Engineering of Role/Permission Assignments

by

Pete A. Epstein
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____	Ravi Sandhu, Director
_____	Edgar Sibley
_____	Richard Baum
_____	Stephen Nash
_____	Stephen Nash, Associate Dean for Graduate Studies and Research
_____	Lloyd J. Griffiths, Dean, School of Information Technology and Engineering

Date: _____ Spring Semester 2002
George Mason University
Fairfax, VA

Engineering of Role/Permission Assignments

A dissertation submitted in partial fulfillment of the requirements for the degree of Ph.D. at George Mason University

By

Pete A. Epstein
M.B.A
Marymount University, 1984
Bachelor of Science
University of Maryland, 1980

Director: Ravi Sandhu, Professor
Department of Information System

Spring Semester 2002
George Mason University
Fairfax, VA

Copyright 2002 Pete A. Epstein
All Rights Reserved

DEDICATION

In memory to my father, John Epstein, who provided me with the vision that I should continue to grow through education and that I should never stop growing.

In dedication to my beautiful wife, Mary Ann, who continued to support me and listen to my repeated discussion about my educational journey, even though she had edited my dissertation more times than there are stars in the sky.

To my children, Jennifer, Jeremy and Justin, please accept your grandfather's advice.

ACKNOWLEDGEMENTS

I wish to thank all of the educators of all my classes throughout public school, college, and universities that in one way or another had a part in teaching me the essentials to allow me to write this dissertation.

I wish to thank my dissertation director and committee chair, Prof. Ravi Sandhu for spending hundreds if not thousands of hours supporting me through the Ph.D. program. He allowed me to attend an advanced Computer Security class at GMU as my first class at George Mason University and my first class in computer security. After following Prof. Sandhu to San Diego, he accepted my request to be my committee chair and dissertation director. He allowed me to conduct research in an area of Information Technology (as long the subject area was related to RBAC).

I wish to thank Prof. Sibley for his forceful advice. He was always willing to allow me to drop by and talk with him about my progress. His advice was always a golden nugget that helped me learn. I will strive to learn as much as he has.

I wish to thank Prof. Baum. He has the spirit of a true educator. I asked Prof. Baum to be on my committee and brought him the committee form after his class on a raining and winding night. He told me I could have just mailed him the form. I told him out of respect, I felt I should have brought the paper to him in person. He told me the acceptance of my request was part of his job.

I wish to thank Prof. Nash for being on my committee even though I have never taken a class by him. After Prof. Nash became Dean he remained on my committee and supported my work.

I wish to thank Dave Roberts who continued to remind me of the importance of completing my work. If my progress was not sufficient, he made sure everyone else knew. I also wish to thank Mike Joyce for his excellent support and valuable advice.

I wish to thank ACSAC conference for allowing me to attend and present my work. I started my Ph.D. research at the San Diego conference by receiving a commitment by Prof. Sandhu and finalized the research by presenting my last paper in New Orleans. I also wish to thank ACM for publishing my first RBAC paper.

I wish to thank the wonderful GMU staff that supported my efforts, especially, Karen Alarie, Robert Vay, and Jonathan Goldman.

I wish to thank my company and my customer for supporting my efforts over the last several years, during school, conferences, and research.

I wish to thank Ed Amorso for his early advice that my Ph.D. was a journey and that my dissertation director was an exceptional adviser. His advice was precise.

I wish to thank all the people I have not named who assisted me by allowing me to work on my research and complete my journey. **What an educational journey it has been!**

TABLES OF CONTENTS

Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

ABSTRACT

ENGINEERING OF ROLE/PERMISSION ASSIGNMENTS

Pete A. Epstein, Ph.D.

George Mason University, 2002

Dissertation Director: Dr. Ravi Sandhu

No longer should any person be allowed to simply sit down and start working with a computer; only authorized personnel should be allowed to use the computer and its applications. Traditionally, an administrator would assign each person accesses to the applications. In assigning the accesses, the administrator would then grant all of the necessary permissions needed for the person to complete his/her work, while preventing that person from performing any unauthorized work.

Using an access model such as Discretionary Access Control (DAC), the permissions are granted to each individual user. Granting permissions to several users over many applications, DAC quickly became cumbersome, difficult, and costly to administer. An alternative access model that resolves these issues is Role Based Access Control (RBAC). RBAC is a proven technique to assign permissions to users via roles. A core aspect of RBAC is the Role/Permission Relation. Previous research has applied

RBAC models to create roles. In addition, application developers have accepted the definition of RBAC permissions; however, the research has not detailed a systematic model for determining the assignment of permission to roles. To evolve the RBAC model, and to align permissions with role responsibilities, an approach must be developed to ensure that all, and only those, permissions that are required by a role are assigned.

One solution is to further define the granularity of a role by studying the work that is being conducted by that role. My goal was to define a layered model that served as a basis for detailing an effective methodology to assign permissions to roles. This model concentrated on the assignment of flat roles to permissions. The model also required that the roles and permissions be defined. The methodology defined the layer-to-layer mappings, an aggregation approach, a decomposition approach, and model properties. After defining the methodology, I determined the benefits of the model by comparing it against other “decomposition” and “aggregation” models.

1.0 INTRODUCTION

1.1 Problem Statement

Role Based Access Control (RBAC) simplifies the administration of assigning permissions to users based on their job responsibilities and qualifications. Instead of directly assigning permissions to users for each application, users are assigned to roles and roles are mapped to permissions for each application. If the user needs to change his/her role, the administrator simply assigns a new role containing the appropriate permissions, rather than updating the user access to each application.

The well-known RBAC96's [SCFY96] Permission Assignment (PA) is a many-to-many permission to role relation. Using RBAC, it is possible to ensure that the necessary application accesses required to perform the work of an organization are mapped to a role.

The effectiveness of implementing the RBAC model lies in the administrator's ability to assign permissions correctly to a role. Currently, there is no formal framework for decomposing roles to permissions; or in the reverse, for aggregating permissions to roles. Starting from either a role or from the permissions, we need to develop an extensible, reusable framework that accurately defines the application accesses of a role.

1.2 Approach

In this research dissertation, I present a comprehensive approach to role engineering. I extend the Permission Assignment relation [SCFY96] of RBAC96 by creating a new model called Role Permission Assignment Model 2001 (RPAM01). RPAM01 supports a methodology to either decompose roles to permissions or aggregate permissions to roles, using either a role-focus, permission-focus or application-focus approach. When a role is decomposed, I answer the question: Which application's permissions are required for an agent to perform the role's work? Analogously, when I aggregate permissions, I answer the question: Which permissions need to be assigned to the roles to ensure that all of the work of the application can be effectively performed? In addition, there are other related issues, which need to be considered, such as minimization, reuse, and completeness. These are important factors for improving the effectiveness of the role/permission assignment.

My approach is to extend the permission assignment of the RBAC96 model between the roles and permissions. The extension does not require the modification of the end-points (i.e., roles or permissions) or defining the elements within these end-points. I concentrate on defining the mappings and the creation of the elements between the end-points, not the role and permission end-points themselves. I can allow the definitions of roles to be created by previously defined work such as "Role-finding" [RSW00] and the creation of permissions by the developers of those applications. As such, the definition of roles, permissions, and role hierarchy are not discussed within this dissertation.

RPAM01 provides a foundation for both the decomposition and aggregation of role/permission assignment. It introduces the concept of jobs, workpatterns, and tasks as intermediaries between roles and permissions. These intermediaries form a sequence: role, job, workpattern, task and permission. I also refer to these as layers. Each layer of RPAM01 and its relation to adjacent layers are formally defined. By these definitions, I provide further clarity and scope to the problem, following the well-known principle of divide and conquer.

I further enhance RPAM01 by applying properties to the framework. The properties: equivalence, uniqueness, minimization, reuse, and completeness assist the role engineer in defining PA.

Next, I define the decomposition methodology. I use the framework as a means to define the functional approach. The decomposition methodology begins by selecting a focus approach and then proceeds through each of the phases: role-to-jobs, jobs-to-workpatterns, workpatterns-to-tasks, and tasks-to-permissions. Each phase includes: 1) the development of the current layer and 2) the mapping from the current layer to the next layer.

Following the definition of the decomposition methodology, I use the role/permission framework to define the aggregation of roles to permission methodology. In contrast to decomposition, aggregation groups elements and map that group into a single element at the next higher layer (i.e., a group of permissions are mapped to a task, a group of tasks are mapped to a workpattern, etc.). Similar to decomposition,

aggregation defines its focus and then proceeds through the layers in a reverse order: permissions-to-tasks, tasks-to-workpatterns, workpatterns-to-jobs, and jobs-to-roles.

In either methodology, I do not redefine permissions. I consider a permission as the ability for an agent to access an application. I understand that the applications will process data and each set of data may have its own permissions.

In summary, I will show that it is possible to develop a model that can be used to engineer effectively the RBAC96's Permission Assignment (PA) relation.

1.3 Organization of the Dissertation

I begin the dissertation with a review of prior work, which was performed to examine the role/permission relation in Section 2. Next in Section 3, I formally define RPAM01. I then review the two methodologies of RPAM01: with the decomposition of roles to permissions, as discussed in Section 4, and the aggregation of permission to roles as discussed in Section 5. In Section 6, I validate these methodologies using three case studies. Finally, in Section 7, I summarize the conclusions and work for future research.

2.0 RELATED WORK

The work of roles did not start with the need to access computers but under another discipline, sociology for organizational theory [TB79], where a role is "set of rights and duties associated with a position, which are assigned to a person who occupies that position."[1] Even before Sandhu's work [SCFY96] in developing the RBAC model, a concept of roles was evolving; Ting [T88] researched a data security approach based on User-roles.

Prior to the creation of a central forum to present RBAC work, papers were introduced in publications such as Database Security, IEEE Computer Symposium on Research in Security and Privacy [B90], and NIST/NSA on Role Based Access Control [FK92]. Not until 1995, was the Association for Computer Machinery (ACM) Workshop on Role-Based Access Control created in the Washington, D.C. area.

During the first ACM RBAC workshop in 1995, there was a need to discuss fundamental concepts and to provide a standard definition for RBAC. Other discussions concentrated on the prioritization of RBAC features, roles versus Groups, issues, and future direction in RBAC [CY95]. There was even the first glimpse of Role Engineering

[1] [M98]

[C95], where Coyne's defines role engineering as "an approach to defining roles and assigning permissions to the roles."[2] He provides a simplistic seven-step approach for identifying roles. However, the paper does not provide details on how to assign permission to roles.

Since the initial RBAC conference in 1995, there have been five subsequent workshops, the latest in 2001, which broadened the scope of RBAC to include all Access Control Models and technologies.

RBAC papers have not been limited to the ACM conferences; there have also been papers published at the IEEE Applied Computer Security Applications Conference (ACSAC) such as "A New Model for Role Based Access Control" [G95] and "RBAC: Features and Motivations" [FCK95]. In addition, in the Sigmod Record, a paper was published on "Role-based Security Object Oriented Database and Separation of Duties" as early as 1993 [NO93].

Early research required RBAC to prove its benefits in the area of access control. This was accomplished by comparing RBAC against other security policies and access models. One such model is the famous Bell-La Padula (BLP) access model to Mandatory Access Control (MAC) [BLP75]. Nyanchama and Osborn compared the two models by interacting MAC with RBAC [NO95]. Later Osborn used a graphical mapping tool called a role graph [O97] to aid in determining when a single role or edge violated the conditions required by MAC. Analogously, Sandhu [S96] showed how RBAC

[2] [C95]

components could be configured to enforce lattice-based mandatory access controls. He demonstrated how lattice-based *-properties could be presented using RBAC terms.

RBAC research is not limited to theory; there have been implementations by Notargiacomo using Oracle 7 [N95], by Myers using Trusted DG/UX® [M97], and by Giuri [G98] using Java.

The ORACLE 7.0 relational database management system also allowed for the definition of RBAC application specific policies. Oracle RBAC implementation is consistent with Sandhu's RBAC96 model. As a result of their efforts, Oracle submitted and received acceptance from the ANSI/ISO X3H2 SQL standards committee for their role functionality into the SQL3 specifications.

Within DG/UX®, Myers suggested that there were mechanisms, which were flexible enough to support RBAC capabilities in different applications. The three mechanisms were DG/UX B2 Security Option Capability Access Control (CAC), creation of a user-created DG/UX B2 Security Option session initiator, or a user-developed, presentation layer. The low-cost, high-assurance, and application solution would be the CAC solution.

Giuri studies the security features of the Java platform to determine if they can be improved by implementing a role-based access control mechanism. He concluded that the Java Development Kit provided a framework that could be modified to implement RBAC policies and that further work was needed to consider explicit denials of authorization and the activation of roles within privileged security regions.

By reading RBAC and related materials I realized that the assignment of roles to permissions was not explained, rather, it was left to the reader to determine the method. There were many examples, but few explanations on how permissions were decomposed from the roles [G95] [B98] [GH00], and aggregated into roles.

Of particular interest were four papers that delved into role engineering. Coyne [C95] is the first to define the concept of role engineering, a preliminary approach, and its relationship to the RBAC96 model [SCFY96]. Three of the papers, Chandromuli [C99] [C01], Thomsen et al [TOP99], Roeckle, et al. [RSW00] provided details on how to perform role engineering. In fact, these three articles became a basis for my research and are further discussed in this section; and the latter two articles are analyzed in detail in chapter six. As with RPAM01, these efforts defined an approach, which assigns permissions to roles.

Chandramouli [C99] [C01] discussed a decomposition approach for identifying roles in a Healthcare Information System. Chandramolli had created the roles and their hierarchy from his years of experience working with the well-defined roles and structures of the healthcare system; however, as with the RPAM01 model, permissions must still be defined by the application developers. Thomsen, et al. [TOP99] presented an aggregation-layered methodology called Napolean. They define keys and key chains as opposed to roles and hierarchies. As with RPAM01, permissions are grouped into objects and are defined by the application developer. Roeckle, et al. [RSW00] describe their decomposition experience in role-permission engineering in a large corporate environment. Roeckle defined roles by using a process-oriented approach for “Role-

Finding." Once the roles are found, they can be aligned with the hierarchical organizational positions to create a role hierarchy. Similar to the stated research, permissions are defined by the application developers.

In addition to the research in role engineering, there is related work that can be used to support the definition and maintenance of roles and their assignments. After the PA has been defined, Barkley and Cinotta [BC98] and Perwaiz and Sommerville [PS01] provided a method for managing the assignments. Tidswell and Jaeger [TJ00] discussed further rules that may could to be incorporated into RPAM01 such as mutual exclusion and conflict of interest. Kang, Park, and Roscher [KPF01] discussed inter-organizational workflow. Nyanchama [N94] [NO94] [NO95] presented a role graph that is beneficial when depicting the relationship of decomposed hierarchical roles and for ensuring that the parent role does not contain a permission that is contained in its child's role.

2.1 A Role-Finding Approach

The most compelling related work for engineering of Role/Permission Assignment is the paper [RSW00]. Roeckle, Schimpf, and Weidinger use a process-oriented approach to find roles. They begin by defining the set of all roles by classifying roles. They deduce from their classification, that functional roles require a process-oriented role-finding approach. They use a meta-model that is composed of three layers: process, role, and access rights.

This decomposition approach starts with roles and works towards the rights and process required by the roles. It is a formal approach that can benefit from eliminating duplicate elements and from reusing existing elements of their meta-model. The Role-

Finding approach can further benefit by decomposing the role from the steps required to perform the components defined in the process layer: Job Position, Job Function, Security System Application, and Attribute Occurrences. The Role-Finding Approach, however, does not define an aggregation approach from permissions to roles.

In the following dissertation, I will discuss reuse and minimization properties, process definition, aggregation and decomposition.

In summary, while the existing models structure the identification of roles and permissions, the relations are insufficient. The models do not fully exploit aggregation and decomposition between roles and permissions. Furthermore, the existing approaches do not consider many role engineering efficiency alternatives. At best, they may consider reuse of components, but they do not consider other efficiencies such as eliminating redundancy.

2.2 The Napoleon Approach

The Napoleon work by Thomsen et al. encompasses two papers. The first paper [TOB98] describes the responsibilities of the application developer and the local system administrator in a seven-layer model. The second [TOP99] extends the Napoleon model by supporting workflows and policies in semantic layers.

The Napolean Model aggregates permissions to roles by using layers as building blocks to map permissions to roles. The layering is divided into three groups: Local Policy, Semantic Policy, and Application Policy. The authors state, “The Napoleon approach is that distinct sets of users maintain different parts of the policy based on their

understanding and their responsibilities."[3] Users are assigned to roles at the local policy layers. The purpose of the application "is to encapsulate application specific information so that it can be incorporated into the higher layers in a uniform manner." The middle layer, combines keys, key chains, policy, and constraints into semantic layers.

The layering concept provided several benefits, such as reusing previously defined layers and the ability to define policy. Introducing properties to the layers in order to reduce the number of duplicate elements can assist in the definition of the Napolean model. In addition, I will introduce a detailed approach on how to perform the aggregation of permissions to roles. Another enhancement is the ability to decompose from roles to permissions.

2.3 Access Control for a Healthcare Information System

Chandramouli [C99] describes a five-step methodology for defining an Access Control Service for a Healthcare Information System.

Step1: For a given Healthcare Information System, identify:

(a) Supported Business Processes and

(b) Information Domains associated with Business Processes;

Step 2: Identify Information Objects and Methods to support Business Processes;

Step 3: Identify mapping policy requirements to derive protection requirements for objects and their methods;

Step 4: Define roles and determine business processes to be assigned to Roles based on Protection Requirements; and

[3] [TOP99] page 146

Step 5: Determine access decision variables for each of the methods for a given access instance to determine effective rights.

Chandramouli later [C01] defined a framework, called DAFMAT (Dynamic Authorization Framework for Multiple Authorization Types), for multiple authorization types in a healthcare application framework. He joined RBAC with a Dynamic Type Enforcement (DTE), which was then combined with a logic-driven authorization engine in the DAFMAT model. The DAFMAT model provided a dynamic authorization framework.

Chandramouli's work can benefit from a detailed decomposition methodology. Chandramouli defined his DAFMAT components based on his pre-existing knowledge of the healthcare industry. For another organization where the pre-existing roles responsibilities are not known, RPAM01 can be used to decompose the roles. Another benefit of RPAM01 is that it allows the reuse of elements by decomposing the process into detailed steps. Further efficiency can be accomplished by eliminating unnecessary, unused, and equivalent elements. Although Chandramouli's approach provided a detailed decomposition methodology, it cannot aggregate the elements into roles.

3.0 RPAM01

The RBAC96's PA assignment can be considered to be a layered model, I call RPAM01. Instead of mapping roles to permission directly, I add three new layers: Jobs, Workpatterns, and Tasks. Each layer of RPAM01 (See Figure 1) is independent to all other layers.

RPAM01 describes a model that can be followed as either top down by assigning roles to permissions or bottom-up by assigning permissions to roles. If I start from the top I decompose roles into permissions. When I start from the bottom I aggregate permissions into roles.

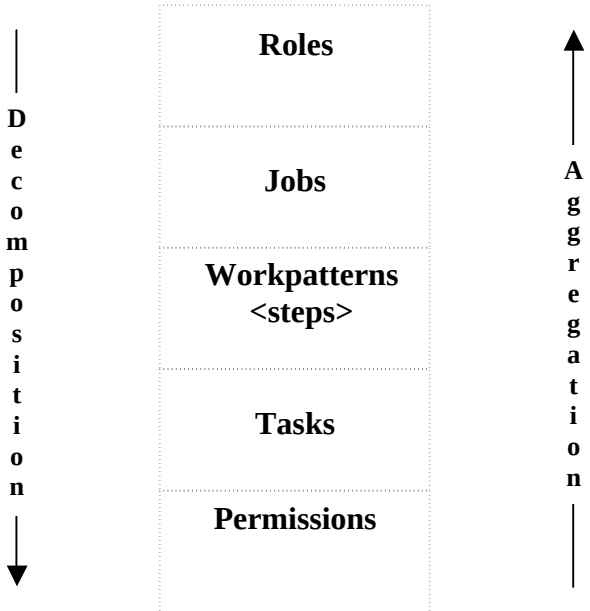

Figure 1: RPAM01

At the top of RPAM01 is the role layer. A role may be responsible to perform more than one type of work. Each type of work is defined as a job. The agent performing the role will follow some sequence of steps to complete an aspect of the job. The steps need not be in any sequence; but for organizational purposes, I group all the steps that may be performed by a job into a set I call a workpattern. Each workpattern may require one or more permissions to access applications. Instead of mapping directly to permissions, I realize that there is a benefit of grouping permissions into a set that other workpatterns can also use rather than re-identifying individual permissions every time a workpattern is defined. I start the grouping with the concept of a step. Each unique workpattern step is assigned to a task. The workpattern in turn is mapped to these tasks. I do not know until I decompose tasks into workpatterns if the tasks will map to a set of permissions. If the task does not map to a set of permissions, I label the task as permission-free.

A quick example of the layered RPAM01 is the role of a professor who performs the jobs of a teacher and a researcher. In the example, I will underline the steps that will map to tasks. These tasks will be mapped to permissions. In the workpattern of a teacher, the steps: create lecture, test mid-term exam, record mid-term grades, lecture, test final exam, and record final grades are performed. In the workpattern of a researcher, the steps: hypothesize, research hypothesis, document hypothesis, research solution, document solution, and lecture are performed. From the first workpattern, I have lecture and record, while the second workpattern document and lecture that require permissions to access applications. So I map the first workpattern to the tasks lecture and record, and

I map the second workpattern to a new task called document, and reuse the task called lecture from the first workpattern. I then map the tasks to the permissions that grant accesses to the needed applications.

This chapter details the background, scope, definition, aggregation, and decomposition of RPAM01.

3.1 Background

The Engineering of Role/Permission Assignments research is derived from RBAC96 [SCFY96]. There are three components of RBAC96 that I use for the RPAM01 extension: users (U), roles (R), and permissions (P).

The RBAC96 model, illustrated in Figure 2, defined the components of role/permission assignment by PA. It also defined a role hierarchy RH. It does not, however, state how to engineer a role/permission assignment.

In Figure 2 and later in Figure 3, the double-headed line means “many.” A single-headed line represents “one.” A double-headed arrow represents a many-to-many relation between the sets.

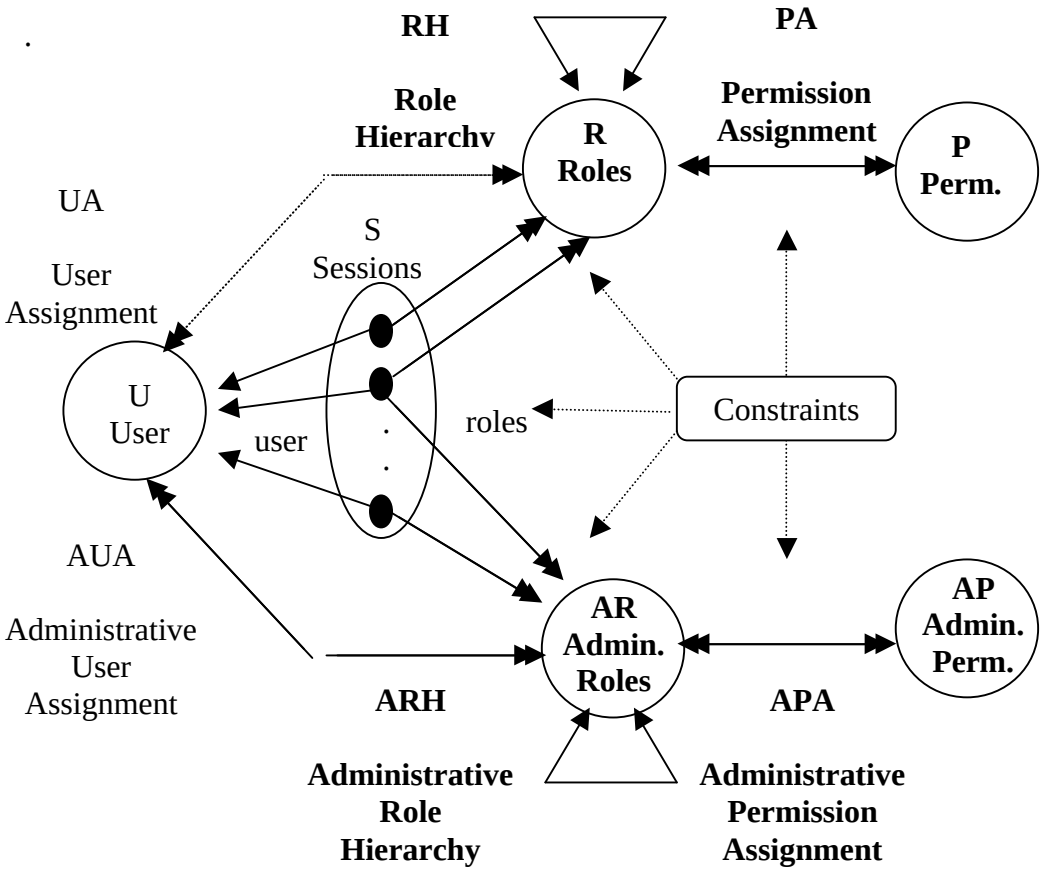

Figure 2: RBAC96

This dissertation concentrates on the permission assignment (PA) of roles to permissions. Figure 3 illustrates the roles, permissions, and PA. The PA aspect of the RBAC96 model is expanded to create an approach to engineer roles to permissions and permission to roles.

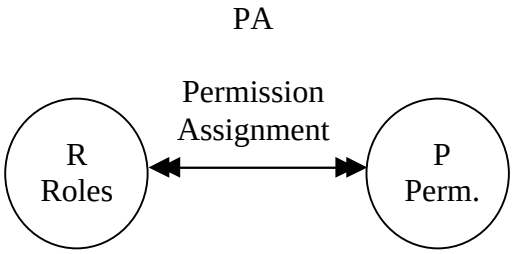

Figure 3: RBAC96's Permission/Role Assignment

The notational syntax that I use for my extension from the RBAC96 model requires that:

- R is a set of roles;
- P is a set of permissions; and
- PA $\subseteq$ P x R is a many-to-many permission to role assignment and administrative permission to administrative role assignment relations.

3.2 Scope

In this dissertation, I research the design of role/permission assignment prior to the deployment of the system.

RPAM01 uses a subset of the RBAC96 model that does not include constraints, any administrative components, or user/role assignments. It does include roles, permissions, and role/permission assignment. I assume the users, roles, and application permissions pre-exist.

I investigate two approaches: decomposition and aggregation. In both approaches, the organization administrator identifies roles and the application developer defined permissions.

3.3 RBAC Extension

I have modified the RBAC96 model as depicted in Figure 4 by adding three sets: Jobs, Workpatterns, and Tasks.

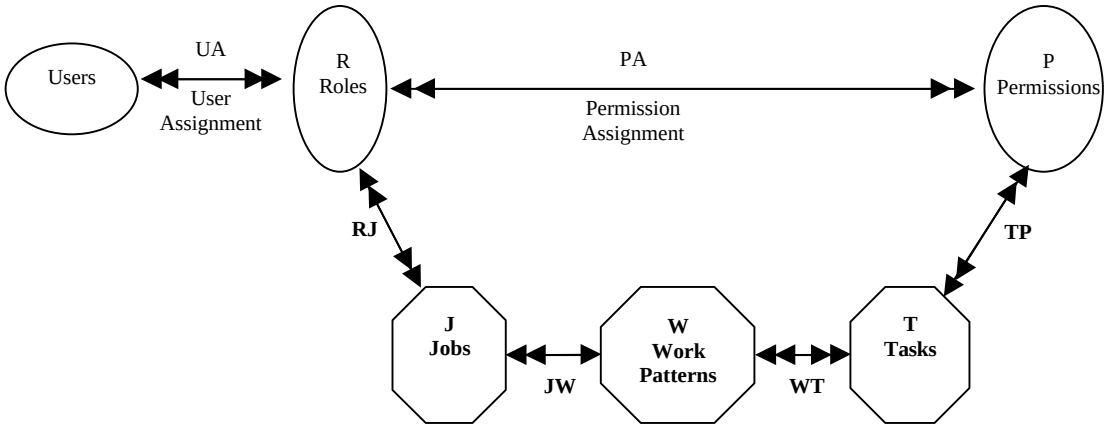

Figure 4: RBAC96 Extended

The notational syntax and the terms that are used in this extension follows:

- R is a set of Roles;
- J is a set of Jobs;
- T is a set of Tasks;
- P is a set of Permissions;
- RJ $\subseteq$ R x J is a many-to-many role to job assignment relation;

- JW $\subseteq$ J x W is a many-to-one job to workpattern assignment relation;
- WT $\subseteq$ W x T is a many to many workpattern to task assignment relation; and
- TP $\subseteq$ T x P is a many-to-many task to permission assignment relation;

As with the RBAC96 model, the double-headed line means "many." A single-headed line represents "one." A double-headed arrow represents a many-to-many relation between the sets. In the case of jobs-to-workpattern, a double-headed arrow points to jobs and a single-headed arrow points to workpatterns, so jobs-to-workpattern is a many-to-one relation.

The many-to-one relation restricts the JW mapping to a single solution for each job, which eventually will be mapped to a set of permissions. If I allow JW to be a many-to-many mapping, a job will be able to be mapped to multiple solutions or different sets of permissions. The PA needs only one solution for a role to have the needed permissions to access the applications to perform its work, since multiple solutions provide limited benefit. In fact, there will be RPAM01 inefficiencies because:

- Additional solutions require additional resources to create those solutions;
- Additional solutions require additional maintenance resources to maintain the additional solutions;
- Any changes or review of jobs will require the analysis of all solutions;
- Each different solution is another set of permissions that provided a different access path to the applications;
- Restricting access to the work performed by a job cannot be denied simply by a single job (each solution must be restricted); and

- Analysis for unauthorized access is required for each solution.

One of the goals of this dissertation is to detail a methodology to increase efficiency when performing the role/permission assignments. 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. I can improve the engineering of roles by reducing the number of elements in a layer to the exact elements required by the mappings. This can be accomplished by deleting duplicate elements that map to all of the same elements that are contained in a lower layer or by not defining new elements by reusing existing elements. To work towards these goals, I introduce properties that can be applied to the elements within a layer and the mapping between layers. These properties are: equivalence, uniqueness, minimization, reuse, and completeness. They will be defined later in greater detail.

I strive to minimize the number of jobs, workpatterns, and tasks that will be used to perform the role/permissions assignments. Ideally, each element is unique and therefore each set will not contain duplicate entries. I can determine if an element is unique if there is not another element that is equivalent to that element. My real interest in equivalence is that, when I finish mapping the elements of a layer to permissions, I want to know if the layered elements map to the same set of permissions. If the elements are equivalent, then the element will grant exact, same permissions to the application; and there may not be a benefit to have more than one element mapping to the same set of permissions. To continue with this line of thought, I may not need to define another element if I can reuse an existing element if it maps to the desired element(s). Once I

finish with the approach, I verify that all the pre-defined elements (i.e., roles and permissions) have been mapped. I check the completeness of PA by mapping each role to at least one permission and each permission to at least one role. As I stated earlier, I strive to minimize the number of elements; however, there may be a benefit in not eliminating duplicate unique elements. I will discuss these potential benefits later in this dissertation.

For each of the following properties, there is a formal definition. These definitions are based on the following terms:

- X is a set of layered elements;
- Y is a set of layered elements;
- $f(x) = y$;
- $f^{-1}(y) = x$;
- If f is a function from X to Y, then f is **onto** if for each y in Y there is an x in X such that $f(x) = y$;
- $f\ (x{:}X) \rightarrow 2^{Y}$ the mapping of set of X layered elements to a set of Y layered elements; and
- P is a set of Permissions.

Property 1: Equivalence –

¦ $(x_a:X)$ ® Y_c
¦ $(x_b:X)$ ® Y_d
If $\{Y_c\} = \{Y_d\}$ then I say x_a is <u>equivalent</u> to x_b in the set X, which is denoted as $x_a @ x_b$

Property 2: Permission Equivalence –

¦ $(x_a:X)$ ® P_c
¦ $(x_b:X)$ ® P_y
If $\{ P_c \} = \{ P_d \}$ then I say x_a is <u>permission equivalent</u> to x_b in the set X, which is denoted as $x_a @_p x_b$

The equivalence property is applicable to the J, W, and T layers. Two sets, within the same layer, are equivalent if they contain the exact same elements. Permission equivalence available only in the W layer is a special case of equivalence and is defined as two sets that map to the same set of permissions. 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 such as logon to the computer and check e-mail, and require a third task of fax 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 map to slightly different tasks; consequently, workpatterns A and B are permission equivalent workpatterns.

Property 3: Uniqueness –

x_a , x_b Î X
If x_a is not equivalent to x_b then x_a is <u>unique</u>.

Uniqueness, applicable at all layers, checks if two or more elements are equivalent. If they are, I may be able to minimize the number of elements. For example, the Information Technology and Psychology Departments require the same set of permissions 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. I work towards uniqueness when I: 1) minimize duplicate elements or 2) reuse a unique element. I need to be careful that I do not minimize an element that is needed for role-permission completeness (completeness is defined later in this section).

Property 4: Minimization –

$x , x_i \hat{I} X$
¦ (x:X) ® Y
¦ (x_i:X) ® Y_j
" x_i that are equivalent, I can say x is the <u>minimization</u> of the x_is and {Y} = $\dot{E}_{i=1, \ldots, n} \{Y_i\}$.

The minimization property, only available in J, W, and T layers, merges copies of equivalent elements. As stated earlier, minimization is a goal but not a requirement**.** Equivalent elements can be minimized into one element to eliminate the need to administer multiple copies of elements that map to the same tasks. I can also minimize permission equivalence workpatterns, by either including some, all or none of the permission-free tasks in the minimized element. This decision is left to the role engineer in determining the needs of the workpattern with respect to future permission implementation. In the previous paragraph, minimization was performed on the

workpattern layer that contained equivalent workpatterns A & B by eliminating workpattern B.

Property 5: Reuse –

¦ $(x_a:X) \rightarrow Y_c$
¦ $(x_b:X) \rightarrow Y_d$
If $y \in \{Y_c\} \cap \{Y_d\}$ then I say workpatterns x_a and x_b in the set X are <u>reusing</u> y

Instead of inefficiently creating a new element every time it (e.g., element A) is mapped to exactly the same element in another layer (e.g., element X), I can reuse the element in the destination layer. For example, element A already maps to element C in the destination layer, and I create a new element B that can also be mapped to an existing element C. There is no reason to create another element in the destination layer called D; I can simply reuse element C. The reuse property permits two elements from one layer to reuse the same element from an adjacent layer. Using the previous Information Technology example, when the Psychology Department 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.).

Property 6: Completeness –

If ¦ (x) and ¦ $(y)^{-1}$ are "onto" then I say X is <u>complete</u> to Y.

A final property to define is completeness. A layer is complete if all the elements of its layer are mapped to an element in the same destination layer. There is completeness of roles, jobs, workpatterns, and jobs; but they are all subservient to completeness of permissions. I can also consider role-to-permission completeness, completeness of permission to roles, and roles to permission. This property is important because, if a permission is not assigned to at least one role, then the application being granted access by the unassigned permission is inaccessible; and, thus, the work performed by that application cannot be accomplished. For example, the human resource application has a permission to backup human resource data; and if the backup permission is not granted to a role, a backup cannot 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 permissions that grant access to any applications.

In summary, there are equivalence, uniqueness, minimization, reuse, and completeness properties that apply to RPAM01. Not all of the properties apply to each layer. Table 1 shows the applicable property by an "X" in the relevant layer for the decomposition approach, and Table 2 shows the aggregation approach. The completeness verification starts from the reverse direction. In the case of decomposition, the verification starts at the permission layers while for aggregation it begins at the role layer.

The only difference between the two tables is in the reuse and completeness columns. The first reuse property difference is caused by the many-to-one mapping restriction. I am restricted from decomposing one job into multiple workpatterns and from aggregating many workpatterns into one job. In addition, I can not reuse the elements of the layer in which I begin the approach (i.e., in the decomposition approach I do not reuse roles). The completeness property is important when I map to the next layer. With respect to the decomposition algorithm I map to the next lower layer. With respects to the aggregation algorithm, I point to the layer above it. The decomposition approach does not have a layer below permission and, similarly, there is no layer above role when I use the aggregation approach. Therefore, an "X" is not indicated in the Completeness column for the final layer of each approach.

Table 1: Decomposition Table of Properties

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

Table 2: Aggregation Table of Properties

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

Before I illustrate an example, I want to exemplify the importance of a workpattern. The workpattern layer is different from the other layers because it: 1) is a subset of another set I call steps, 2) adheres to the permission equivalence and equivalence properties, and 3) has a JW many-to-one relation.

A workpattern is created to assist the role engineer in identifying permissions. It assists in translating conceptual work into an organized, related set of steps. Although the steps may be related, they do not have to be in sequence. One step can occur repeatedly in a workpattern; however, only one occurrence of the task that represents the permission required by that step is mapped to the workpattern. If a workpattern does not contain at least one step, I will consider the workpattern meaningless and delete it. For example, there is no benefit of having a workpattern that performs no work.

Permission equivalence is based on the concept of permission-free tasks. The difference between equivalence workpatterns and permission equivalence workpatterns is the addition of permission-free tasks mapped to by the workpatterns. This leads to the understanding that two separate jobs mapping to two distinct permission equivalent workpatterns may not be equivalent because their workpatterns may map to different permission-free tasks. As a corollary, two equivalent workpatterns must be permission-equivalent.

The benefit of a permission-free task is that it can be used as a placeholder for a future defined property. By means of the minimization property, a role engineer chooses to delete or retain a permission-free task. For example, there may be a task to place a

telephone call from your workstation. Currently, there is no special permission required to access the telephone; later, there may be a need to use an access code to dial a phone number. I know in the future, that the special permission will be required. I do not want to forget about the future access needs, so I insert a placeholder in the workpattern to remind me about future enhancements.

The generalized six properties discussed earlier can be tailored to the workpattern. They are listed below along with the additional definitions of steps and the permission-free task.

- S is a set of Steps;
- W is a set of Workpatterns, $W \subseteq 2^S$;
- Workpattern_tasks(w:W) $\rightarrow$ 2^T the mapping of workpattern w to a set of tasks;
- Task_permissions(t:T) $\rightarrow$ 2^P the mapping of task t to a set of permissions; and
- TP $\subseteq$ T x P is a many-to-many permission to task assignment relation.

Equivalence
Workpattern_tasks(w_a:W) $\rightarrow$ T_x
Workpattern_tasks(w_b:W) $\rightarrow$ T_y
If $\{T_x\} = \{T_y\}$ then I say w_a is **equivalent** to w_b in the set W, which is denoted as $w_a \cong w_b$.

Permission Equivalence
Workpattern_tasks(w_a:W) $\rightarrow$ T_x
Workpattern_tasks(w_b:W) $\rightarrow$ T_y
Task_permissions(t_x:T) $\rightarrow$ P_c
Task_permissions(t_y:T) $\rightarrow$ P_d
If $\{ P_c \} = \{ P_d \}$ then I say w_a is **permission equivalent** to w_b in the set W, which is denoted as $w_a \cong_p w_b$.

Permission-free task:
Permission_free(t) = $\{p \in P \mid (t,p) \notin tp\}$

Uniqueness:
$w_a , w_b \in W$
If w_a is not equivalent to w_b then w_a is **unique**.

Minimization
$w , w_i \in W$
Workpattern_tasks(w:W) $\rightarrow$ T
Workpattern_tasks(w_i:W) $\rightarrow$ T_j
$\forall$ w_i that are equivalent, I can say w is the **minimization** of all w_is and $\{T\} = \cup_{i=1, \ldots, n} \{T_i\}$.

Reuse
Workpattern_tasks(w_a:W) $\rightarrow$ T_x
Workpattern_tasks(w_b:W) $\rightarrow$ T_y
If $t \in \{T_x\} \cap \{T_y\}$, then I say t is being **reused** by workpatterns w_a and w_b in the set W.

Completeness

If Workpattern_tasks and Workpattern_tasks^{-1} are "onto", then I say W is **complete** to T.

3.4 Model Example

Before I present RPAM01, I consider a design example (See Table 3.): 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.

Doctor is a role in R. 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 the 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 hospital database (P_1). Task T_2 requires a permission to review the doctor's office record (P_2). Task T_3 requires permissions to the three referring doctors' records (P_3); and Task T_4 requires two permissions: the doctor's record (P_2) and the patient's record (P_6).

The set of Roles = {R_1 (doctor)},
There is a set of Jobs = {J_1, J_2, J_3, J_4 },
There is a set of Tasks = { T_1, T_2, T_3, T_4 },
There is a set of Permissions = { P_1, P_2, P_3, P_4, P_5, P_6 }

Table 3: Doctor Example

Role	Job	Workpatterns	Steps	Tasks	Permission
R_1	J_1	W_A	S_1, S_2, S_3, S_4	T_1, T_2, T_3, T_4	P_1, P_2, P_3, P_4, P_6

I can show the agility of using the notational syntax by a more complex abstract example. The example is:

There is a set of Roles = {R_1, R_2, R_3 }
There is a set of Jobs = {J_1, J_2, J_3, J_4 }
There is a set of Workpatterns = {W_A, W_B, W_C, W_D}
There is a set of Steps = { S_1, S_2, S_3, S_4, S_7 }
There is a set of Tasks = { T_1, T_2, T_3, T_4, T_7 }
There is a set of Permissions = { P_1, P_2, P_3, P_4, P_5 }

For this example, I am given the following information:
J_1 = W_A that contains = { S_1, S_2, S_1, S_1 } that requires tasks { T_1, T_2}
J_2 = W_B that contains = { S_2, S_7 } that requires tasks { T_2, T_7 }
J_3 = W_C that contains = { S_3, S_4, S_3} that requires tasks { T_3, T_4 }
J_4 = W_D that contain = { S_3 } that requires task { T_3 }

Tasks T_1 requires permission = { P_1, P_2, P_3}
Tasks T_2 requires permission = { P_2, P_4 }
Tasks T_3 requires permission = { P_2}
Tasks T_4 requires permission = { P_3, P_5}
Tasks T_7 requires permission = { P_2, P_5}
R_1 = {J_1}
R_2 = {J_2, J_3}
R_3 = {J_4}

Table 4: Role/Permission Syntax Example

Role	Job	Workpatterns	Steps	Tasks	Permission
R_1	J_1	W_A	S_1, S_2, S_1, S_1	T_1, T_2	P_1, P_2, P_3, P_4
R_2	J_2, J_3	W_B, W_C	S_2, S_7, S_3, S_4, S_3	T_2,T_3, T_4,T_7	P_2, P_4, P_3, P_5
R_3	J_4	W_D	S_3	T_3	P_2

Table 4, shows the data from the example. The stated syntax is used in chapters 4 and 5 to discuss RPAM01. Specifically, I minimize equivalent tasks and permissions, reuse tasks and permission, and identify minimal unique sets.

4.0 DECOMPOSITION

The RBAC96 model is extended to create a methodology to map roles to permissions, (See Figure 5.). There are four phases: Roles-to-Jobs, Jobs-to-Workpatterns, Workpatterns-to-Tasks, and Tasks-to-Permissions. Each phase will be discussed in detail in its own section. The final section summarizes the entire methodology.

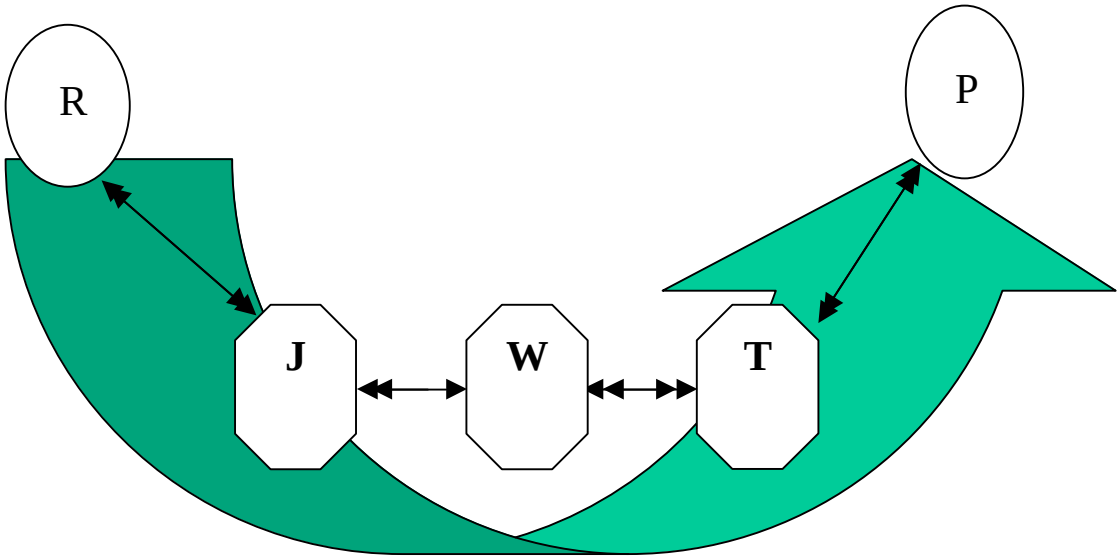

Figure 5: Decomposing Roles to Permission

Each section is discussed in detail by describing:

- the layer's objectives;
- the issues to be resolved;
- an approach to resolve the issues;
- an example; and
- the consequences of not implementing the layer.

Before I begin, I highlight a few important characteristics of the decomposition approach that are addressed later in the following sections. I observe the following in Figure 6:

- The many-to-one relation of jobs to workpatterns, with J_1 and J_2, both map to W_A (a many-to-one mapping, the reverse mapping of many workpatterns to a single job is not allowed);
- Two different workpatterns, W_A and W_B, map to the same task T_8 (the task is being reused);
- Two different tasks map to the same permission T_1 and T_8 to P_1, and T_1, T_4 and T_8 to P_5 (the permissions are being reused); and
- T_2 is a permission-free task.

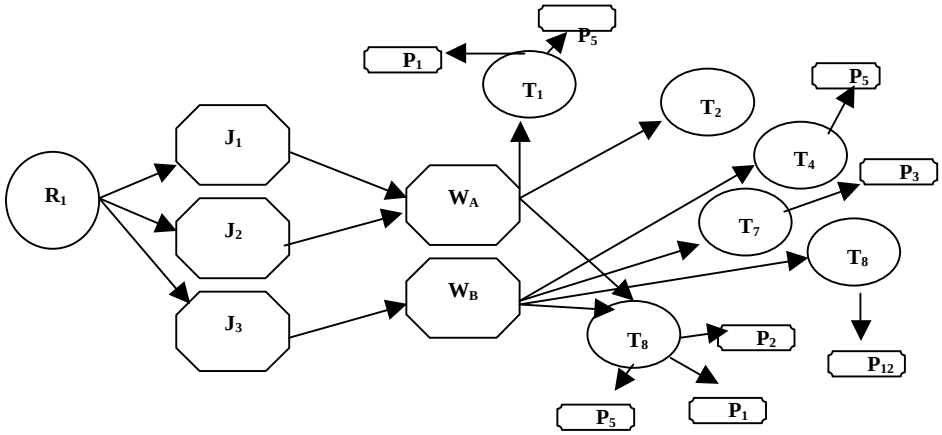

Figure 6: Roles to Permission Mappings

4.1 Roles to Jobs

The first phase of the approach is to decompose roles into jobs. I map pre-existing roles to jobs as a many-to-many relation (See Figure 7).

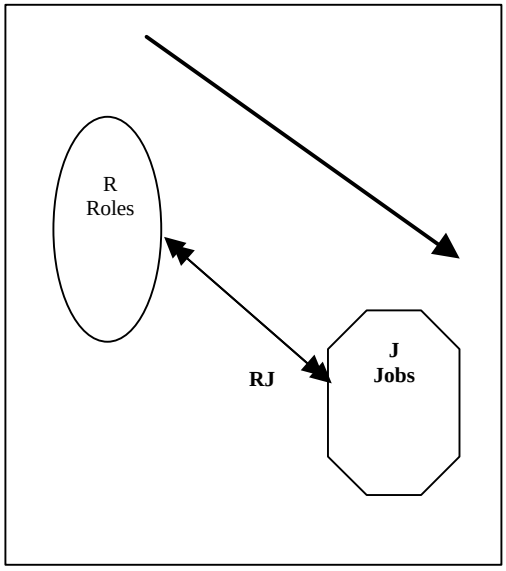

Figure 7: Role to Job Layer

The challenge is to create a methodology to decompose an element from the current layer into elements of the next layer. I do not want to arbitrarily decompose an element but rather develop the decomposition based on sound decision criteria. To assist in the decision process, I introduce a concept called "focus-based." I focus my engineering decisions based on a set of pre-determined criteria.

Focus-based requires the definition of two components. I first define my focus for the future role engineering decisions and then choose the criteria that will be used as a basis for these decisions. For this dissertation, I select one of three focuses: role, application, or permission. Then I choose the attributes that will be used as the decision criteria.

As one may expect, a "focus-based" role involves making decisions based on the characteristics of the pre-defined role; "focus-based" application involves making decisions on the characteristics of the applications that are being accessed; and "focus-based" permission involves making decisions on the characteristics of the pre-defined permissions.

After choosing the focus, I select the focus attributes that can best guide my role engineering decisions. Table 5 suggests how to define the attributes.

Table 5: Focus Approaches

Role	Application	Permission
▪ Learn about each role ▪ Determine key information about the work of the role ▪ Define role attributes	▪ Learn about each application's functionality ▪ Analyze the interrelations between functionality ▪ Define application attributes	▪ Learn about each application ▪ Identify the permissions for each application ▪ Learn the access capabilities of each permission ▪ Analyze how the access capabilities may interrelate ▪ Define permission attributes

I can identify focus attributes for

- Roles as skill sets, educational level, abilities, experience;
- Applications as functionality, manageability, interoperability; and
- Permissions as operating system, access type, application type, and capability.

For example, I realize I have a set of university roles. I determine from the roles that the important attributes are education, experience, and discipline. If I wanted to focus on application, I may choose the attributes of functionality (i.e., human resource, accounting, etc.), type (i.e., mail, database, etc.), and interoperability (i.e., applications that will communicate to another application via the internet).

After choosing the criteria, I learn more about the role's responsibilities by categorizing each role into one of the following groups where:

1. the responsibilities of a role have been documented,
2. the responsibilities of a role have not been documented but where the role has been defined, and
3. neither a role nor the responsibilities have been defined.

For roles that fall in the first group, I divide a role's responsibilities into sets based on the criterion defined from the focus-based approach. Related responsibilities that can be part of the same job set are merged into "like job" sets, J_1, … J_n. Unless there is a special need to have a non-unique job, a job is reused and each J_i is unique. For example, the responsibilities of the office administrator role can be to 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 and its responsibilities have not been documented. Extra effort is required to determine the responsibilities from the undocumented roles. This is accomplished by monitoring and then by documenting the activities performed when a user has activated a role. From the documentation and my knowledge of the chosen approach and its attributes, I analyze the responsibilities required to perform a role's activities similar to the earlier sets, J_1, … J_n. Unless required by the organization, jobs are reused and each J_i is unique.

For example, I have a role for a computer administrator. The computer administrator has not had time to create her list of responsibilities; however, a role engineer can follow the computer administrator while she performs her role to record her activities. The approach is application focus-based and the attributes are functionality and manageability. The jobs are chosen based on the attributes of application functionality and manageability. By observing the work of the role, I determine that the computer administrator performs archiving, software maintenance, and password management of application servers. Subsequently, I map the computer administrator role to the jobs of:

Application Server Archiving, Application Server Software Maintenance, and Application Server Password Management.

The final group is for a role that has been identified, but has not been defined or documented. I need to deduce the role's expected responsibilities by interviewing the designer of the organization. I then document each role's responsibility based on the focus approach criterion. Related responsibilities that can be part of the same job set are merged into like job sets, J_1, … J_n. Unless required by the organization, a job is reused and each J_i is unique. For example, if the government agency ABC.gov has a new position for a Chief Information Officer (CIO) that is required by the Clinger-Cohen Act[4], I need to create the jobs that perform the responsibilities of a CIO role. For many government agencies this is a new role, although it does exist in some agencies and certainly is commonplace in industry.

I need to determine from management what the responsibilities are of the CIO role. The approach is role-focus and the attributes are skill sets and experience. After talking with the Chief Financial Officer, Chief of the Agency, and the Chief of Operations, I determine that the jobs are Program Oversight, Technical Management, and Budget Review. Although a role has the skill set to understand technological information, it does not have the experience to perform in-depth technical review. Fortunately, the CIO can hire a person to perform a role that contains the job of in-depth technical reviewer.

---

[4] "The Clinger-Cohen Act of 1996 was previously called the Information Technology Management Reform Act (ITMRA or, PL.104-106.). It establishes the role of a Chief Information Officers in the government, and forms the interagency Chief Information Officers' Council." Documented on the NSF Clinger-Cohen Act Page by the National Science Foundation Office of Information and Resource Management.

The benefit of this layer is beginning with a comprehensive approach to assign permissions to roles. A role is complex; and without knowing its responsibilities, there may not be permission completeness to a role. Table 6 is a summary of the approaches.

Table 6: Approaches for Defining Role's Responsibilities

Documented	Existing	Undefined
▪ Read the write-up of the roles ▪ Identify responsibilities ▪ Categorize responsibilities by job	▪ Monitor a User performing a role ▪ Document a role's responsibilities ▪ Categorize responsibilities by job	▪ Interview a role's designer ▪ Deduce responsibilities ▪ Document a role's responsibilities. ▪ Verify responsibilities by simulating a role ▪ Categorize responsibilities by job

A role-job phase begins by detailing a role. Through decomposition, I learn the responsibilities of each role. These responsibilities are the jobs that identify the work that can be performed by a role.

As each subsequent role is decomposed into jobs, I may find that one of the jobs exists as part of another role. Rather than create a non-unique job, I can map a role to the same job that was previously defined. By reusing jobs, I obtain the benefit of saving development, administration, and maintenance resources.

Rather than solve the harder problem of assigning roles to permissions, I use the divide and conquer principle to subjugate the roles into categories of jobs. This phase lays the foundation to decompose the job into steps by using a process flow. Without this phase, I could not decompose the more complex role into steps by using a process

flow. For a simple, intuitive role, this phase may not be needed; however, for a multi-faceted role I may miss work responsibilities that could prevent the role permission assignment from complying with the completeness property. In addition, without this phase, each role will need to be entirely decomposed, and I will not have the benefit of reusing previously defined jobs.

An example of applying the above stated methodology is shown in Figure 8. A role-focus approach was used to define the jobs of three roles: R_1, R_2, and R_3. R_1 performed the jobs of J_1, J_6, and J_3; R_2 performed the jobs of J_1, J_9, and J_{72}; and R_3 only performed the jobs of J_{72}. Fortunately, I can reuse jobs J_1 and J_{72} because the former was defined during the decomposition of R_1, and the latter for R_2.

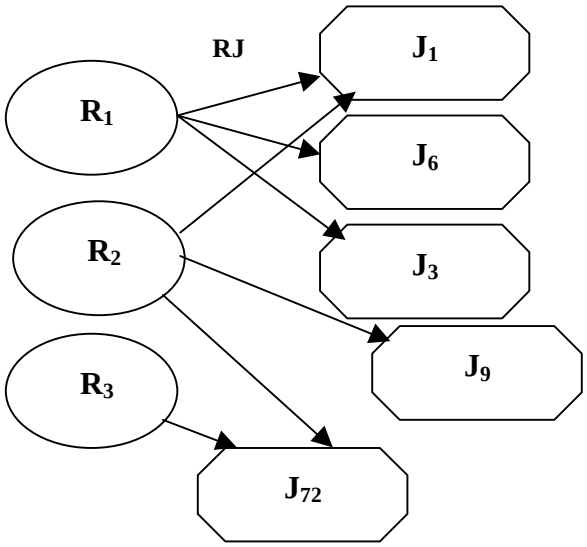

Figure 8: Role-to-Jobs Example

4.2 Jobs to Workpatterns

My next objective is to identify the set of steps that is required to perform the work of the job. The job to workpattern phase is the only mapping that is not many-to-many, but is a many-to-one relation (See Figure 9.). The reason for this limitation is described later in this section.

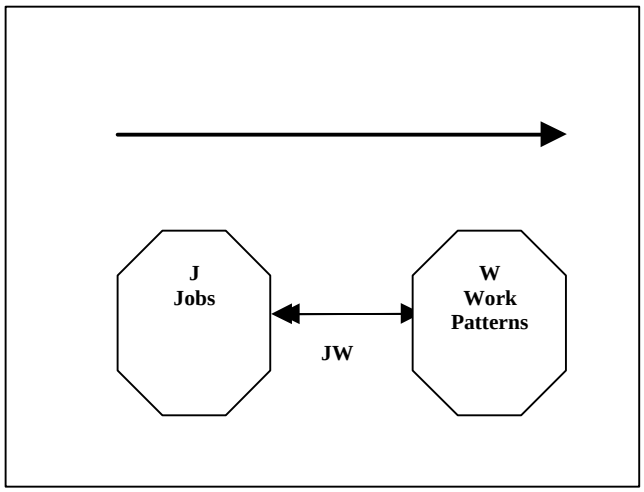

Figure 9: Jobs to Workpattern Layer

In this phase, each job is mapped only to one workpattern and the steps contained in the workpattern are defined. The workpattern provided an engineering aid to explicitly detail the steps that are required to perform all of the job's activities. If there are disjointed activities in a job, each group of activities can be split into its own distinct job.

The many-to-one mapping introduces the following limitations:

- There is only one unique workpattern for each job. One job cannot be mapped to two distinct workpatterns, as depicted in Figure 10;

- All steps required to perform the work of the job must be contained in the same workpattern; and
- There is only one set of permissions that is required to perform the work of the job.

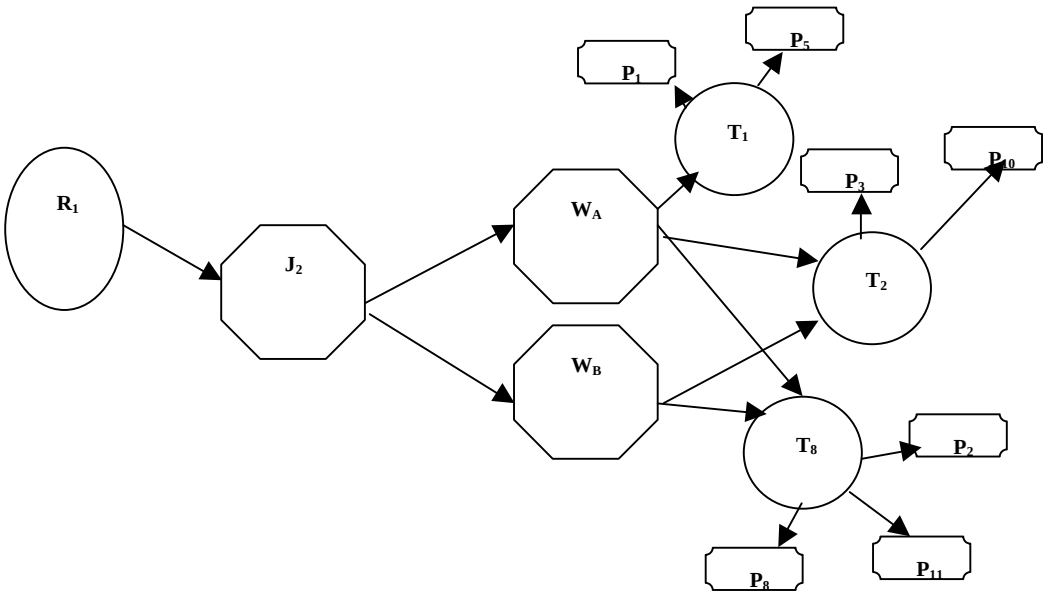

Figure 10: One to Many Jobs to Workpattern Example

The benefits of the many-to-one relation include:

- There cannot be two sets of permissions that can perform the same work. This condition increases the security risk by allowing more than one way of performing the same work. From Figure 10, I see that J_2 maps to W_A and W_B. The difference is that W_A contains P_1 and P_5, as opposed to W_B's P_8, P_3, and P_{11};
- All of the steps of the job can be identified in one set: the workpattern;

- The complexity and resources needed to maintain multiple sets of jobs is reduced; and
- There are no duplicate workpatterns.

If there are two jobs that map to the same workpattern, then both jobs require the same permissions. For minimization, I prefer to minimize jobs that are equivalent. This may not be possible because

1. Two jobs within two different organizations may desire to maintain a name distinction for their jobs although the work required by the jobs is equivalent, or
2. I choose to maintain the uniqueness of permission equivalent workpatterns so that I can allow for future permission growth.

The current phase begins by mapping jobs to workpatterns. Jobs are simply either mapped to new or to existing workpatterns. I determine the mapping by reviewing the work performed by the job. If the work of the job does not match, I create a new workpattern. If they do match, I determine if there is a need to map to a separate workpattern by the two criteria stated earlier: 1) there is a need by the organization or 2) for future permission growth. The latter cannot be determined until the tasks-to-permission phase has been defined.

The second part of this phase is to list all of the steps contained in the workpattern that are required to perform the work of the 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 I can identify the step's

logical sequence as a process, I have an engineering aid to reduce the complexity of the workpattern definition. I 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; or
2. The steps are part of multiple processes. At least one step is not in the same workpattern and the workpattern must wait for another step that is outside of the workpattern to finish; or
3. The steps cannot be defined as a process.

Single process is a set of steps that must be derived from the job. I know that the steps are formulated, as a process so there is some semblance of sequence. I define the process within the criteria of the focus approaches. For example, if there is a role-focus approach for a role of Professor for the job of Teaching within the criteria of Educational Level and Skill Sets, I determine that the process steps that are required to satisfy the work of a role are:

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

For multi-process, I need to identify the steps within the external process that the workpattern satisfies. Ideally, the master process has been created and the steps have been defined. Thus, I 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 I am aware that a job is part of the external process, then I define the steps. As with single process, I know that the steps are formulated as a process, so there is some semblance of sequence that is done to satisfy the work of a role. I define the process within the guides of the focus approaches discussed earlier.

For example, if the job is for a mortgage collection clearing house, I need to understand that the job is part of a larger process that includes other jobs such as the mortgagee (the person paying the mortgage) and mortgagor (the company receiving the money). I determine from the information that I obtained when I defined a job 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)>.

The last suggested method is an ad-hoc set of steps that may not be related. I 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. I must deduce from the present information what steps are required by the workpattern. For example, I may determine from the documented role of a computer 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.

After creating the steps, I need to review my work. I check for completeness. If there is a missing step, I need to add it to the workpattern. I can also divide large

workpatterns that contain too many steps into multiple jobs. Each new job will map to its own workpattern. Table 7 is a summary of the three approaches for defining the steps.

Table 7: Workpattern Definition

Single-process	Multi-process	Ad-hoc
▪ Identify the process. ▪ Identify the steps within the process.	▪ Identify the different processes. ▪ Identify the number of workpatterns. ▪ Identify the workpattern steps within the process.	▪ Deduce existing capabilities and document or use the workpattern as a holder for remaining tasks to ensure that all work can be performed. ▪ Determine if steps need to be combined. Determine if the workpatterns need to be subdivided; if so, divide them into more than one job.

After all the steps have been defined, I create one task to represent all occurrences of each step in a workpattern. The workpattern is then mapped to that task. Then, in the next section, I map the tasks to permissions.

The job-workpattern mapping provide two benefits: 1) the reuse of workpatterns and 2) the capability for defining permission in a logical method of a process that reflects the work of a business. Thus, instead of saying what application permissions I need to perform a role, I can consider the question, "What are the steps the job needs to perform the work of a role?" The answer further describes the logical flow of decomposing, enabling me to research my ultimate goal of assigning permissions to roles. I try to determine the purpose and the process that is performed by a role. A final benefit of a

job-to-workpattern mapping is that the ad-hoc set can contain all other jobs that are required to ensure role-permission completeness.

If I do not have the job-permission phase, I assign jobs to tasks without trying to deduce a logical method of determining all the steps that are required to perform the work of the job. As such, steps may be missed, and I may not have a role/permission mapping that is complete.

An example of the above stated methodology is a role-focus approach used to define the job J_1. J_1 steps are defined as a process contained in a single workpattern (See Figure 11.). The steps required for workpattern W_A are S_1, S_2, S_4, S_3, and S_7.

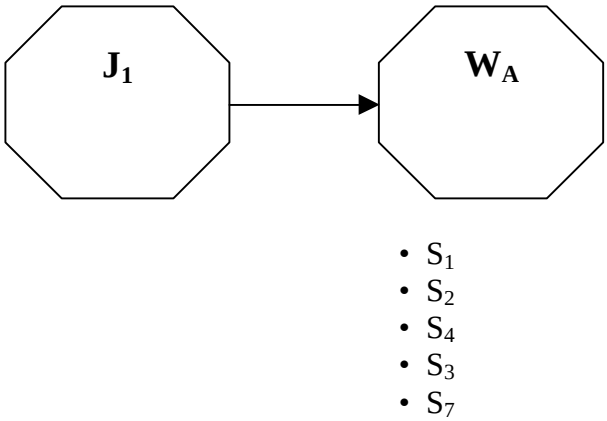

Figure 11: Jobs-to-Workpattern Example

4.3 Workpatterns to Tasks

My next phase is simply to assign each step to a task so that I can lay the "ground work" to assign permissions to a task. Mapping workpatterns to tasks is a continuation of the

previous phase. Earlier, I identified the steps required to perform the work of a job in a workpattern; however, I did not consider minimization or permission assignment. By assigning steps to tasks, I mapped the workpatterns to the tasks. In so doing, I minimize the amount of redundancy and reuse tasks. I begin distinguishing between tasks that are mapped to a permission rather than to tasks that will not be mapped to a permission. Figure 12 shows the mapping from workpatterns to tasks.

There is a many-to-many mapping from workpatterns to tasks. In this phase, steps are assigned to tasks; consequently, the workpattern of the steps are mapped to the assigned tasks.

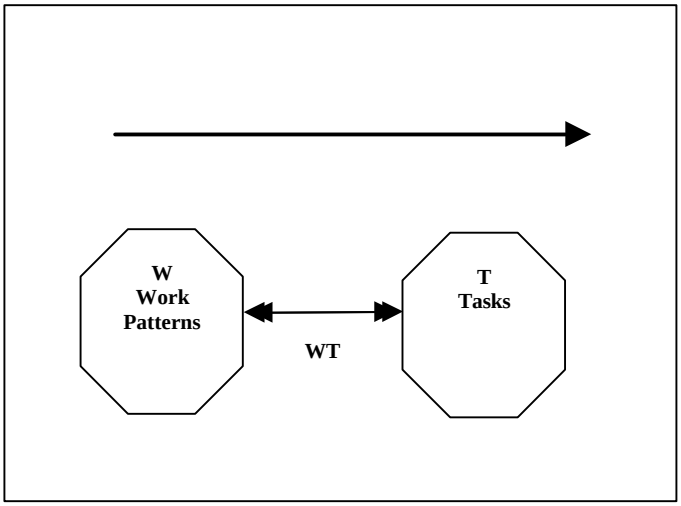

Figure 12: Workpattern to Task Phase

I start considering the concept of permission/permission-free tasks. If it does not map to a permission, I can still maintain the task as a placeholder, so that in the future if there is a permission required by that task, I can map it to that permission.

In the previous section, I identified the steps of a workpattern. For each step of a workpattern, it was assigned to a task. To minimize the number of tasks, I can reuse existing tasks by assigning steps that perform the same work to a pre-existing task.

Figure 13 shows an example of permission equivalent workpatterns. Two workpatterns --Teaching and Teacher Support-- contain a list of steps that are mapped to tasks.

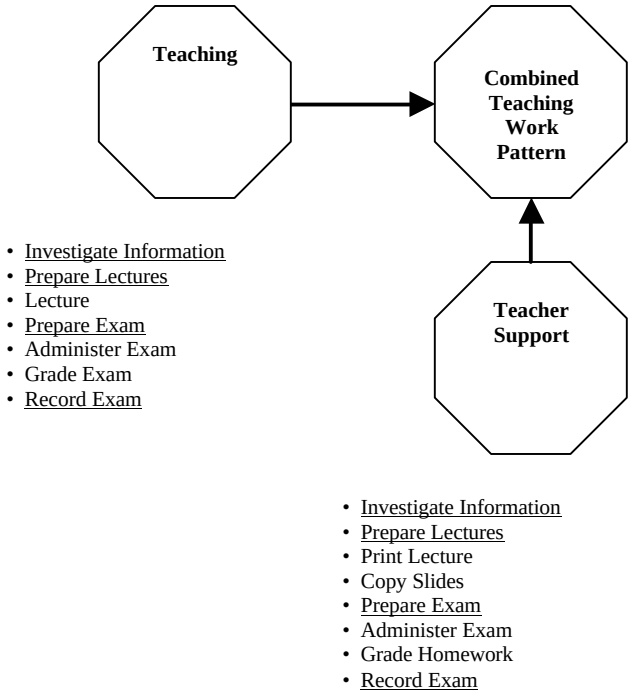

Figure 13: Permission-Equivalent Workpattern Example

After mapping tasks to permissions, I determined that the underlined tasks in Figure 13 require permissions; consequently, the Teaching and Teacher Support workpatterns are permission-equivalent workpatterns. These two workpatterns can be minimized into one workpattern. If at a later time a permission-free task needs to be mapped to permissions, I can split the combined workpatterns into separate workpatterns and map each workpattern to its own job.

One of the benefits of the WT phase is that I started concentrating on permissions in order to minimize the amount of redundancy. In addition, by defining a task that can be mapped to many workpatterns, tasks can be reused.

The steps within the workpatterns detail the work that is accomplished by the workpatterns. Without this phase, the workpattern is mapped to permissions. Each workpattern must be defined in its entirety, without the savings of reusing previously defined sets of permissions. Furthermore, there is benefit to considering the minimization and future growth of permission-equivalent workpatterns. Later, if permissions are added, I only need to re-evaluate the workpatterns, rather than evaluate the tasks.

Figure 14 depicts an example of mapping of workpatterns to tasks.

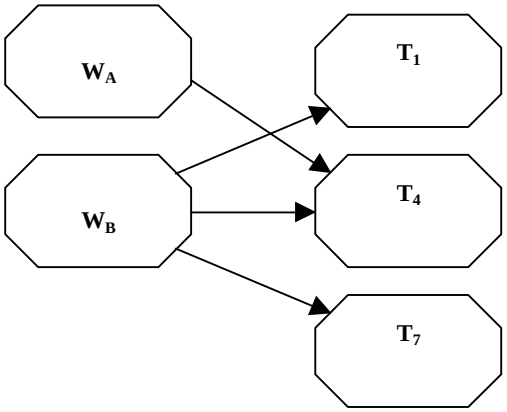

Figure 14: Workpattern-to-Task Phase Example

Workpattern W_A is mapped to task T_4, and workpattern W_B is mapped to tasks T_1, T_4, and T_7. Until the next phase, I do not know the tasks that are permission tasks. If T_1 and T_4 are permission-free tasks, then W_A and W_B are permission-equivalent workpatterns.

4.4 Tasks to Permissions

In the final phase of decomposition, I map the permissions to tasks (See Figure 15). Through the decomposition methodology, I have a clear understanding of the accesses required by the defined tasks. After finishing this phase, I check for completeness of permissions, uniqueness, and minimization of jobs, workpatterns, and tasks.

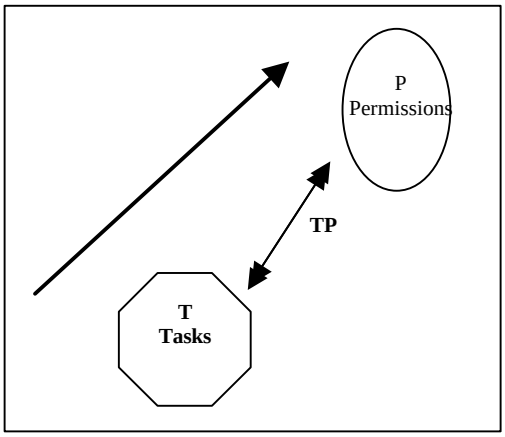

Figure 15: Tasks-to-Permission Layer

By the original assumption, the permissions have been predefined. I want to match the accesses required to perform the work of the task with the accesses granted by the permissions.

As defined by the steps of the workpatterns, I know the work that will be performed by the task. However, if I did not choose the application-focus or permission-focus approach, I may not have defined the purpose of the applications and the permissions that would provide the accesses to each application.

For each task, I decided the permissions the task needs to perform its work. I mapped the task to the permissions that granted the required accesses. If the work of a task does not match a permission, I tried to gather additional information from the previous layers. If I still cannot find a match, I marked the task as permission-free.

Figure 16 shows an example of mapping one of the tasks of the Teaching workpattern from the previous section.

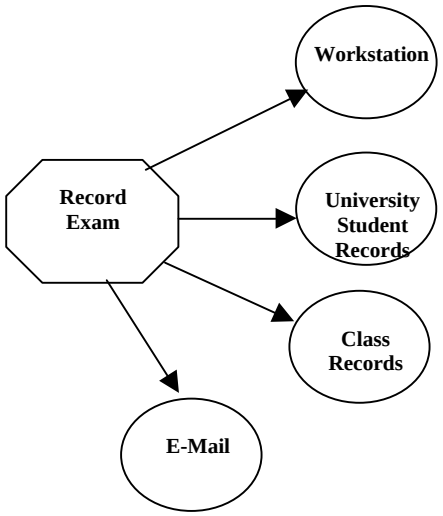

Figure 16: Mapping tasks to permissions

From the Record Exam task, I determined from its history and the available applications that the permissions required to complete the work of the task are: Workstation, University Student Records, Class Records, and E-Mail.

After the mappings, I finished the approach by checking for minimization and completeness. For minimization, I needed to minimize duplicate tasks and workpatterns and then determine if permission-equivalent workpatterns need to be minimized. Once finished, I needed to determine if equivalent jobs should also be minimized.

Next, I check for role-permission completeness by mapping back, from layer to layer, each permission to a role. If I find that there is a permission that is not mapped to a role, per the scope of the dissertation, the best I could do is to recommend that:

- Either another role needs to be created, or
- The responsibilities of a role need to be expanded, or

- The unmapped permission should be deleted.

Otherwise, there are capabilities of the applications that will not be able to be accessed by user.

The benefit of mapping tasks to permissions is that it provides the ability to reuse tasks. The tasks to permissions are defined once and I no longer have to determine the permissions required by the task. If the task exists, I do not need to rethink which tasks need to be assigned; I simply choose from the tasks that are available.

Without this phase, there would be mapping from the workpatterns to the permissions. I would not know which task is a permission task or a permission-free task. Thus, there can be an increase to the number of workpatterns that need to be created. In addition, I do not have the benefits of reusing tasks.

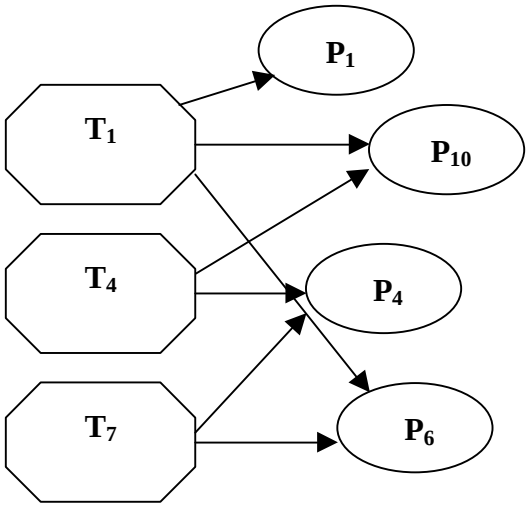

Figure 17: Tasks-to-Permission Phase Example

Figure 17 depicts an example of mapping tasks to permissions. A workpattern maps to three tasks. They are: T_1, which maps to P_1, P_6, and P_{10}; T_4, which maps to P_4 and P_{10}; and T_7, which maps to P_4 and P_6. By inference, the workpattern is mapped to the permissions: P_1, P_6, P_{10}, P_4, P_{10}, P_4 and P_6. There is no benefit of duplicating permissions, so eliminating one of the P_4's, one of the P_6's and one of the P_{10}'s I could minimize the number of permissions. First, by eliminating duplicate permissions, and then by determining the least number of tasks that map to all of the permissions, I could also minimize the number of tasks. In this example, the task, the workpattern, does not need to map to T_4 because the permission P_4 is mapped to by T_7, and T_1 is mapped to by P_{10}.

In summary, the decomposition phases are: Role -> Jobs -> Workpatterns -> Tasks -> Permissions, thus Roles -> Permissions.

4.5 Summary Methodology

A role engineer is responsible for following a role/permission methodology but may need assistance from the administrators and developers. The methodology is a sequence of activities to decompose a role into permissions.

Role to Jobs:

1. Review the roles to learn about the roles and their associated criteria.
2. Choose a decomposition target approach: Role-focus, Application-focus, or Permission-focus; define the attributes.
3. Determine if a role is: Documented, Existing, or Undefined.
4. Identify the jobs of a role, and if possible, reuse jobs.

5. For all jobs not inherited, map a role to those jobs.

Jobs to Workpatterns:

6. For all jobs, map the job to a workpattern.

Perform the next two steps if the workpattern cannot be reused.

7. Determine if the Workpattern is Single process, Multi-process, or Ad-hoc.
8. Create the steps: either by Single process, Multi-process, or Ad-hoc.

Workpatterns to Tasks:

9. Assign each step to a task, and map the step's workpattern to the task.
10. Create a new task, if the task cannot be reused.
11. Validate that all the necessary steps required to perform the work of the job has been identified in the workpattern.

Tasks to Permission:

12. Determine the accesses required by the task**.**
13. Map each task to the permissions.
14. If a task cannot be mapped to an existing permission, then label the task as permission-free.

Minimization:

15. Maintain uniqueness by minimizing equivalent tasks and reusing tasks.
16. Maintain uniqueness by minimizing equivalent workpatterns and reusing workpatterns.

17. Identify permission-equivalent workpatterns, and determine if permission equivalent workpatterns should be minimized. If so, minimize those workpatterns.
18. As desired, minimize equivalent jobs.

Completeness:

19. Map all of the permissions back to the roles.
20. Check for role-permission completeness by ensuring that all pre-defined permissions that have not been mapped to a role have been assigned.

5.0 AGGREGATION

In the previous chapter, I discussed the decomposition of a role into permissions. I started with a role; and through four layers of RPAM01, I decomposed a role into permissions. The reverse is also possible. I can aggregate a pre-defined set of permissions into roles using the framework of RPAM01 (See Figure 18.)

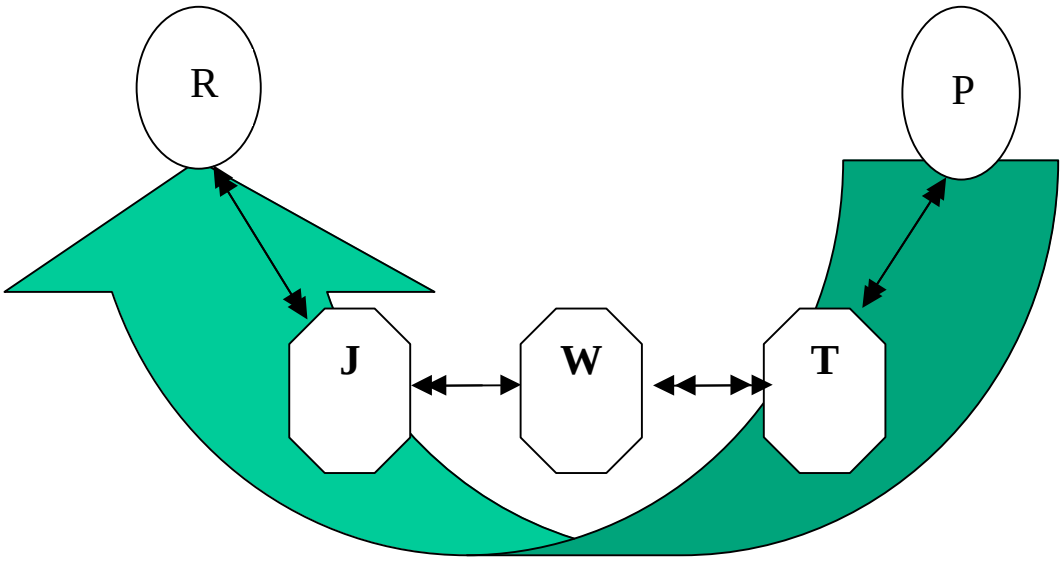

Figure 18: Aggregating Permissions to Roles

As before, there are four phases in this methodology: Permissions-to-Tasks, Tasks-to-Workpatterns, Workpatterns-to-Jobs, and Jobs-to-Roles. Each phase will be discussed in detail in its own section by describing:

- the layer's objectives;
- the issues to be resolved;
- an approach to resolve the issues;
- an example; and
- the consequences of not implementing the layer.

Before I begin, I highlight a few important characteristics of the decomposition approach that are addressed later in the following sections. I observe the following in Figure 19:

- A permission can be mapped to more than one task;
- A task can be mapped to more than one workpattern;
- A workpattern is mapped to (e.g., observe the direction of the arrow for T_3) a permission task before creating the steps of the workpattern;
- A workpattern can only be mapped to (e.g., observe the direction of the arrow for T_2) a permission-free task after creating the steps of workpattern; and
- The many-to-one relation of jobs to workpatterns provided the ability of two different workpatterns to map to the same job.

These highlights are discussed in more detail later in this chapter.

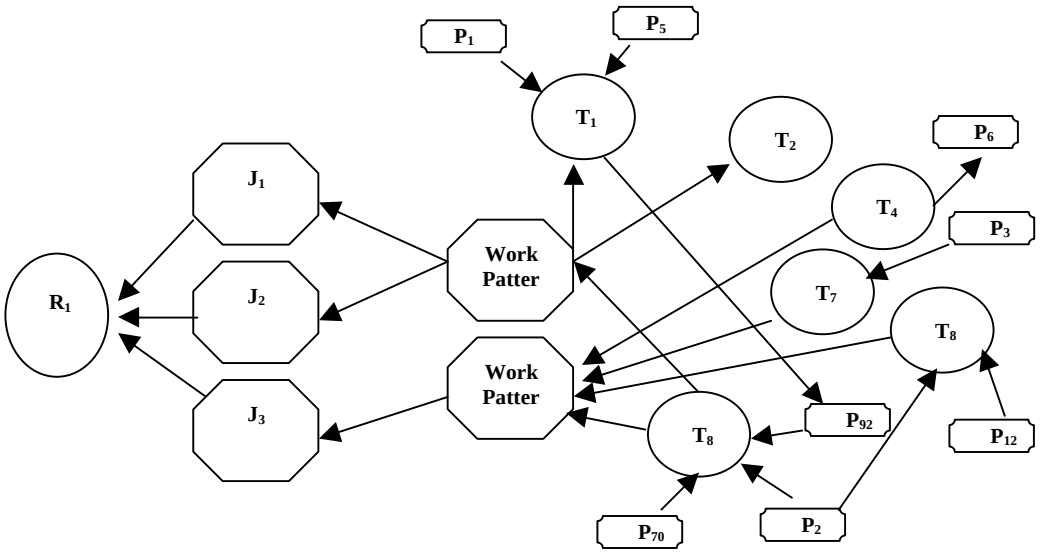

Figure 19: Permissions to Roles Mapping

5.1 Permissions to Tasks

Permission aggregation to roles begins with the assignment of permission to tasks (See Figure 20.). I am given the permissions and I strive towards completeness by mapping all of the permissions to tasks.

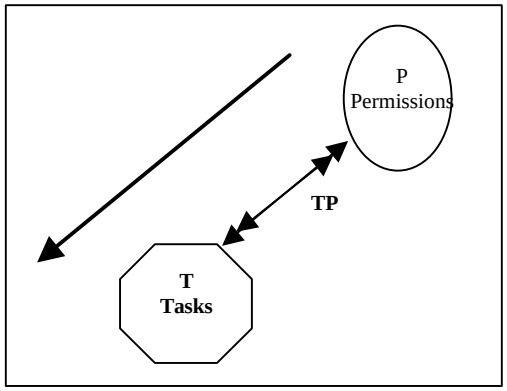

Figure 20: Permissions to Tasks Phase

I consider three approaches to assign permissions to task assignment based on the:

1. attributes of the permissions (Permission-focus),
2. attributes of the applications (Application-focus), and
3. needs of a role (Role-focus).

Starting with the first approach, I group the permissions into a bucket according to similar capabilities that are based on the permission-focus attributes. To ensure completeness, each permission has to be a member of at least one bucket. A bucket that represents multiple tasks must be subdivided into additional buckets. For example, all the data file updates are defined into one group. In the bucket are permissions to update database files, router scripts, and operating systems programs. I further categorize the group by data type. Now I have three buckets.

Using the second approach, I identify the attributes of the application, and then choose the permissions based on the criteria set by the application-focus. I begin by learning about the permissions required to perform the functionality of the application. Then I place all permissions of similar functionality into the same bucket. For example, I may have a bucket that provided a system backup; another bucket that provided the status of applications; and a third that searches the application for a list of software revisions.

To ensure completeness of the permissions, I verify that all of the permissions are placed into at least one bucket. I then check the buckets for permissions that may constitute distinct tasks and I subdivide the distinct tasks into separate buckets. For example, in the bucket for “system backup,” I may have permissions to access the network for a remote backup, permission for backing up the system on tape, and

permission for backing up the system on CD-ROM. I place the network access for a remote backup permission in a separate bucket than the other two permissions.

The final approach, role-focus, may be the most prevalent used approach. It benefits role engineering by concentrating the aggregation decisions based on the final layer, roles.

To perform the final approach, I need to learn the responsibilities of the roles. Following a role-focus approach, I use the attributes to identify the target work that can be accomplished by a role. A bucket is created for each targeted work area. Next, I place each permission into all the buckets requiring that permission, ensuring that all permissions have been placed into at least one bucket. I then review the buckets and subdivide a bucket if it contains more than one unique task.

For example, if I have an attribute of “skill set,” I compare the attributes against my two roles of Professor and System Administrator. I determine from my researched knowledge of the roles that two buckets are needed: one is needed for user account maintenance and another bucket is needed for teaching. After these buckets have been created, each is mapped to only one task.

The benefit of mapping permission to tasks is that I start to group permissions into like accesses. I do not have to continually rethink about the creation of new tasks and their related permission assignment; I can reuse existing tasks.

If I do not have a permission-to-task assignment, I would not have tasks and I map permissions straight to workpatterns. Further, I would not be able to list the workpatterns as a grouping of tasks because I would be mapping disparate permissions to

a workpattern. As such, I no longer have the benefit of using a process flow to validate that all required steps have been identified in the workpattern. Furthermore, I lose the benefits of reusing a previously defined set of permissions in the form of tasks.

Figure 21 illustrates mapping permissions to tasks. There is a blue bucket containing permissions P_1, P_2, and P_6, and red buckets that contain permissions P_2, P_3, P_4, and P_6. The blue bucket is mapped to task T_1 and the red bucket is mapped to task T_2. As I see from Figure 21, P_2 and P_6 exist in both buckets.

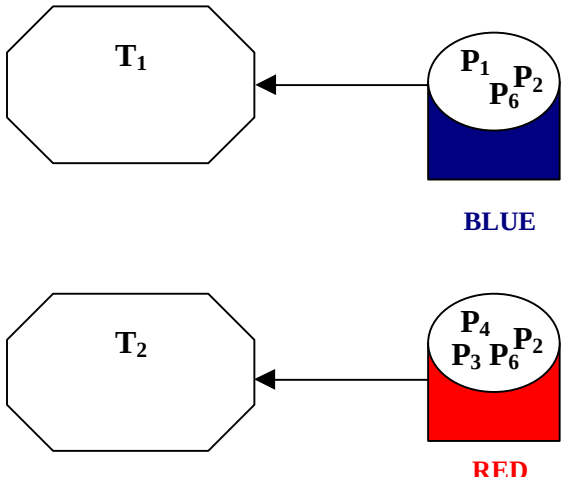

Figure 21: Permission to Task Example

5.2 Tasks to Workpatterns

After defining the tasks, the next step is to map the tasks to workpatterns (See Figure 22.). Before this can occur, the tasks are grouped, based on the focus attributes, in anticipation of the processes that are defined in the next phase. After mapping the tasks-to-workpattern, I validate task completeness.

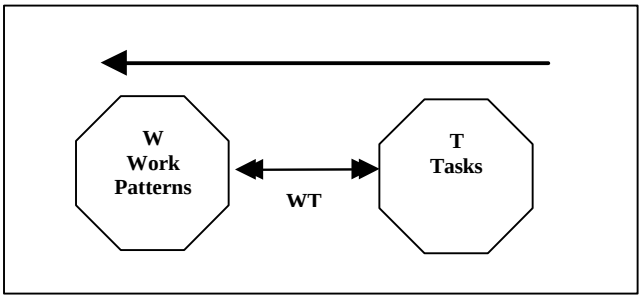

Figure 22: Tasks to Workpattern Phase

Within each of the three approaches stated in the previous section -- Role-focus, Application-focus, and Permission-focus -- I group the tasks into buckets so that they can later be used to define a workpattern. This phase represents the beginning of creating a grouping of work that is finished by a job. Aggregating tasks into jobs requires two passes: the first pass involves identifying the tasks for each workpattern and the second pass ensures that all the tasks are identified, which may include the creation of permission-free tasks.

By the Permission-focus approach, all tasks are grouped into buckets. I begin by reviewing the tasks and identifying the different types of buckets that can be created under the guides of the attributes. Next, I identify the tasks that are placed in each bucket. I remember that one task can be assigned to more than one bucket. After all the tasks are assigned, I check to ensure that all tasks perform an activity that supports the goal of the bucket.

Application-focus studies the applications and creates buckets by functions under the guides of the Application-focus attributes. The approach focuses on the individual

applications as well as the interrelations of more than one application. I may find that a bucket encompasses the work of a system of applications. As with Permission-focus, I identify the tasks that are placed in each bucket and then check to ensure that all tasks that are required to perform the workpattern exist within the bucket.

Role-focus directs the aggregation engineering from both top-down and bottom-up directions and provided a distinct advantage in this phase and in the next phase. I begin by reviewing a role under the guides of the attributes and determine the types of buckets that can finish activities. With this knowledge, I assign tasks to the buckets. I then check to ensure that all buckets have been assigned to a task. After the buckets have been assigned, I map each task to a workpattern and I minimize equivalent tasks.

The benefit of this phase is that it provides me with the ability to start defining the job using reusable tasks. At this point I do not consider completeness or minimization. I provide three approaches that can assist me in determining the logical flow of tasks to roles.

If I do not have this phase, I map tasks straight to jobs, and there will not be an understanding of the purpose or the structure of the tasks that are used. I cannot use the benefits of a process to assist in ensuring that all the necessary permissions that are required by a job have been identified, thus there may not be completeness.

Figure 23 illustrates mapping tasks to workpatterns. There is a blue bucket containing tasks T_1, T_2, and $T_{6;}$ red buckets that contain tasks T_2, T_3, T_4, and T_6; and a green bucket that contains the exact same tasks as the blue bucket. The red bucket is mapped to task $W_{7;}$ the blue bucket is mapped to task W_1; and the green bucket is mapped

to W_4. As I can see Figure 23, W_1 and W_4 are equivalent workpatterns; but they are not minimized in this layer.

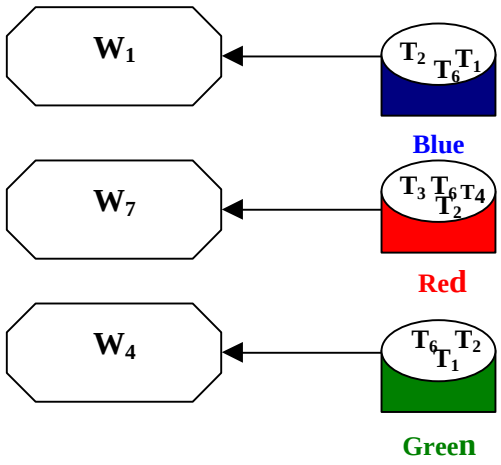

Figure 23: Tasks to Workpattern Example

5.3 Workpatterns to Jobs

Now I map workpatterns to jobs (See Figure 24.). In the other phases, there could be aggregation of elements from the previous layer into elements of the current layer (i.e., many permissions can comprise one task and many tasks can comprise one workpattern).

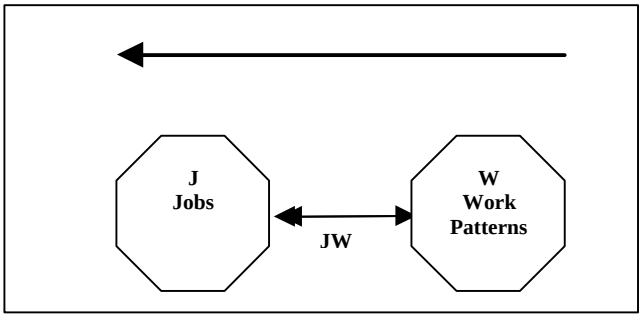

Figure 24: Workpatterns to Jobs Phase

In this section, I need to: 1) identify the steps that may be needed to identify the work of a job, 2) decide if there is completeness, and 3) decide if a permission-free task needs to be mapped to the workpattern.

I begin the aggregation by identifying the workpattern steps. I use the three guides stated in Section 4.2: single process, member of another process, or ad-hoc, as a support tool to find missing steps.

The decisions to engineer the steps are guided by the focus approaches: Role-focus, Application-focus, and Permission-focus. For each approach, the procedures are similar. First, I decide to create the workpattern into one of the three guides. Once accomplished, I start with the single process workpattern. The tasks are placed in a sequence based on the focus of the approach. Each task is assigned to a step. If there appears to be a missing step, I determine if there is an existing permission task that should be assigned to that step; if not, I add a permission task by the methodology that I previously defined in Sections 4.3 and 4.4. A step that cannot map to a permission is assigned to a permission-free task. In all cases, the task is mapped to the workpattern.

For the workpatterns that are members of a process outside of their workpattern, I determine if there is a sequence that relates the steps of all of the workpatterns. From the sequence, I determine if there are any missing steps and then the workpattern will contain the missing steps. If needed, I will create a new workpattern. For each missing step, I determine if there is an existing permission task that should be assigned to that step; if not, I add a permission task by the methodology that I previously defined in Sections 4.3 and 4.4. A step that cannot map to a permission is assigned to a permission-free task.

Alternatively, if a multi-workpattern process has not been predefined, I may be able to engineer a process that is contrived only of workpatterns, since I have no knowledge of the steps that are not contained in a workpattern. I can review the workpatterns and attempt to create a process from a sequence of tasks that are mapped to the workpatterns. I assign each task to a step. After sequencing the tasks of the workpatterns, I determine if the workpatterns are missing steps. If they are, I follow the approach stated in the first option for missing steps.

There is neither additional engineering nor steps required for an ad-hoc workpattern. The tasks assigned to these steps were created in the previous layer. I simply map the ad-hoc workpattern to the job. There is no need to create Permission-free tasks.

Equivalent workpatterns can be minimized in this phase; however, permission equivalent workpatterns should not be minimized without considering the work requirements of the role. I have already determined that all the necessary steps required to perform the work of the job have been identified in the workpattern. Next, I need to map the workpatterns to the jobs. A role may dictate the need to map one workpattern to many jobs. For Role-focus, I may be able decide if I need to map a workpattern to more than one job or minimize permission-equivalent workpatterns.

The benefit of this section, workpatterns-to-jobs, is that it provided me with a tool to determine if the job has all the permission it requires to perform its work. I can also minimize equivalent/permission-equivalent workpatterns and I now have the ability to reuse a workpattern that may be required by more than one job. Finally, the ad-hoc

workpattern can be used as a set of tasks that map to unassigned permissions so that there is role-permission completeness.

If I do not have this phase, I map workpatterns straight to roles without knowing if a role has all the permissions required to finish its work. Figure 25 illustrates mapping workpatterns-to-jobs. The first example shows a straight mapping of workpattern W_C to job J_5; however, in the next phase, I determine that it is necessary to map two roles to distinct jobs. I have role R_1 mapped to job J_6 and role R_2 mapping to job J_5, both of which map to workpattern W_C. The dash lines indicate the mapping that is created in the next phase.

In the second example, where workpattern W_B is mapped to job J_3, I realize that, after checking the steps, there is an additional step that requires the creation of a new permission task T_8.

The final example starts with two distinct workpatterns, W_A and W_D. After checking the steps, I see that there are additional steps that require the creation of a new, permission-free task T_2 and the need to reuse the permission task T_5. After performing the new mapping, I realize that workpatterns W_A and W_D are now permission equivalent. In the next phase, I may minimize these workpatterns.

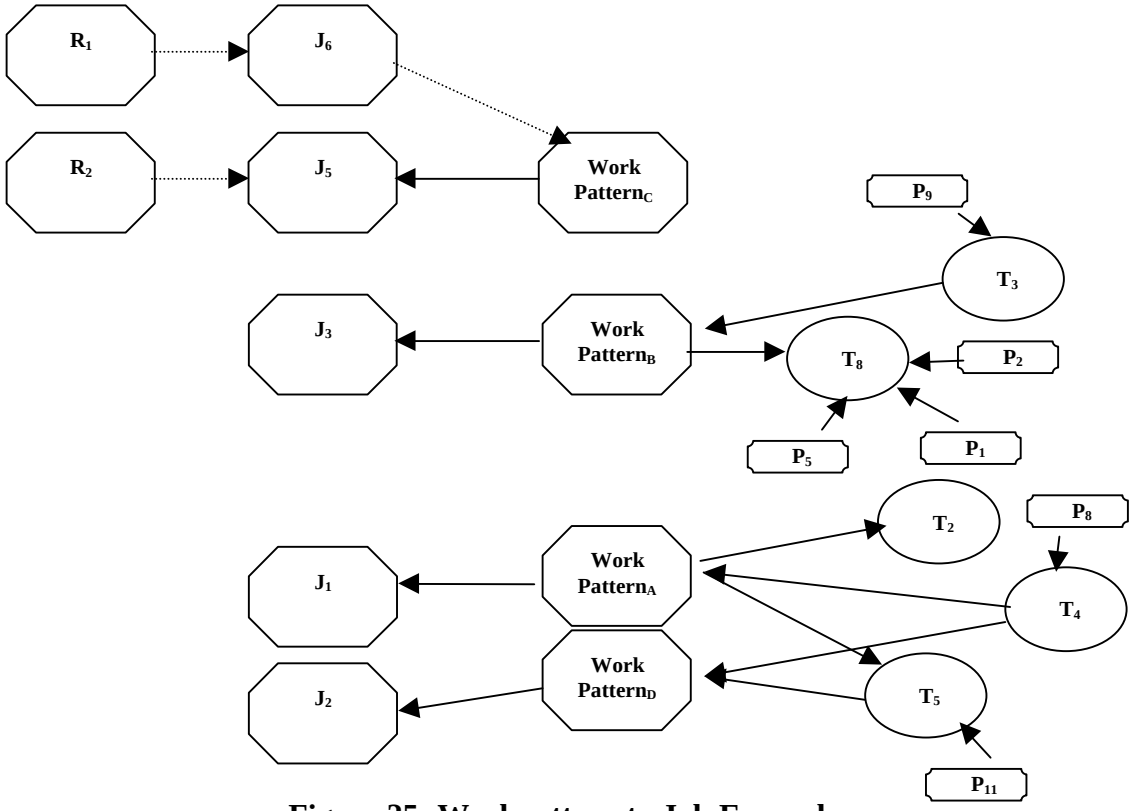

Figure 25: Workpattern to Job Example

5.4 Jobs to Roles

In the final phase, I define roles, map the jobs to roles, and consider minimizing permission equivalent workpatterns (See Figure 26.). Unlike the prior phases where I defined the elements of the next higher layer, I now map from a known job to a known role.

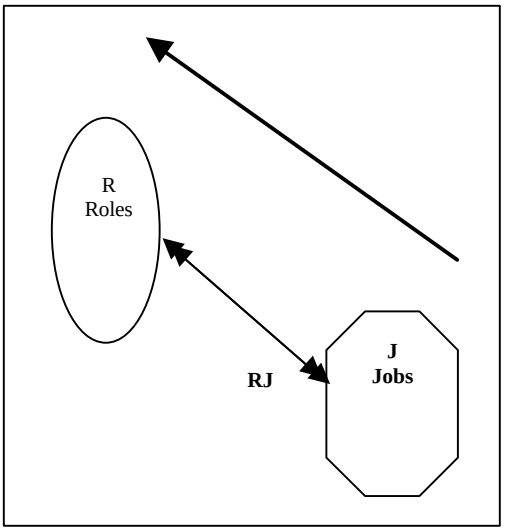

Figure 26: Jobs to Roles Phase

Before jobs can be assigned to roles, I need to learn about the work that is performed by a role. As in Section 4.1, I can categorize roles into three groups (documented, existing, and undefined roles); and then I determine their responsibilities within the scope of the chosen focus approach.

After roles have been defined, I can map jobs to roles. For each job, I compare the job's capabilities against a role's responsibilities, and determine the best fit of each job to a role. Once found, I map the job to a role. One job can be assigned to more than one role, unless the organization desires a unique job for a role. In that case, I do not reuse a job; but I create a new job and map it to the same workpattern as the previously mapped job. For any role not mapped to a job, I recommend to the organization administrator to eliminate that role or to define an additional role's responsibilities so that the role can be mapped to a job.

After the aggregation of permissions to roles, I determine, from the organization administrator, if there is a reason to minimize jobs that map to permission equivalent workpatterns. If there is a reason to keep these jobs (i.e., anticipate future mapping of future permissions to permission-free tasks), then no change is required. If jobs must be minimized, then I need to join the tasks that map to all workpatterns and map them to a single workpattern. Next I map this workpattern to the job, which, in turn, will be mapped to a role.

To finalize the aggregation, I need to check for role-permission completeness. Although all of the jobs have not been mapped, it does not necessarily mean that there is not role-permission completeness. I map back each role to permissions and determine if all permissions have been mapped to a role. After checking for completeness, I find that there is a permission that must be mapped to a role and a role does not exist. Per the scope of the dissertation, I recommend that either more roles be created to be decomposed to these unaggregated permissions or that the responsibilities of a role be expanded so that the permissions can be aggregated to the new jobs that have been formed from the new responsibilities. Otherwise, there are capabilities of the applications that will not be accessible to any user.

The benefit of this phase is that I assign roles to permissions and I potentially reuse jobs. In addition, I have checked role-permission completeness; and if necessary, I have recommended to the role engineer to either add more roles or expand the existing responsibilities of the roles.

If I do not map jobs to roles, then I map jobs to users, which defeats the purpose of using roles in the RBAC Model.

Figure 27 illustrates mapping jobs-to-roles. The first example depicts a straight mapping of job J_5 to role R_2. The second example depicts the mappings of job J_5 to roles R_2 and R_1; however, the organizations desire unique jobs so job J_5 is mapped to role R_2 and job J_6 is mapped to role R_1. In the final example, workpatterns W_C and W_D are permission equivalent. The organization wants to minimize the jobs J_5 and J_6, so the unique tasks mapping to workpatterns W_C and W_D are minimized to workpattern W_F. W_F maps to a new job J_7; and, in turn, job J_7 is mapped to roles R_1 and R_2.

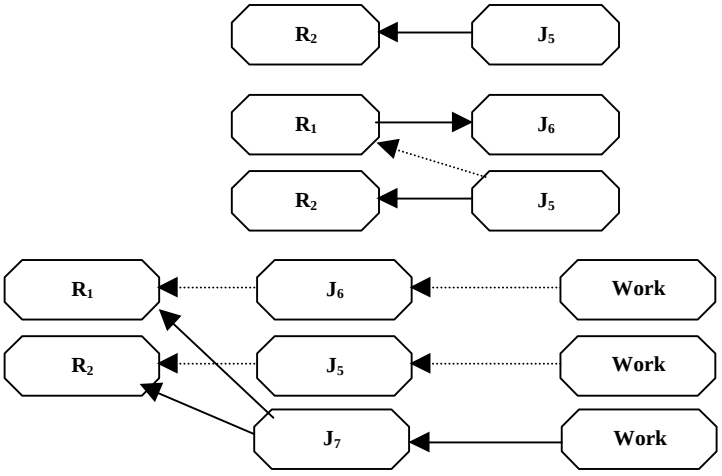

Figure 27: Jobs to Roles Phase Example

In summary, the aggregation phases are: Permissions -> Tasks -> Workpatterns -> Jobs -> Roles, thus Permissions -> Roles.

5.5 Summary Methodology

A role engineer is responsible for following a role/permission methodology, but may need assistance from the administrators and developers. The methodology is a sequence of activities to aggregate permissions to roles.

Permission to Tasks:

1. Review the permissions to learn about the applications and their associated authorizations; choose an aggregation target methodology: role-focus, application-focus, or permission-focus; and define the attributes.
2. Group permissions into buckets that can be assigned to a task.
3. Map the permissions to tasks (Note: If all permissions have not been mapped, then there is no completeness.).

Tasks to Workpatterns:

4. Group the tasks into potential workpatterns.
5. Map the tasks to a workpattern (Note: If all tasks have not been mapped, there may not be role-permission completeness.).

Workpatterns to Jobs:

6. Determine if the workpattern is: Single Process, Multi-process, or Ad-hoc.
7. Create a logical sequence of the tasks as a set of steps in sequence to check if all the necessary steps required to perform the work of the job have been identified in the workpattern using Single Process, Multi-process, or Ad-hoc.
8. Check Workpattern completeness.

9. From the set of tasks for the workpattern, assign a task to each step. If a step is not assigned to an existing task, create a new task; if the new task cannot be mapped to at least one permission, then it is a permission-free task.
10. Minimized equivalent workpatterns.
11. Map each workpattern to a new job.

Jobs to Roles:

12. For each role, determine if it is: Documented, Existing, or Undefined.
13. Identify a role that should perform the work of each job.
14. Determine if two jobs that map to permission equivalent workpatterns should be minimized. If so, minimize the workpatterns.
15. Make sure each role is assigned to at least one job.
16. Notify the organization administrator of unassigned roles.

Role-permission completeness:

17. Create a set of permissions by mapping roles back to permissions.
18. Compare role permissions against the pre-defined set of permission.
19. Ensure that all pre-defined permissions that have not been mapped to a role have not been assigned.

6.0 CASE STUDIES

I have developed but have not yet validated the theoretical approach for decomposition and aggregation of roles and permissions. To do so, I analyze other approaches and compare them to RPAM01. Earlier, I identified three related models. I will analyze the first and second models. They are most closely related to the two approaches, decomposition and aggregation. The first article, "Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organization" creates a process for defining roles and is related to the decomposition approach. The second article, "Napoleon Network Application Policy Environment," defines policies; this approach depicts concepts similar to the aggregation approach. The third article, "Access Control for a Healthcare Information System", is also a decomposition approach and could be analyzed similar to the Role-Finding Model.

For each approach, I perform comparative analysis between the approaches of RPAM01 and this dissertation. Intertwined within the comparative analysis, I provide the following types of information:

- Purpose: a statement on why the model was created and what the objectives are for using this approach;
- Results: a listing of the capabilities that was produced;
- Approach: a high-level description of the model's architecture;

- Methodology: a summary of the logical sequence of the approach;
- Model Benefit: the advantages that the proposed model has over RPAM01;RPAM01 Benefit: the advantages that RPAM01 has over the proposed model in six areas: Layers, Properties, Target based Decisions, Role Identification, Workpattern Decomposition, and Workpattern aggregation; and
- Conclusion: Summary of the above stated information and the benefits of adding the stated dissertation approach to the proposed methodologies.

Using this information, I analyze the models, the Process Oriented Approach and the Napoleon, in Section 6.1 and Section 6.2.

6.1 Process-Oriented Approach for Role-Finding to implement Role-Based Security Administration in a Large Industrial Organization

RBAC was considered as a response to the need to centrally administer security accesses for a trusted network across platforms. The challenge was not in the integration of a trusted network and a single point of administration, it was with the identification of roles. To solve this challenge, Roeckle, Schimpf, and Weidinger, proposed a process-oriented approach for role finding.

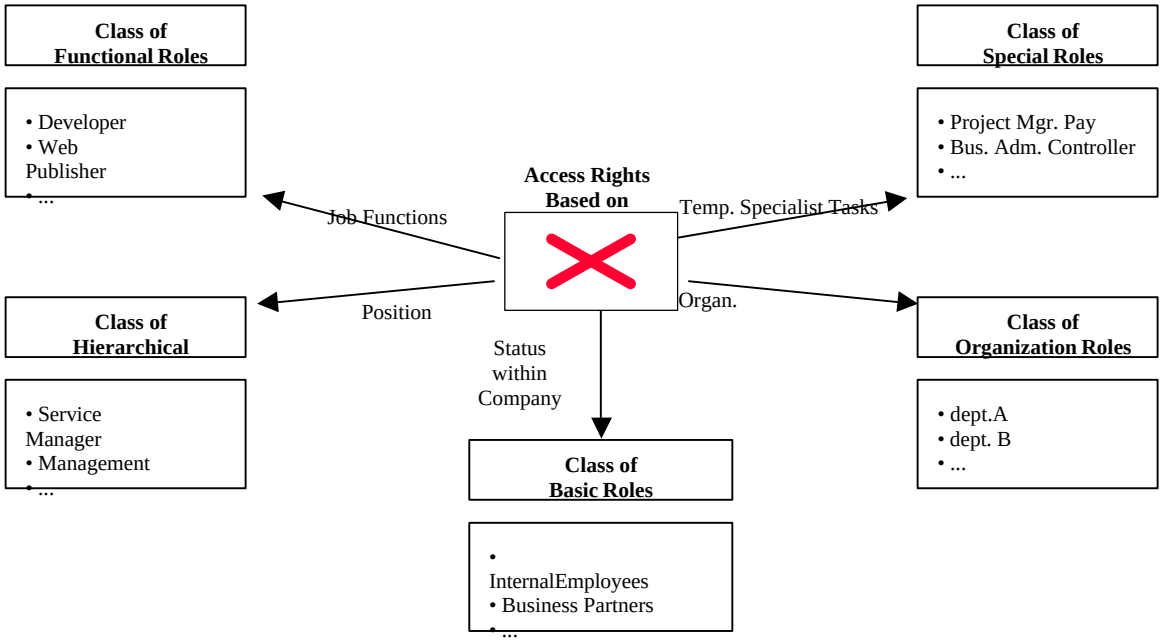

Figure 28: Classification of Roles

Initially, Roeckle, Schimpf, and Weidinger studied the classification of roles that existed within their company (See Figure 28). Access rights required by individuals were based on job function, temporary specialist tasks, position, status within the company, and organization association. Each of these roles can be categorized into five role classes: Functional, Special, Hierarchical, Basic, and Organizational. The paper states that "basic role," "hierarchical role," and "organizational roles" were easily determined and the definition of these roles was not discussed. The fourth role category, "special role," is a group of temporary roles that are defined when needed. The remaining role, "functional," is too complex to readily define. The approach, "Role-Finding," was

researched as a solution for identifying functional roles. A Role-Finding Model is created from three areas: Security Administration, Meta Model, and Procedural Model.

Three subprocesses perform Security Administration: User Administration, Role Administration, and System Administration. Whereas, User Administration is the assignment of users to roles, Role Administration is the identification and implementation of roles, and System Administration is the creation and maintenance of application permissions. A process-oriented approach is then used to show the flow between the three administrators (See Figure 29).

Each administrator starts by defining basic information. For the user administrator, it is users; for a role administrator, it is roles; and for the system administrator, it is permissions. Once roles are created and the required rights identified, the system administrator provides bundles of rights. A role administrator creates a role with the bundle of rights. The roles are then available for the user administrator to assign users to roles.

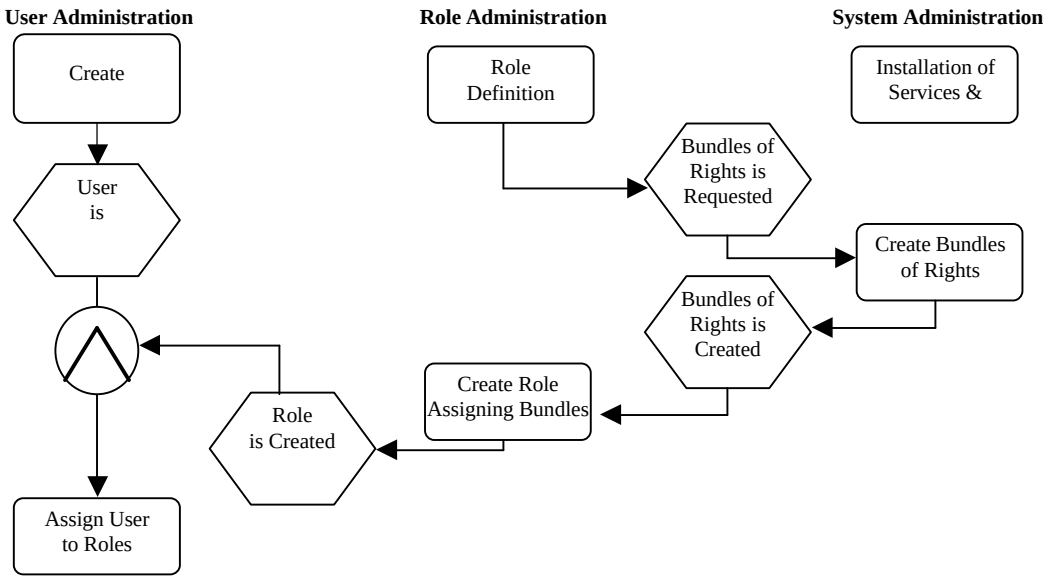

Figure 29: The Process of Security Administration

The process-oriented approach integrates the Meta Model with the procedural model and uses three views as an interface to a process model (See Figure 30).

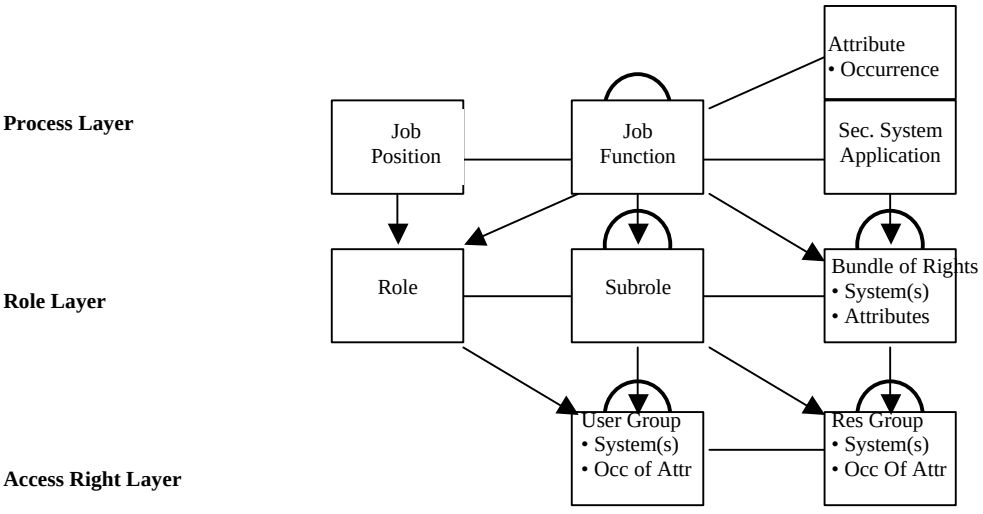

Figure 30: Meta Model for Process-Oriented Role-Finding

The process layer is the interface to a business process model. A role layer is a central location for the cross-platform business roles, and the access layer is the organization of the elements that permits the execution of the business roles.

The approach permits a role layer to be inferred from the process layer and the access layer from a role layer. Using the process layer "entities" as "job function," "job position," "organization unit," "information system," "security system," and "attributes," I can determine the job functions and their assignment to job positions. From this, I can deduce a role. As seen from Figure 30, roles can be derived from job positions and a function; subroles can be derived from job function and bundles of rights can be obtained from a job function. This leads to determining the needed access rights.

Finally, the procedural layer is where the procedures and steps are identified. These procedures are used to define a role layer from the process layer, and then the access layer from a role layer. Using these procedures on a business process model, the administrators should be able to complete the Meta Model. This allows for the definition of a role and their related permission assignments.

I analyze a Role-Finding Model against RPAM01 discussed in this dissertation. First, I discuss the additional capability that is provided by the model. To do this, I compare the key components of a Role-Finding Model against RPAM01. Next, I compare the key components of RPAM01 and its layers against a Role-Finding Model. The key components of a Role-Finding Model that I consider are:

- Cross-platform approach,
- Classification of Roles,

- Decomposition,
- Layers (Process, Role, Access Rights),
- Workflow, and
- Administration (user, role, system).

A Role-Finding Model was intended to create roles that contain access for applications that execute across different types of platforms. The dissertation approach is also platform independent.

Classifying roles identifies categories of roles that may need to be defined. Roeckle determined that only one category of roles needed a special finding process (e.g., Role-Finding). RPAM01 does not distinguish between the different types of roles when decomposing roles.

A Role-Finding Model is more than just finding a role from the functional roles. It also defined the access rights for a role. This model decomposes functional roles by starting with the Process Layer and assigns rights at the Access Rights Layer, whereas RPAM01 starts with the predefined roles, and then determines if a role's jobs can be defined as a process. If a role cannot be defined as a process, then it can be defined as an ad-hoc set of steps, even though a Role-Finding Model does not have a concept of an ad-hoc set of steps.

A Role-Finding Model offers the benefit of an algorithm for finding role (See Figure 30). The algorithm and constraints on the relations are considered proprietary and not available for analysis.

The purpose of the Procedural Layer is to define the steps of the Process Layer. The model suggests six steps for creating the Process Layer to achieve the benefits of RPAM01; however, the details for creating this process are not publicly available at this time.

Another benefit of a Role-Finding Model is that it provided three types of administrators that are responsible for administering security: the user administrator, role administrator, and system administrator. The process allows me to correlate users, permissions, and roles. RPAM01 has a similar concept where an administrator assigns a role to a user. An application developer defined the permissions; and a role engineer, with the aid of the organization, decomposes a role.

To further analyze a Role-Finding Model, I learn how RPAM01 benefits a Role-Finding Model. The areas of interest are:

- Approaches: Decomposition, Aggregation;
- Model Layers;
- Focus Approach; and
- Properties.

A Role-Finding Model presents a compelling decomposition approach for a specific type of roles known as functional roles. As stated earlier, a Role-Finding Model starts with a process prior to defining a role, which can be a precursor for RPAM01. During role definition, the other four types of roles become a part of a process. Later, this process information can be used during the definition of workpatterns.

A Role-Finding Model does not have an aggregation approach. With further thought, and following the "bucketing" guidelines stated in RPAM01, I use a Role-Finding Model to map out an aggregation approach.

RPAM01 contains five layers, whereas the Meta Model for Role-Finding has three categories of layers. The comparison of the layers can be seen in Table 8.

Table 8: Layers: Role-Finding vs. Role/Permission

RPAM01	**Meta Model for Role-Finding**
▪ Workpatterns ▪ Jobs	Process Layer
▪ Roles ▪ Tasks	Role Layer
▪ Permissions	Access Rights

A Role-Finding Meta Model starts with the process layer, which contains a job and its function. The actual steps are defined by another model, which is the procedural model. The procedural model yields the practical steps of the process layer. These practical steps are similar to the workpattern and its steps. The next layer, a role layer, has the concept of roles, subroles, and a bundle of rights. A role and a subrole can be defined within a role hierarchy.

There is a similarity between how a process is defined between both approaches. Analogous to bundles of rights that can be mapped to a practical step or procedure, which creates the process layer, I can consider the tasks as the bundle of rights that, in turn, can be mapped to the steps of the workpattern.

The dissertation concept of focus can provide a benefit to the Role-Finding Model by concentrating on the procedural model step and procedural definition towards the creation of the process layer. The process definition can be used for a role or it can be used to administer multiple applications, or to restrict all application(s) permissions to a specific set of roles that contain a security clearance.

A Role-Finding Model can also benefit from the properties of: Equivalence, Uniqueness, Minimization, Reuse, and Completeness. Equivalence, uniqueness, and minimization, working in conjunction with reuse, can reduce redundancies when recreating the same jobs, job functions, subroles, bundles of rights, and role assigning bundles. In addition, completeness can ensure that all the necessary rights have been assigned to roles so that all required work can be accomplished. The functional roles may not identify all of these rights. Functional roles may need to be combined with the other four categories of rights. Without having a methodology to check completeness, I may miss needed rights.

In summary, a Role-Finding Model can provide an excellent method for finding roles from processes. It can be improved by considering the advantages of aggregation using bucketing, process alternatives (i.e., ad-hoc), focus, and properties. RPAM01 can be improved by considering the classification of roles when evaluating decomposing roles. Finally, I may consider deriving roles from the process and remember this information as part of the focus approach when I consider the creation of workpatterns.

6.2 Napoleon Network Application Policy Environment & Role Based Access Control Framework for Network Enterprises

The Napoleon Model created by D. Thomsen, R. O'Brien, and C. Payne was initially introduced as seven layers. Under the original Napoleon Model, the first four layers -- Objects, Object Handles, Application Constraints, and Application Keys -- were administered by the Application Developer; and the top three layers -- Enterprise Keys, Key Chains, and Enterprise Constraints -- were administered by the Local System Administrator.

Through additional research, the Napoleon authors found that there was a need to support workflows and capture security policies associated with suites of application. This led to the extension of the Napoleon Model by inserting a Semantic Policy Layer between the Local Policy and Application Policy Layers (See Figure 31.). The figure shows three groups of layers: the Application Layer, Semantic Layer, and Local Layer. The bottom layer, the application layer, interfaces with the permissions of the applications. There are fewer changes (more static) to these interfaces as opposed to the Local Policy Layer that interfaces with the users. The dynamic nature of assigning users to the key chain occurs more often then the addition of new application accesses.

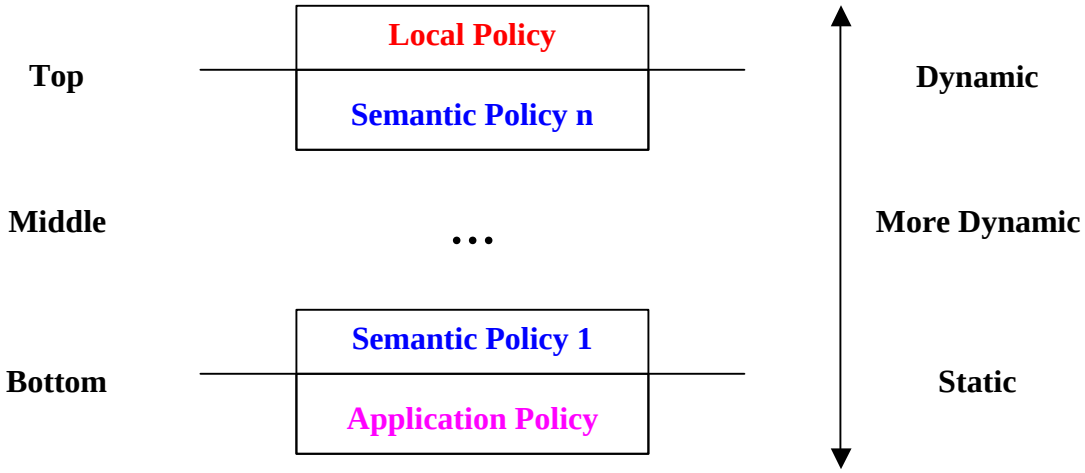

Figure 31: Revised Napoleon Model showing the general trend from static application policies to dynamic local policies

The revised model is a policy approach, where the policy dictates how permissions are assigned to users. Many of the original concepts are used in the revised model. However, unlike the original Napoleon Model, there can be any number of users, who can be administrators by creating a policy within a user's own semantic layer. The original model does not directly map into the new model. The concept of objects, handles, key, key chains, and constraints still exists in each layer[5]. The application policy layer still interfaces with the applications and the local policy layer still binds users to keys.

[5] The following are the definitions of these terms[TOB98]:

- An object is an abstract description of data within the system.
- The object handle captures the way the object is used.
- A key is an abstract representation of some rights.
- A key chain is a collection of keys.
- Constraints are used to capture policy information that cannot be represented as sets.

The addition of the semantic layers permits users, other than the administrator and developer, to define policy. Using the Napoleon Model, semantic policies need not be layered. The required semantic policy is applied to the application policy. For example, the application developer creates a clipboard policy and the architect can state the platforms that can use the clipboard policy.

Figure 32 shows the key chain and how the constraints restrict the ability of the key chain. A key chain can be considered a key at the next higher level. One key may contain multiple key chains, whereas each key chain has its own applied constraints.

The top most key chain points to a local policy and the local administrator assigns the key chain to a user. The user can perform the accesses defined by the objects under the restrictions dictated by the constraints.

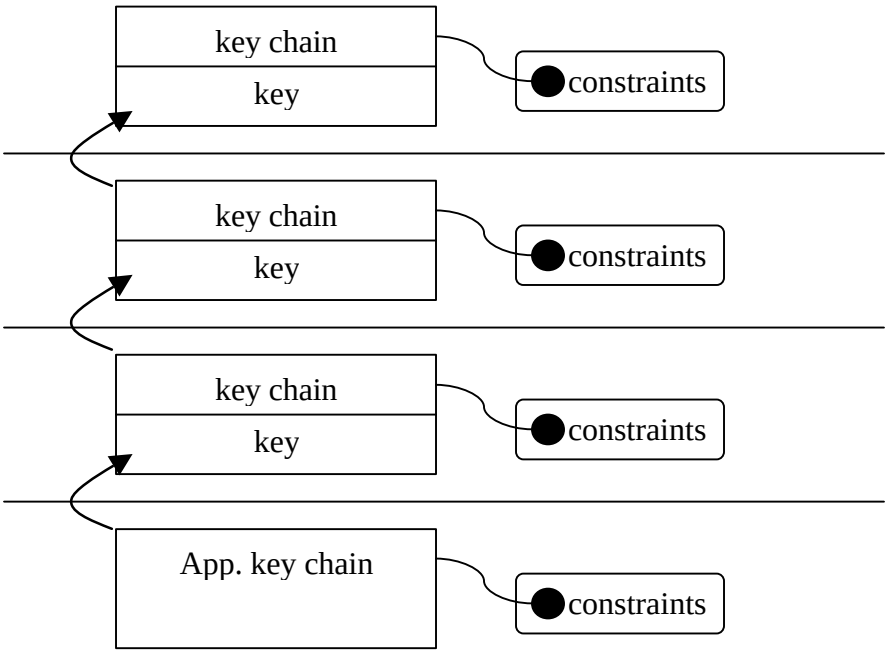

Figure 32: Interface between Semantic Layers

To analyze the Napoleon Model against RPAM01 discussed in this dissertation, first, I discuss the additional capability that is provided by the model. To do this, I compare the key components of the Napoleon Model against RPAM01. Next, I compare the key components of RPAM01 and its layers against the Napoleon Model. The key components of the Napoleon Model that I consider are:

- Aggregation,
- Policy,
- Workflow,
- Permission Structure,
- Constraints, and
- Administration.

Using the terminology from this dissertation, the Napoleon Model is considered an aggregation approach. The application policy contains keys that are an aggregation of objects into handles and handles into keys. The next subsequent semantic layer in the Napoleon Model, which is not necessarily contiguous, forms a Partially Ordered Set (POSET) of key chains, which are assigned to users at the top local layer. This approach covers the major components that are needed to aggregate permissions to assign them to users. Both approaches conclude that there is a need to use a process that is derived from a policy to assign permissions to users.

The model discussed policy as a means for defining the steps of a workflow. The Napoleon Model article does not provide details on how the steps are created nor does it provide specifics on how to create policy. I know at a high level that there are three types

of policies: application, semantic, and local. From this, I can determine the components that can be used to define the process; but I do not have details. This dissertation paper uses a process as an aid to accurately define the permissions that users need to perform their work.

Furthermore, the semantic layers are used as an alternative for role hierarchy. While role hierarchy is out of the scope of this dissertation, the summary solutions point towards role hierarchy as part of the decomposition/aggregation approaches. The main purpose of using a role hierarchy in this dissertation is to ensure that all roles within the hierarchy have been considered. If necessary, instead of using the iterative process to consider each role, I can also use the iterative process to consider each key.

The constraints that control policy manipulate user accesses. Constraints are a tag to key chains, which can improve the control of accesses. This dissertation does not include constraints; however, if constraints are used, they can be tagged to roles, jobs, tasks, or permissions. Job, task, or permission constraints control accesses differently. A constraint on a job can restrict when a job can be performed (i.e., from 6:00 a.m. to 12:00 p.m.) or a constraint on a task can restrict how a user can access an application (e.g., allow only a browser with a 128-bit encryption to access a server). Constraints on permissions can restrict a user's view of data. Note that while a user has permission to access to an application, the user may not have the ability to see another employee's records.

Another benefit of the Napoleon Model is its ability to decentralize administration to the organization, which creates the policy. In addition to the developer and the local

administrator, there can be many organizations involved in creating policy. In the approach of this dissertation, I discuss three groups that help a role engineer architect a role-to-permission relation:

1. Organization that provided the information for a role and if there should exist permission equivalent workpatterns**;**
2. Developer that defined the permissions; and
3. Administrator that assigns roles to the users.

A role engineer is solely responsible for performing the approaches defined in this dissertation with the support of the groups stated earlier. The dissertation does not explicitly state that there are policies, nor does it constrain the capabilities of the components of RPAM01. However, inherent within the approaches, policy is dictated by the final role permission assignments. The people involved in performing the approaches are representative of the groups discussed earlier.

To further analyze the Napoleon Model, I look at how RPAM01 can benefit the Napoleon Model. The areas of interest are:

- Approaches: Decomposition, Aggregation;
- Model Layers;
- Focus Approach; and
- Properties.

The Napoleon Model does not have a decomposition approach. Instead, the Napoleon Model provided the building blocks for the local administrator to determine the

keys that are required by a role. The model does not consider a given set of roles or the permissions that are required to perform the work.

RPAM01 contains five layers, whereas the Napoleon Model has three layers. The comparison of the layers can be seen in Table 9. On the left side of the table are the layers of RPAM01, which are aligned to the layers of the Napoleon Model.

Table 9: Layers: Napoleon vs. Role/Permission

RPAM01	**Napoleon Model**
▪ Roles	Local Policy Layer
▪ Jobs ▪ Workpatterns ▪ Tasks	Semantic Layer
▪ Tasks ▪ Permissions	Application Layer

Under the Napoleon Model there is no concept of a job. I consider that jobs are the responsibilities of a role and that the closest related concept is a key chain in the semantic layer. A job can be considered as a set of keys, whereby a role may need to perform multiple jobs to complete the work of roles. As such, a role requires multiple key chains. The semantic layer is where processes are defined and is similar to workpatterns that can be defined as a process. In RPAM01, I assign steps to tasks so that I can consider properties. Tasks can also be related to the grouping of permissions similar to object handles. Permissions are defined in both models by the application developer.

The dissertation concept of focus can be considered a policy. The concept of focus was introduced to assist the administrator in engineering a role by directing his/her decision based on known attributes of a target element. Decisions are based on a major architectural component of a role/permission assignment (e.g., a role, the application, or the permissions). The stimulus for the decisions is not discussed; however, policy is the end result of the decisions.

Another valuable concept introduced in the dissertation is properties. The Napoleon Model can benefit from the properties of: Equivalence, Uniqueness, Minimization, Equivalence, Reuse, and Completeness. While the benefit of reuse may exist, there is no discussion of the concept in the Napoleon Model. Reuse reduces the amount of overhead involved in recreating the same policy. In addition, reuse can establish a standard for the multitude of users that are creating policy. With the number of users creating the policy, there is a high possibility of inefficiency. Equivalent policies can be minimization, possibly to a point of uniqueness. Policies may be reused, rather than new ones created. Finally, completeness helps ensure that all the required permissions are available to users. There is less of a chance that a required permission has not been assigned, ensuring that a user cannot have access to a critical application.

In summary, the Napoleon Model provided policy, multi-purpose administration, role hierarchy alternatives, and constraints. The dissertation can provide the following advantages: decomposition, granularity of roles/jobs, workflow, focus approach, and properties. RPAM01 can benefit from the Napoleon Model by including constraints and policies.

6.3 Summary of RPAM01 vs. Role/Finding and Napoleon Models

I have analyzed the relation between RPAM01 and two other models, the decomposition Role-Finding Model and the aggregation Napoleon Model. I observed the commonality of these models. Each is concerned with:

- assigning permissions to roles,
- administering the model, and
- considering processes in the approach.

More importantly, I found the benefits from the original concepts of RPAM01 to assist in the development of other decomposition and aggregation approach models. These benefits include:

- Focusing the Decisions: The other models do not consider focusing their decisions on the applications;
- Defining Model Properties: There is no concept of reuse to prevent the recreation of model components, nor is there a concept of completeness to ensure that all of the necessary model elements are used.
- Decomposing Roles into Jobs: Neither of the models considers that a role is comprised of multiple jobs. A Role-Finding Model considers job position and job function when defining a task. There is a concept of subroles, but neither roles nor subroles are defined as a process. The Napoleon Model provided the capability to have accesses based on a key chain. The key chain can combine a set of permissions that can be aggregated into a job. The jobs can

be combined to create a role. Each job may be considered a semantic key chain.

- Relation Between Jobs and Workpatterns: Workpatterns can use a process to decompose a role into a logical flow of steps that are required to perform the work. These can simply be assigned to a permission. Both models consider a process, but they do not implement a process as a logical flow of steps to identify the permissions that are required to perform the work of each job.

Finally, I observed that only RPAM01 provided an approach for both decomposition of roles to permission and the aggregation of permission to roles.

7.0 CONCLUSION

7.1 Summary

This dissertation demonstrates that it is possible to create a model and methodology to decompose roles into permissions and aggregate permissions into roles. It also shows that, by applying properties to the RPAM01 model, the effectiveness of role administration can be increased.

The RPAM01 model extended the RBAC96 model by adding three layers: job, workpattern, and tasks. The responsibilities of a role were subdivided into a concept of a job, which was then decomposed into a discrete, atomical unit of work that is referred to as a task. The translation from the job to a task was accomplished by identifying the steps required to perform the work of a job in a set called a workpattern.

Decomposition from roles to permissions was accomplished with an approach that started by selecting a focus that could be used in defining the criterion for decision-making. Once the criterion is chosen, the concept of analyzing, tracking, or brainstorming is used to decompose roles into jobs. After determining if the workpattern is single-process, multi-process, or ad-hoc, the approach identifies the steps. The tasks that are assigned to by the steps are mapped to permissions.

The aggregation approach starts with the definition of the focus; however, the next two layers use the concept of buckets to determine the permissions that should be grouped into tasks; and subsequently, the tasks that should be grouped into a workpattern. The result may require that the workpattern perform an additional task to complete the work of the job. Next, the workpattern is assigned to the job; and from the list of jobs, the jobs are selected and assigned to the roles based on the access needs of the work being conducted by the role.

The aggregation and decomposition approaches were improved by applying the minimization property, which eliminates equivalent elements that are not unique. In addition, a property was introduced, which allows the reuse of elements from one layer to the next layer. The completeness property was then applied to ensure that all permissions and roles were assigned.

Finally, the dissertation demonstrates that a benefit of the approaches is the ability to create a placeholder for permission growth. The placeholder is a permission-free task that can be assigned in the future when there is a need to access an application that does not currently require permissions. Another benefit of using this approach is that it provides the ability to not have to completely redefine a role when a new job is assigned. Instead, the new job can simply be mapped to a role.

7.2 RPAM01 Contributions

In this dissertation, I 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 a layer of the model layer and to define the relation between each of these layers.

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

Another concept to aid in engineering is the ability to define the jobs of a role. I began by categorizing roles into Documented, Existing, or Undefined. I further detailed the jobs by using process as a means of decomposing the job into a set of steps. I realized that not all permissions that are required by a job might not be part of a process, so I added an ad-hoc set for disjointed steps.

I also found that I needed a concept to aid in the aggregation of permissions. I 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, I considered the economy of re-using terms, efficiency of eliminating redundancy, and the ability to perform all necessary work. I could enhance mapping the 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 (If so, minimize equivalent (permission or not) elements to provide an efficient assignment); and
3. Performing all the necessary work to ensure that there is a complete mapping of elements between layers.

7.3 Future Research

During the presentation of this dissertation, I identify future research that could enhance RPAM01.

I note that constraints offer an important approach for restricting data access. However, these constraints were considered out of the scope of this dissertation and were not included within the model. By introducing constraints, I supplemented the model by explaining additional concepts. For example, I could create a policy based on how the constraints are applied to the different layers of the model. I learned that: 1) constraints can be used to resolve the issue of distinguishing between an application access and the data the application executes; 2) even when the user has permission to access an application, he/she cannot unequivocally provide the user with the authority to see all of the data executable by that application; and 3) by creating a set of rules constraints can restrict access to the data based on some attribute.

I note another area of research, the introduction of properties, into other models. Currently, no other known research has a concept of properties. Permissions are eventually assigned to users without consideration of reuse, minimization, or completeness. These concepts could help improve efficiency for other models.

I observe that I can include role hierarchy and consider how a role hierarchy would affect the definition of the jobs. With the addition of role hierarchies, I can determine if the responsibilities of a job can be considered as a “child” role and be inherited by a role hierarchy, instead of defining another job.

Finally, I can learn that additional research is needed to concentrate on the benefit of a process. While I can provide a detailed analysis on how a business process can be used to define a role, I realize that I need to consider more than the process. For example, I may need to include attributes such as the organization, geographic location, or a company to determine the needed permissions of a role.

REFERENCES

8.0 REFERENCES

[B90] R.W. Baldwin, Naming and grouping privileges to simplify security management in large databases. In Proceedings, IEEE Computer Security Symposium on Research in Security and Privacy, IEEE Computer Society, 1990.

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

[B98] Konstantin Beznosov, Requirements for Access Control: US Healthcare Domain, In Proceedings of Third ACM Workshop on Role-Based Access Control, October 22-23, 1998.

[BC98] John Barkley, Anthony Cincotta, Managing Role/Permission Relationships Using Object Access Types, In Proceedings of Third ACM Workshop on Role-Based Access Control, October 22-23, 1998.

[BLP75] D.E. Bell and L.J. La Padula. Secure Computer System: Unified Exposition and Multic Interpretation. Technical Report MTIS AD-A023588, MITRE Corporation, 1975.

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

[C01] Ramaswamy Chandramouli (NIST), A Framework for Multiple Types in a Healthcare Application System, In Proceedings of 17th Computer Security Applications Conference, New Orleans, Louisiana, December 10-14, 2001.

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

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

[CY95] E. Coyne, C. Youman, Workshop Discussion, In Proceedings of First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, November 30-December 1, 1995.

[ES1] Pete Epstein, Ravi Sandhu, Engineering of Role Permission Assignments, In Proceedings of 17th Computer Security Applications Conference, New Orleans, Louisiana, December 10-14, 2001.

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

[FCK95] D. Ferraiolo and J. Cugini, and R. Kuhn, Role-based access Control (RBAC): Features and motivations, IN Proceedings of 11th Annual Computer Security application Conference, December 11-15, 1995.

[FK92] D. Ferraiolo and D.K. Kun, Role Based access control. In 15th National Computer Security Conference, NIST/NSA, 1992.

[FSGKC01] David Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, Ramaswamy Chandramouli, Proposed NIST Standard for Role-Based Access Control, 2001.

[G95] Luigi Giuri, Role-Based Access Control: A Natural Approach, In Proceedings of First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, November 30-December 1, 1995.

[G98] Luigi Giuri, RBAC in Java, In Proceedings of Third ACM Workshop on Role-Based Access Control, October 22-23, 1998.

[GH00] Thomas Gebhardt, Thomas Hildman, Enabling Technologies for Role-based Online Decision Engines, In Proceedings of Fifth ACM Workshop on Role-Based Access Control, July 26-27, 2000.

[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, Fairfax, VA October 28-29, 1999.

[KPF01] M. Kang, J. Park, J. Froscher, Access Control Mechanisms for Inter-Organizational Workflow, In Proceedings of Sixth ACM Workshop on Role-Based Access Control, May 3-4 2001.

[M97] William J. Meyers, RBAC Emulation on Trusted DG/UX, In Proceedings of Second ACM Workshop on Role-Based Access Control, November 6-7, 1997.

[M98] Jonathan Moffett, Control Principle and Role Hiearchies, In Proceedings of Third ACM Workshop on Role-Based Access Control, October 22-23, 1998.

[N94] M. Nyanchama, Commerical Integrity, Roles and Object Orientation, PhD thesis, Department of Computer Science, The University of Western Ontario, London, Canada, Sept. 1994.

[N95] L. Notargiacomo, Role-Based Access Control in ORACLE7 and Trusted ORACLE7, In Proceedings of First ACM Workshop on Role-Based Access Control, November 30 – December 1, 1995.

[NO93] M. Nyanchama and S. Osborn, Role-based Security, Object Oriented Databases and Separation of Duty, Sigmod Record, 1993.

[NO94] M. Nyanchama and S.L. Osborn, Access rights administration in role-based security systems. In J. Biskup, M. Morgenstern, and C.E. Landwehr, editors, Database Security, VIII, States and Prospects, Proceedings of the IFIP WG 11.3 Working Conference on Database Security, North-Holland, 1994.

[NO95] M. Nyanchama and S.L. Osborn, Modeling mandatory access control in role-based security systems. In D.L. Spooner, S.A. Demurjian, and J.E. Dobson, editors. Proceedings of the IFIP WG 11.3 Ninth Annual Working Conference on Database Security, Chapman & Hall, 1995.

[NO98] M. Nyanchama and S.L. Osborn, The Role Graph Model, In Proceedings of First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, November 30-December 1, 1995.

[O97] Sylvia Osborn, Mandatory Access Control and Role-Based Access Control Revisited, In Proceedings of Second ACM Workshop on Role-Based Access Control, November 6-7, 1997.

[PS01] N. Perwaiz, I. Sommerville, Structured Management of Role-Permission Relationships, In Proceedings of Sixth ACM Workshop on Role-Based Access Control, May 3-4 2001.

[R95] R. Sandhu, Rationale for the RBAC96 Family of Access Control Models, In Proceedings of First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, November 30-December 1, 1995.

[R96] Rational Rose/C^{++}, Rational Rose Software Corporation, Summit Software, Santa Clara, CA, **www.rational.com**, Copyright 1996.

[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, July 26-27, 2000.

[S75] Louis Shapiro, Introduction to Abstract Algebra, 1975, McGraw-Hill.

[S96] R.S. Sandhu, Role Hiearchies and Constraints for Lattice-Based Aaccess Controls. In Computer Security – ESORICS 96, Springer Verlag, 1996 Lecture Notes 1146.

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

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

[T88] T.C. Ting, A User-Role Based Data Security Approach, in Database Security: Status and Prospects, C.E. Landwehr, Editor. 1988, Elsevier.

[TB79] Thomas, E. and B. Biddle, The Nature and History of Role Theory, in Role Theory: Concepts and Research, B. Biddle an E. Thomas, Editors, 1979 Kreiger Publishing.

[TJ00] J. Tidswell and T. Jaeger, Integrated Constraints and Inheritance in DTC, In Proceedings of Fifth ACM Workshop on Role-Based Access Control, July 26-27, 2000.

[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, December 7-11, 1998.

[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, Fairfax, VA, October 28-29, 1999.

CURRICULUM VITAE

Pete A. Epstein was born on May 27, 1957, in Mount Vernon, New York and is an American citizen. He graduated from Valhalla High School, Valhalla, New York, in 1975. He received his Bachelor of Science degree from University of Maryland in 1980. He received his Master of Business Administration from Marymount University in 1984. He has been employed in the Information Technology field for 21 years and received his Doctor of Philosophy in Information Technology from George Mason University in 2002.