, and tasks. As stated earlier, minimization is a goal but not a requirement**.** Equivalent elements can be merged into one element to eliminate the need to maintain multiple copies that contain the same accesses and information. In the previous paragraph, minimization was performed on the workpattern layer that contained equivalent workpatterns A & B by eliminating workpattern B.

Instead of inefficiently creating a new element every time the domain element maps to the same range value, we can reuse the range element in the target layer. The reuse property permits two elements from one layer to reuse the same element from an adjacent layer. Using the previous example the Information Technology Department has already defined its workpatterns to administer student records. When the Psychology Departments wants to create a workpattern, they find that a pre-existing task has been defined to access student records. Instead of creating a new task, they reuse the task that has been defined by the Information Technology Department. Reuse of elements occurs for workpatterns, tasks, permissions, and jobs, except for the aggregation of workpatterns to jobs. (Note: recall that the mapping from workpatterns to jobs is a many-to-one relation.)

A final term to define is completeness. This concept is important when we validate that all the elements of the domain are mapped to an element in the range. If an element is not mapped to the range, then it will not be part of the aggregation or decomposition approaches. In that case, either all permissions will not be assigned or the element is not required to assign all of the permissions to a role (e.g., a permission-free task or equivalent jobs). There is completeness of roles, jobs, workpatterns, and jobs; but they are all subservient to completeness of permissions. If a permission is not mapped to at least one role, then a portion of the application cannot be accessed; and, thus, it cannot be executed. For example, the human resource application has an access to backup its data; if that the backup access is not granted to a role, a backup can not be performed on the resource application. Analogously, if there exists a role that is not assigned to a permission, the role will not perform any work because it will not have any accesses to any applications.

In summary, there are equivalence, uniqueness, minimization, reuse, and completeness properties that apply to the permission-assignment model. Uniqueness, equivalence, and minimization apply to the elements within a layer; whereas, reuse and completeness apply to elements that are mapped between layers. Not all of the properties apply to each layer. Table 1 depicts the applicable property by an "X" in the relevant layer for the decomposition approach, and Table 2 for the aggregation approach. The completeness verification starts from the reverse direction; in the case of decomposition, the verification starts at permissions while for aggregation it begins at roles.

The only difference between the two tables is that the aggregation table does not show that a workpattern cannot reuse jobs. Recall that the workpattern to jobs is a many-to-one relation. If one workpattern can reuse more than one job, than there can be many workpatterns mapping to the same job, which is a violation of the initial definition of the job to workpattern mapping.

	Uniqueness	Equivalence		Minimization	Reuse	Completeness
			Permission			
Role	X					X
Job	X	X		X	X	X
Workpattern	X	X	X	X	X	X
Task	X	X	X	X	X	X
Permission	X				X	X

Table 1: Decomposition Table of Properties

	Uniqueness	Equivalence		Minimization	Reuse	Completeness
			Permission			
Role	X					X
Job	X	X		X		X
Workpattern	X	X	X	X	X	X
Task	X	X	X	X	X	X
Permission	X				X	X

Table 2: Aggregation Table of Properties

Let us consider the following design example: Mary, in the role of a doctor, is caring for her patient at the hospital. She needs to be able to perform the jobs: 1) Gathering information about her patients, 2) Operating medical equipment, 3) Researching nationally to diagnose ailments, and 4) Annotating the patient's hospital record. To perform the first job of gathering patient information, Mary needs to review hospital records, her own office records, the referring doctor's records, and the patient's long-term history.

The role "R" is a Doctor. The doctor role can perform four jobs: Job J_1 - Gathers information about her patients; Job J_2 - Operates medical equipment; Job J_3 – Researches nationally to diagnose ailments; and Job J_4 - Annotates the patient's hospital record.

For Job J_1, the Workpattern W_A is the following sequence of tasks: Task T_1 is to review hospital records; Task T_2 is to review the doctor's (Mary's) office records; Task T_3 is to refer doctor's records; and Task T_4 is to review the patient's long-term history.

Task T_1 requires a permission to review the hospitals database (Application A_1). Task T_2 requires a permission to review the doctor's office record (Application A_2). Task T_3 requires permissions to the three referring doctors' records (Applications A_3, A_4, and A_5); and Task T_4 requires a permission to review the patient's own record from the general practitioner (Application A_6). Task T_4 also requires two permissions: the doctor's and the patient's.

5. Summary

In this paper, we have introduced the layering of roles, jobs, workpatterns, tasks, and permissions to logically show an approach to decompose or aggregate roles and permissions. This led to the need for concepts that could be used to engineer the model's layer and define the relationship between each of these layers.

To strategically guide the role engineer in consistently defining the model, we presented a concept of "Focus." Focus provides information about a foundation component (i.e., roles, applications, or permissions) that we use to engineer the approach.

Another concept to aid in engineering is the ability to define the jobs of a role. We began by categorizing the roles into: Documented, Existing, or Undefined. We use a process flow to decompose the job into a set of steps. We realized that not all accesses that are required by a job might be part of a process, so we added an ad-hoc category for disjoint steps.

We also found that we needed a concept to aid in the aggregation of permissions. We combined aggregated permissions from one layer to the next layer by using buckets. Buckets were used to group permissions into tasks and tasks into workpatterns.

Finally, we considered the economy of re-using terms, efficiency of eliminating redundancy, and the ability to perform all necessary work. We found that we could enhance mapping of elements between layers. These properties were accomplished by:

1. Reusing previous work;
2. Minimizing the number of elements by determining if there was a need for uniqueness ; and
3. Performing all the necessary work to ensure that there is a complete mapping of elements between layers.

References

[C99] Ramaswamy Chandramouli (NIST), "A Framework for defining an Access Control Service for Healthcare Information System Using Roles", A Presentations for 4th ACM Workshop on Role-Based Access Control, Fairfax VA, October 28-29, 1999

[TOP99] D. Thomsen, R. O'Brien, and C. Payne, "Napoleon Network Application Policy Environment", In Proceedings of 4th ACM Workshop on Role-Based Access Control, pages 145-152, Fairfax VA, October 28-29, 1999

[HA99] Wei-Kuang Huang, Vijayalakshmi Alturi, "Secureflow: A Secure Web-enabled Workflow Management System", In Proceedings of 4th ACM Workshop on Role-Based Access Control, pages 83-94, Fairfax, VA October 28-29, 1999

[TOB98] Dan Thomsen, Dick O'Brien, Jessica Bogle. Role Based Access Control Framework for Network Enterprises, In Proceedings of 14th Annual Computer Security Application Conference, pages 50-58, Phoenix, AZ, December 7-11, 1998

[C95] Edward Coyne. Role Engineering, In Proceedings of First ACM Workshop on Role-Based Access Control, pages I-15 – I-16, Gaithersburg, MD, November 30-December 1, 1995.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, CharlesYouman, Role-Based Access Control Models, In IEEE Computer, Volume 29, Number 2, February 1996, pages 38-47.

[BRJ99] Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison Wesley Longman, Massachusetts, 1999

[S98] Ravi Sandhu, Role-Based Access Control, In Advances in Computers, Vol. 46, Academic Press , 1998.

[B95] John Barkley. Implementing Role-Based Access Control Using Object Technology, In Proceedings of First ACM Workshop on Role-Based Access Control, pages II-93 – II-98, Gaithersburg, MD, November 30-December 1, 1995.

[ES99] Pete Epstein, Ravi Sandhu, Towards a UML Based Approach to Role Engineering, In Proceedings of Fourth ACM Workshop on Role-Based Access Control, pages 145-152, October 28-29, 1999

[RSW00] Haio Roeckle, Gerhard Schimpf, Rupert Weidinger, Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organization, In Proceedings of Fifth ACM Workshop on Role-Based Access Control, pages 103-116, July 26-27, 2000.