Towards A UML Based Approach to Role Engineering

Pete Epstein
AT&T Laboratories
6012 Lochanora Lane
Manassas, VA 20111
mepstein@tidalwave.net

Ravi Sandhu
ISE Department,
MS 4A4,
GMU, Fairfax, VA 22030
sandhu@gmu.edu

Abstract

Role based access control (RBAC) is a promising technology for scalable access control. For RBAC to rise to its full potential, the roles must be properly constructed to reflect organizational access control policy and needs. This requires a discipline of Role Engineering to develop various components of RBAC such as role hierarchy, permissions (and permission-role assignment), and constraints. The importance of Role Engineering has been recognized but very little work has been done to date. In this paper we explore the possibility of using the Unified Modeling Language (UML) to support Role Engineering. We chose UML because it is a de facto standard and reflects a consensus in the modeling community. To investigate the capability of UML for Role Engineering, we represent an existing Role framework recently published by Thomsen, O'Brien, and Bogle. This framework can be modeled in UML, with the assistance of adding a new user defined UML vocabulary.

Key Words

Role Based Access Control, RBAC, Unified Modeling Language, UML, Role Engineering

1. Introduction

The continuing interest in RBAC is being addressed in the number of RBAC research papers and projects currently under way. RBAC will continue to grow if its fundamental entities, "roles," can be administered; and permissions can be assigned in a systematic manner to these roles. This approach requires a thorough definition of a new discipline: Role Engineering.

Coyne states [C95] that for Role Engineering to be performed on an $RBAC_3$ model [SCFY96], the following components must be defined:

- Roles,
- Permissions (and permission-role assignment),
- Constraints, and
- Hierarchies.

These components must be engineered as part of deploying an RBAC system. One method of representing engineering results is by using a modeling language.

In recent years UML has emerged as a de facto standard for object-oriented modeling of software-intensive systems. UML is "a graphical language for visualizing, specifying, constructing, and documenting the artifacts" of systems [BRJ99]. There are several companies that provide automated tools to support UML; Rational is one of these companies who offer the Rose product. UML products generate C^{++} code for a program's shell. The generated code can be used to envelop the more detailed logical code that can be written into the structure.

In this paper we explore the possibility of using the UML to support Role Engineering. Documenting the Role Engineering Components of an RBAC model in UML syntax will show this possibility. The model chosen is the Role Based Access Control Framework for Network Enterprises (FNE), developed by Thomsen, O'Brien, and Bogle [TOB98]. Choosing an independently developed existing model for this exercise gives us an element of objectivity in assessing the modeling power of UML in this arena.

Seven abstract layers represent the FNE model. Two different groups engineer the layers. The Application

Developers are responsible for the first four layers. The remaining three layers are maintained by the Local System Administrator.

Within each of these groups, there is a need to create a role hierarchy. Layer 4 requires a role hierarchy of the Application Keys; and Layer 6 requires a role hierarchy in the organization of the key chains. The other five layers also support the implementation of the keys; as such, they also require their own engineering. Innate within the model is the need for Role Engineering of the model's basic components as roles, permissions, constraints, and hierarchies. Programmers can use the resultant UML graphics to implement a solution.

This paper shows the feasibility of using UML to support Role Engineering of an RBAC model. This is accomplished by a sequential approach. In the first section, Role Engineering is introduced. Then, an overview of UML is presented. We discuss the UML syntax that can be used in representing the FNE model and introduce new UML stereotypes. The next section provides a review of the FNE model. Section five covers the representation of each of the seven abstract layers of the RBAC framework model in UML. In addition, we discuss Role Engineering of the model and focus on the role hierarchy presented in layers 4 and 6. We give the UML graphics that can be used to engineer the model. The remainder of the paper summarizes the strengths and weaknesses of using UML to present an RBAC model. The paper concludes with a summary and direction for future work.

2. Role Engineering

We assume basic familiarity with RBAC concepts such as those given in the RBAC96 model [SCFY96]. The roles of the RBAC model must be engineered in a structured manner so that the model will correctly express an organization's policy.

Role Engineering is concerned with designing an RBAC model's components. The components we are concerned about in this paper are Roles, Permissions, Constraints, and Hierarchies. Sandhu defines these terms as [S98]:

Roles – a job function within the organization that describes the authority and responsibility conferred on a user assigned to the role;

Permissions – a description of the type of authorized interactions a subject can have with an object;

Constraints - A relationship between and among roles; and,

Hierarchies – A partial order of permission-inheritance relationship established among roles.

These RBAC model components that require Role Engineering can be represented graphically. We will use the UML language to document an RBAC model, namely the FNE model.

There is some overlap between the terminology for FNE and UML (e.g., an object means something different for each concept). In spite of the differences, in Section 5 we will show how the concepts of FNE and UML interrelate and how they can be graphically depicted.

3. Unified Modeling Language

This section provides a general overview of UML concentrating on the syntax that is relevant to this paper. Figure 1 displays the different types of UML syntax used in this paper. In addition, we introduce new UML syntax (Vocabulary) in the form of stereotypes. For further information on UML the reader is referred to [BRJ99]. UML has three main building blocks: Things, Relationships, and Diagrams.

"Things" are the main components of the model. "Things" are connected by Relationships. Diagrams display the Things and Relationships in different active or passive contexts. For example, a diagram can document a dynamic process in which a student may register for a class or it can document a static data structure of an organization.

There are four kinds of things: Structural, Behavior, Grouping, and Annotational.

One of the seven structural "things" of interest is a class. A class can contain a name, attributes, and operations. Classes will be used with objects.

Behavior "things" are the verbs of UML. They are the dynamic parts of the UML. Behavior "things" will not be discussed in this paper.

A grouping "thing," as the name states, permits the combining of different parts under a similar category. We will use the grouping "thing" named "package."

The final "thing" is annotational (it can also be called a note). Notes comment a model. Notes can be used to comment the enterprise constraints of a key chain.

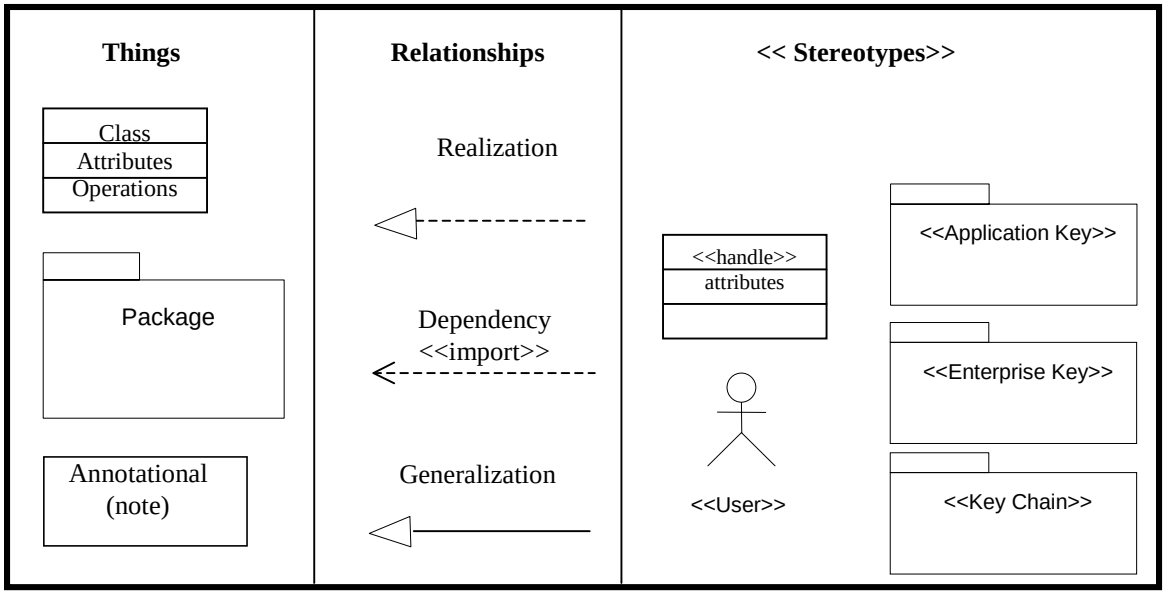

Figure 1: UML Syntax

Interconnections between components are accomplished by relationships. There are four kinds of relationships: Dependency, Association, Generalization, and Realization.

Dependency is a relationship between two structural things where the modification of one "thing" may affect the other "thing." An association shows an n – n relationship between two "things." For example, there can be several houses on a street, or a house can be on multiple streets (i.e., a corner house).

Another type of relationship is a generalization. A generalization is "a specialized/generalized relationship, in which objects of the specialized element (the child) are substitutable for objects of the generalized element (the parent)." This type of inheritance is similar to object-oriented inheritance; both are transitive. Generalization can be used between classes and between packages.

The last relationship is a realization. It is "a semantic relationship between classifiers, in which one classifier specifies a contract that another classifier guarantees to carry out." When we discuss the RBAC model, we will use a realization relationship between an interface of a handle and a class of an object. The handle will specify a contract for the object to carry out.

Unlike classes, packages are more limited; there are only two types of relationships: generalization and dependency. A package uses a special type of dependency called import and access. A dependent package can have access to an imported package if its contents are available to be exported (i.e., the package contents are marked with a "+"). Imported packages are not transitive. Dependencies are used to assign application keys to enterprise keys.

There are nine types of diagrams that can document the system: Class, Object, Use Case, Sequence, Collaboration, Statechart, Activity, Component, and Deployment. We are concerned with the documentation of Classes.

There are a few more items to note with regard to UML syntax: UML distinguishes between operations and methods. Operations can be considered as the name of a service being performed. A method is the implementation (i.e., the logic behind the service) of the operation. This is an important concept to remember when we represent methods.

UML defines an extensive syntax; however, it is not adequate for our purpose. Fortunately, UML permits us to define additional vocabulary by using stereotypes. UML uses stereotypes to define an actor, which is of a type class. We will use an actor to represent a user. Another stereotype is an interface. An interface is also a type class but it does not define an implementation of operations. Instead, the operations in an interface are realized by other classes.

We will define four additional stereotypes. They will be italicized.

The first stereotype is a *Handle*. It is of type class. It will be used as an interface for an object. The remaining three stereotypes are type package. The second stereotype is used for *Application Keys* and the third is used for E*nterprise Keys*. The last stereotype is a *Key Chain*. The *Key Chain* is a collection of *Enterprise Keys* of a type package. Objects, handles, application keys, enterprise keys, and key chains are defined in more detail in the next section.

Commercial UML products check for some syntax errors. However, these products do not check for inconsistent logic. It is the responsibility of the engineer to ensure that the UML syntax is logically and semantically correct.

4. The RBAC Framework for Network Enterprises (FNE)

The FNE model is used in this paper as a representative RBAC model with a Role Engineering orientation. FNE is based on the divide-and-conquer principle. No one person is responsible for the security management of the entire system.

In fact, two different groups, Application Developers and Local System Administrators, are responsible for administering the seven abstract layers of the FNE Model. These two groups are responsible for Role Engineering of their respective layers. The seven layers of FNE are divided into two categories.

The Application Developer is responsible for designing the first four layers:

1. Objects,
2. Object Handles,
3. Application Constraints, and
4. Application Keys.

The Local System Administrator is responsible for managing the upper three layers:

5. Enterprise Keys,
6. Key Chains, and
7. Enterprise Constraints.

The first four layers are the basic building blocks of the model. The main component is an object. An object has a name and a set of public methods that can be used to access the object. The methods can be considered the permissions necessary to perform the actions on the object. The attributes can be used as a condition for constraints. For example, an attribute may contain information about a patient's doctor. If the doctor is not treating a specific patient, then the doctor will not be able to view that patient's record.

These basic building blocks can be grouped into an object handle set. The object handle is a collection of named objects and the methods that can access the objects. Using grouping, the number of disparate methods is reduced. For example, one group can be a handle for all "get" methods.

The next layer is the Application Constraint Layer. An application constraint must be satisfied before access can be granted to the methods in a handle set.

The last layer of the first group is for application keys. Application keys can be considered as an application specific role for health roles (see figure 2). An application key associates a role with objects, data records, and methods. These methods must be created by the application programmer to ensure that at least one role has the ability to perform any required operation within the organization. Each key within the hierarchy is considered a role. The keys can be placed in a hierarchy, similar to a role hierarchy. Controlling the hierarchy allows the application developer to satisfy an organization's security policy.

The application keys are either at a leaf node (e.g., Consulting Physician, Primary Physician, and Nurse) or at a non-leaf node (e.g., Doctor, Health Care Provider). Any key at the non-leaf node is considered to be an abstract role. The importance of this is that an abstract role can not later be mapped to an enterprise key; consequently, a user can not be assigned an abstract role.

The FNE model resolves multiple inheritance ambiguities that plague object-oriented hierarchies. There is no problem in determining the origination of a method nor a problem in determining which method should be used. If a key inherits methods from more than one key, then in the worse case scenario, the keys contain the same method with different constraints. The FNE policy is that constraints are logically "Ored." The user needs only to satisfy either constraint to gain access to the method.

The next three levels, the enterprise layers, are maintained by the local system administrator. The first of these layers is the fifth layer of the model and it is used for the creation of enterprise keys. Each application key that is a non-abstract role is mapped one-to-one to an enterprise key. Users can either be assigned to an enterprise key at this layer or to a key chain at the next higher layer.

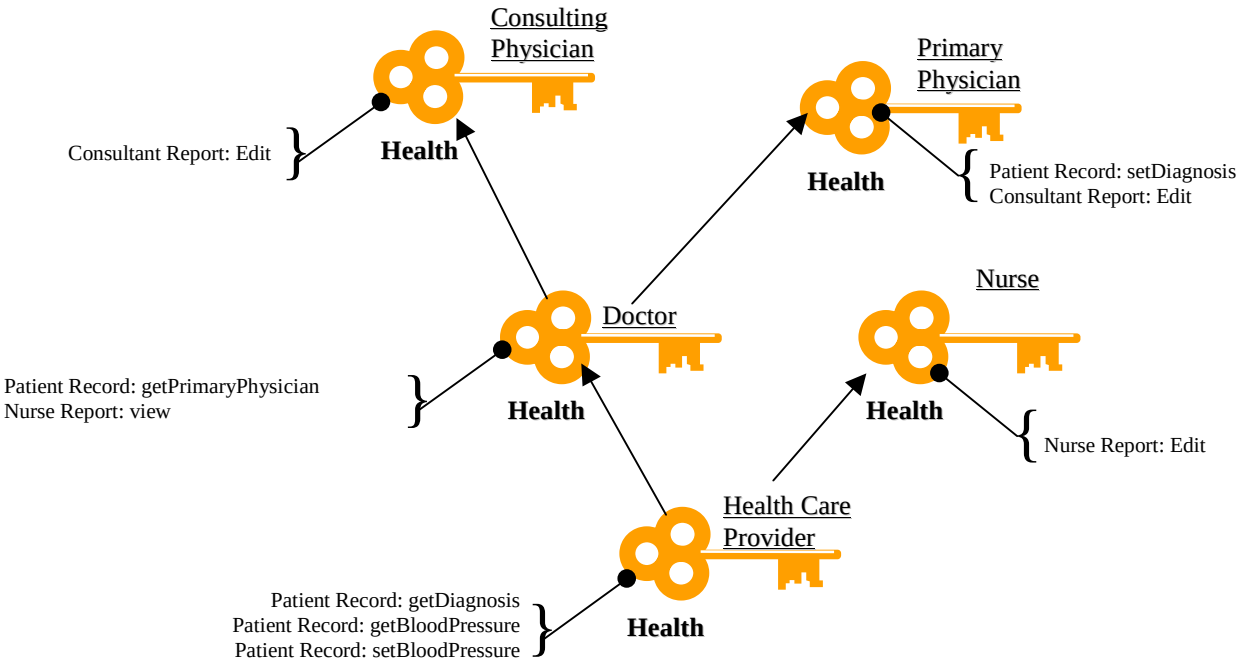

Figure 2: A sample Application key hierarchy

The enterprise key permits the user to access the methods of the object listed in the key only if the application constraints are satisfied. A user can be assigned enterprise keys that are part of different application key hierarchies.

The Enterprise keys can be added to a key chain in layer 6. A key chain can be assigned to a user. Similar to the application developers engineering the role hierarchies at layer 4, the local administrators have the ability to conform to a security policy by structuring the key chains and assigning users to those key chains. Multiple inheritance for key chains is resolved in the same manner as multiple inheritance for the application key hierarchy.

The final layer, layer 7, is used to specify enterprise constraints. Enterprise constraints are applied to key chains to restrict the user from unauthorized access to applications throughout the enterprise, or system.

5. Applying UML to the FNE Model

UML can be used for documenting active and passive flows. Process flows of roles can be shown; but for this paper, only the static data structure of an RBAC model will be presented.

The challenge is to depict the FNE model by using UML syntax. We meet this challenge by presenting each of the FNE model's layers by UML syntax.

5.1 Layer 1: Object

The first UML layer shows the basic building block of an object. The object must have public methods that can access the object. Attributes and private methods are hidden. Under UML, the class contains the name of the object (see figure 3); the operations can be in full view of other classes (i.e., public) or just viewable by the specific class (i.e., private). Attributes and operations can also be listed within the class.

There is an inconsistency between the operation and method terminology defined by UML and FNE. UML defines a method as an implementation of an operation. The FNE paper defines a method as a means of accessing an object. For the purpose of this paper, the methods will be defined within a class.

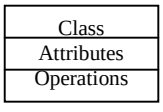

Figure 3: Object and Methods (Layer 1)

5.2 Layer 2: Object Handle

The next layer is the Object Handle Layer. Methods can be grouped into a *Handle*. *Handle* has similar characteristics to an interface. To create an object *Handle*, we establish a realization relationship between a class and a *Handle* (see figure 4). The methods for the "set diag" operation in the "set" handle is defined in the "Patient Record" class under "set diag."

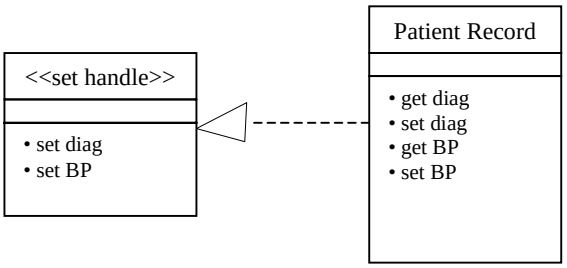

Figure 4: Object Handle (Layer 2)

5.3 Layer 3: Application Constraint

The layer following the Object Handle Layer is the Application Constraint Layer. Constraint restriction on an operation can be documented as a precondition. The precondition must be satisfied prior to executing the operations. The preconditions are entered when the operations are defined in the class specification. There is not a graphic depiction of the precondition; however, when the class and its operations are printed, then the information in figure 5 will also be displayed. Figure 5 shows a precondition for a GetPrimaryPhysican operation of a Patient Record Class.

Operation name:

GetPrimaryPhysician

Public member of: Patient Record
Preconditions:
The person is a patient of the primary physician.

Figure 5: Object and Methods (Layer 3)

5.4 Layer 4: Application Key

Application keys are presented in layer 4. Layer 4 is the first of the two layers that uses a role hierarchy. The *Handles* are grouped within the *Application Key* (see figure 6). The hierarchical diagram, similar to figure 2, uses a generalization relationship to represent role inheritance. The arrows for the generalization relationship are oriented in the opposite direction than in figure 2. The *Application Key* can inherit operations from a lower layer *Application Key*.

Abstract packages can not be represented as easily as abstract classes. Instead, a note can be used to indicate that the local administrator can not assign the non-leaf node to an enterprise key.

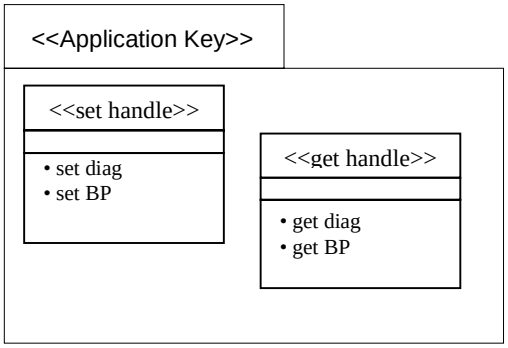

Figure 6: Application Keys (Layer 4)

5.5 Layer 5: Enterprise Key

The next layer is where the enterprise keys are defined. Each *Application Key*, other than an abstract key, is mapped one-to-one to an *Enterprise Key* (see figure 7). The *Enterprise Keys* can be assigned to a user.

.

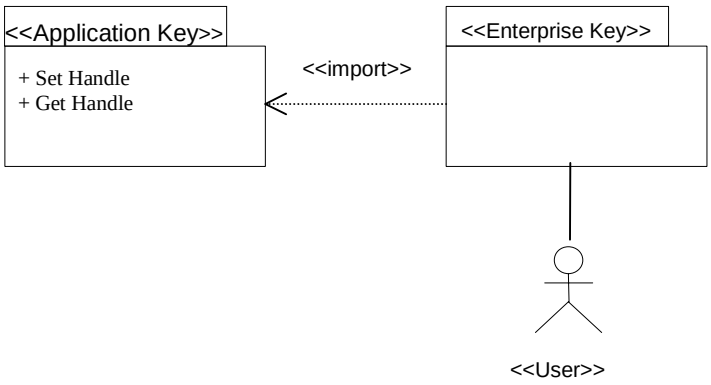

Figure 7: Enterprise Key (Layer 5)

5.6 Layer 6: Key Chain

The keys can be combined onto a key chain. The *Key Chain* contains a collection of one or more *Enterprise Keys*. The *Key Chain* can be assigned to users (see Figure 8). The enterprise keys are labeled E1, E2, and E3. If the application constraints are satisfied, then the user has the ability to use all of the *Enterprise Keys*.

5.7 Layer 7: Enterprise Constraint

Layer 7 is the Enterprise Constraint Layer. A note can be attached to each key chain. The enterprise constraint conditions will be describe within the note (see figure 9).

5.8 UML Approach

We now have a representation of the RBAC Framework for the Network Enterprises Model using UML.

Role Engineering of the FNE model is summarized in Table 1.

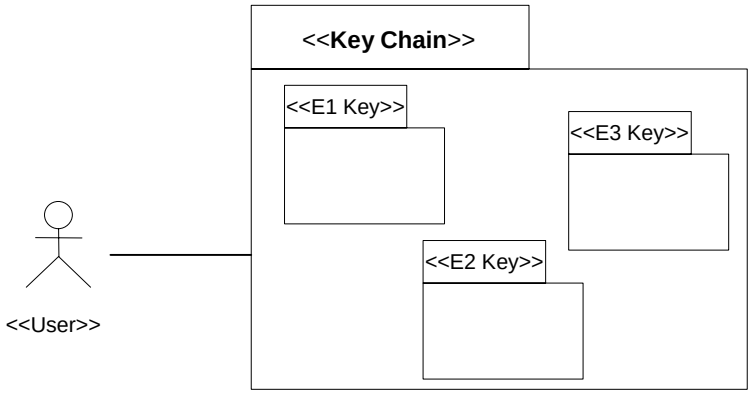

Figure 8: Key Chain (Layer 6)

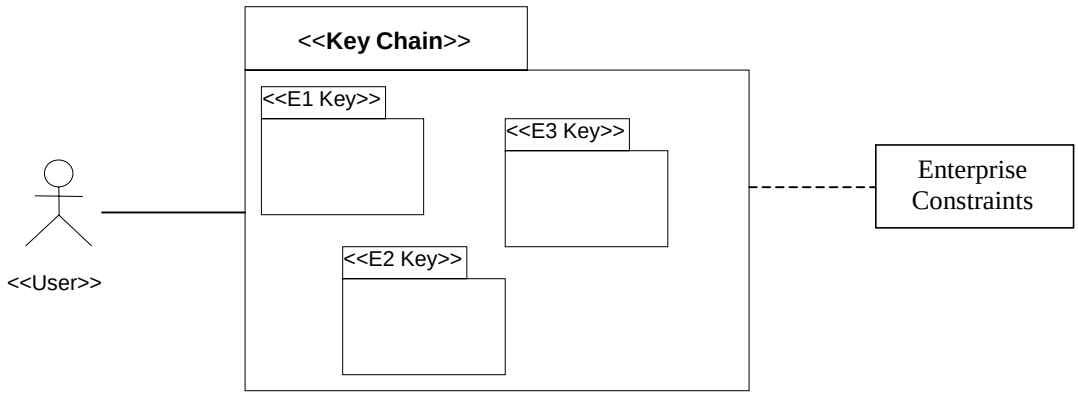

Figure 9: Enterprise Constraints (Layer 7)

An "X" marks each component of the FNE model that can be Role Engineered. We have added User Assignment (UA) to show where users are assigned to roles.

	Roles	Permission	Constraint	Hierarchy	UA
1		X			
2		X			
3			X		
4	X			X	
5	X				X
6				X	X
7			X		

Table 1 : Role Engineering of the Seven Layers

We have shown that the FNE Model can be documented using UML. Furthermore, by representing the FNE model, we are supporting Role Engineering by documenting the components of the FNE model. In essence, the UML produced graphics are providing a UML based approach to Role Engineering.

6. Improving the UML ability to model RBAC

In most cases, UML is able to represent an RBAC model cleanly; however, with some modification to the syntax and the FNE model, the representation can be more precise. This section identifies weaknesses in using UML to document the FNE model.

The first distinction is terminology. The FNE model defines an Object as "an abstract description of some kind of data in the system," and UML defines an Object as "a concrete manifestation of an abstraction." There is also an inconsistency between the operation and method definitions between UML and the FNE paper.

An additional area of distinction is that the UML model has a concept of abstract classes but not of abstract packages. Another small difference is that the arrows for UML's generalization relationship points in a different direction than the arrows in the FNE model's application key hierarchy.

Although there is a check on the UML syntax, there is no logic or semantic check. We have to trust the designer to accurately depict the model.

Further benefits can be obtained if the UML C^{++} generator creates more detailed code rather than use just the shell of a program. This can also save programming time.

The FNE model permits users to be assigned at layers 5 and 6. To reduce complexity, users should be assigned at layer 6. If only one key is used, then the user can be assigned to a key chain that contains that one key.

Documenting constraints is awkward. A formal language has not been defined to state UML preconditions. Once accomplished, we can revisit the creation of vocabulary extensions to implement constraints.

Finally, we did not document the process flow of the two role hierarchies in layers 4 and 6. By documenting the process, we have the ability to identify all the

permissions that may be necessary for the role to perform the required task. The process flow can mirror the organization's workflow.

7. Conclusion – Future Research

In this paper, we presented a UML approach to Role Engineering. Initially, we defined what we meant by Role Engineering. Next, we identified the key syntax of UML that can be used in documenting an RBAC model. The model chosen is the Role Based Access Control Framework for Network Enterprises by Thomsen, O'Brien, and Bogle.

We proceeded to document each layer of the model by UML syntax. By doing so, we also realized the components documented are the same ones that can be created and structured by Role Engineering. The final product of Role Engineering can be the UML generated graphical documentation. We were able to use a UML based approach to assist in the Role Engineering of an RBAC model.

There are several areas that require future research. We have only documented one RBAC model. Another exercise would be to document another model by the UML. Furthermore, this paper does not delve into the process side of Role Engineering. In addition, it does not provide a Role Engineering framework or a methodology to implement that framework. It also does not show a methodical approach for defining constraints for UML or for an RBAC model. Finally, the FNE model does not include all RBAC concepts as separation of duties or least privileges. These concepts need to be introduced and modeled.

With the continued expansion of the Internet, there will be an even greater need for access control. For strong acceptance to continue, RBAC must be integrated into the framework of the Internet. Two of the areas in which this can be accomplished are:

1. Using Role Engineering to generate the personnel or computer organization that can be easily implemented by Internet access control tools.

2. Incorporating RBAC within the soon-to-be released Directory Enabled Network (DEN) standard. The DEN's specification will be part of the Desktop Management Task Force's (DMTF) Common Information Model for enterprise management.

The UML approach to Role Engineering will be able to assist in documenting both of these endeavors.

References

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

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

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

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

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

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

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

[L91] Robert LaFore, Object-Oriented Programming in Turbo C++, Waite Group Press, 1991