Determining Role Rights from Use Cases

E. B. Fernandez and J. C. Hawkins
Dept. of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{ed | jhawkins@cse.fau.edu}

Abstract

We propose a simple and complete method to determine the needed rights for roles in a system. We make use of the concept of use cases, commonly used to determine requirements in object-oriented system development. We extend use cases with rights specifications and we determine all of a role's rights from the collection of all use cases for the system. This method is in strict accordance with the least privilege principle.

Keywords: Authorization models, Object-oriented analysis, Object-oriented authorization, RoIe-Based-Access-Control, Use cases.

1. Introduction

User roles are the basis for many enterprise models, including cooperative work models [Rupi94] and workflow models [Kapp95]. Role-based access control is an attempt to associate authorization rights with specific user roles [Ferr95, Sand96]; that is, this approach is an embodiment of the least privilege principle, where users acquire rights according to their functions.

There is a good amount of work defining formal and structural aspects of roles [Ferr92, Ferr95, Nyan94, Sand96]. Some work focuses on mechanisms to group users and to define the rights of groups [Fern95b]. However, there is little work on how these rights are generated; that is, how to determine the rights to be given to some role. As said above, roles can directly support the least privilege principle, but to do so there must be some method to assign only the needed rights to each role.

One of the most accepted methods to determine functional requirements for the design of object-oriented systems is the definition of use cases [Jaco92, Rumb94a]. The users of the system are interviewed to elicit their ways of interacting with the system and their interaction descriptions are recorded in a specific format. Use cases are the basis for precise requirement documents, for finding class operations and even for finding classes. In our work, we have also found that sequences of use cases are valuable to understand the system and to define test cases [Fern97].

We propose here a method to determine the needed rights for a role by considering use cases and sequences of use cases. This approach guarantees that all the roles receive their necessary rights so they can perform their functions and no more than their functions. A security administrator defines the authorization rules based on all the use cases for the system.

Section 2 provides some background on use cases and their sequences. Section 3 describes our extended use cases where nonfunctional specifications can be added. Section 4 proposes an approach to generating rights for roles, while Section 5 discusses administrative and enforcement aspects. The last section provides some conclusions. We illustrate these concepts with an example from a manufacturing system.

2. Use cases and their sequences.

Use cases describe all the required interactions of users with the system. They are textual semiformal descriptions and can be complemented with graphical representations, e.g., scenario (event trace) diagrams [Rumb9l]. Figure 1 shows a use case for shop order cutting, a step in the processing of shop orders for manufacturing some type of gadgets. A use case describes the actors (users, roles, or other systems) that interact with the system. In the example, the materials employee is the actor (which corresponds to a given enterprise role). A use case also includes a preconditions section, which specifies what must

be true in the system for the interaction to be feasible. The description part indicates details of the interactions. An exception indicates an abnormal, incorrect, or unusual situation, e.g., lack of enough components to fabricate the order. Finally, the postconditions section indicates what must be true at the end of the interaction; in our example, the shop order is in cut status and the required components have been reserved.

Title: Shop Order cutting ("Cutting" indicates the start of manufacturing)

Actors: Materials employee (this is a role in this system)

Preconditions: Shop order is in firm status (created from a customer order)

Description: Employee "cuts" a shop order for a given number of gadgets of a specific type [exception: component shortage].

Exceptions: Component shortage -- shop order is delayed

Postconditions: shop order is in "cut" status. The inventory has reserved the required components.

Figure 1. A use case for shop order cutting

Figure 2 shows a scenario diagram for shop order cutting. In general, there are several scenarios for each use case that correspond to the normal case and to the exceptions. From the scenario of Figure 2 we can see that only one actor is involved (the Materials employee); getBOM indicates getting the Bill of Materials from the gadget object to find out what components are required for its fabrication. The CompInv objects represent the component inventory objects (that keep track of component quantities), and Dist are the locations of these components in storage bins. Note that after the Materials employee applies the cut command,

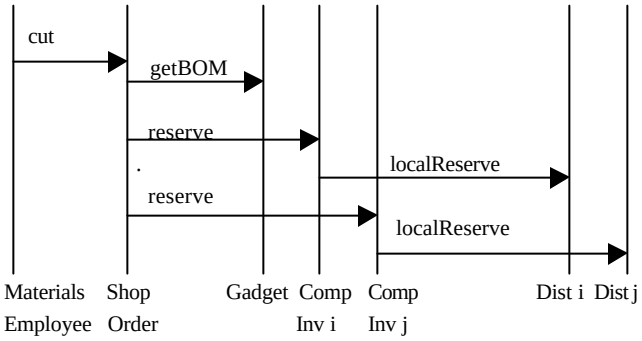

Figure 2. The normal scenario for shop order cutting.

a series of messages between the internal objects is triggered. The specific set of messages depends on the object model of the system; here, each needed component is reserved in the inventory and in the storage bin.

Sequences of use cases may also be important, e.g., a use case must be preceded or followed by other use cases to be meaningful or valid [Fern97]. In our example, shop order creation must be followed by cutting, picking and completion; revision can happen only before completion while cancellation can happen at any time.

3. Extended use cases

We need now to relate use cases to authorization rights to access specific object operations. Authorization rules can take the form (S, O, T, P), where S represents the subject (user or role), O represents the object being accessed, T represents the type of access allowed, and P represents an optional predicate defining access constraints. Object-oriented applications are especially suitable to apply authorization controls of this type since object data access is limited to that provided by specific object member methods or operations [Fern93]. We consider here rights of this type associated with specific roles.

It is clear that actors correspond to functional roles; i.e., some actor initiates a function, other actors may approve it, others may receive the result of some action.

Preconditions provide a way to indicate explicitly the authorizations required by the actors in a use case to perform their actions. Exceptions can be used to specify the action to be taken by the system in case of an attempted illegal access. Postconditions can be used to define shutdown final conditions in case of aborted access or other security actions, e.g., logging.

Security specifications correspond to nonfunctional specifications. Conventional use cases only allow the definition of functional specifications and we have proposed elsewhere an extension of use cases to indicate nonfunctional specifications by means of stereotypes [Hawk97]. A stereotype is a metaclassification of a UML element and is used to distinguish various classes in a model [UML97]. Here it distinguishes the type of requirement being expressed. Some stereotypical system architecture requirements are load, fault tolerance, security, and safety. In the example of Figure 3 we show the shop order use case of Figure 1 extended with security stereotypes to indicate access constraints. It is easy for the persons writing the use case (an application expert in conjunction with an object-oriented analysis expert) to

decide what type of access is needed for each actor (role) to perform its function.

Because use cases exercise all the possible functions of the system we can deduce all the needed role rights by considering the methods that need to be invoked by the corresponding actors. We formalize this principle in the next section.

Title: Shop order cutting
Actors: Materials employee
Preconditions: Shop order is in firm status. {security: Materials employee can cut shop orders }

Description: Employee cuts a shop order for a given number of gadgets of a specific type [exception: component shortage]. {security exception: Employee not authorized}

Exceptions: Component shortage - shop order is delayed {security: Employee not authorized - Log attempt, disable user interface}

Postconditions: Shop order is in cut status. The inventory has reserved the required components {security: interaction has been logged}

Figure 3. An extended use case.

4. Role-based Security Authorization

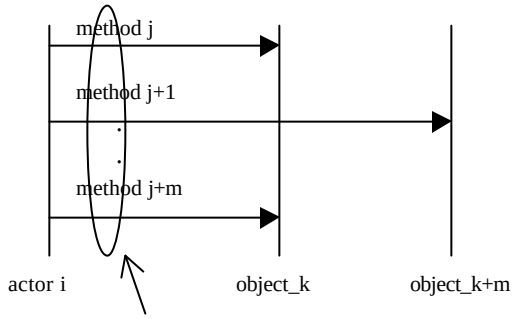

Figure 4. A generic scenario diagram for UseCase_q

We can get a formal expression for role rights by looking at a generic scenario diagram (Figure 4). In this example, actor_i interacts with object_k and object_k+l by invoking some of the objects' member methods, method j ... method j+m. Thus, UseCase_q defines a set of authorization rights, each of which is identified by the particular actor, method, and object. (In a non-object-oriented environment these methods would not be member methods, however the authorization right for a particular actor could still be identified by the method/function/procedure and target data structure.) Symbolically, an authorization right, R, is identified by the triplet:

$$R\ (A_i, M_j, O_k)$$

where A_i is an actor, M_j is a method, and O_k is an object. A right for a particular actor, A_i, to access object O_k using method M_j can be described as:

$$R_{j,k}\ (A_i) = (M_j, O_k)$$

For every actor there exists a set of rights implied by a use case, U_q. This set of rights is given by:

$$Ru_q(Ai) = \bigcup_{j_q,\ k_q\ =\ 1..n} \{Rj_q, k_q(Ai)\}$$

The union of the sets of rights across all use cases for a particular actor is the complete set of authorization rights for the actor denoted by:

$$R(Ai) = \bigcup_{q\ =\ 1..n} \{Ru_q(Ai)\}$$

This set of rights constitutes the role-based rights for actor (role) Ai. The union of the set of authorization rights across all use cases for every actor defines the complete set of authorization rights for the system, all of which are intended to perform a specific role-related function.

5 Administration and enforcement aspects

Based on our previous discussion, it is clear that all that the security administrator needs to do is analyze all the use case preconditions to decide what authorization rights are needed for each role. Depending on the system these rules may be written explicitly or may be deduced from group structuring [Fern95b]. In fact, they could be generated automatically from the use cases; tools such as Paradigm+ or Rational Rose now can keep track of use cases, they could be extended to generate the required authorization rules. From the use case exceptions the administrator implements the actions needed for security violations. The constraints in the preconditions are written when the use cases are developed by the application and analysis experts.

Addition or deletion of authorization rules is only necessary in case a use case is added or deleted or some of the actions of a use case are changed. There should be no other reason to add or delete rules. This is important to preserve the least privilege principle. Users are assigned to roles based on their job descriptions.

Consistently with this approach, authorization should be enforced at the user interface. Object-oriented systems use approaches based on model-view separation, e.g., the MVC [Rumb94b] or PAC architectures [Losa97]. These two models separate the conceptual model objects; shop orders, inventory in our example, from user interfaces that can observe and modify these conceptual objects. The user views should be defined based on use cases [Losa97], and it is clear that they should be the only way to interact with the system. The user views should have access to the set of authorization rules to allow or deny access to the conceptual objects in the system (Figure 5).

Clearly, the lower levels must participate in the enforcement of the rules [Fern95a, Neum86], the user should not be able to bypass the authorization defined in the views. Approaches such as capabilities, cryptography, etc., are valuable for this purpose but security restrictions should not be defined at these levels, these approaches only enforce the rules defined at the application level.

Notice that the approach is independent of the actual system implementation. Only the actor's commands to the system need to be authorized, not the internal object accesses triggered by these commands. As far as the external view of the system does not change, there is no need to change authorization rules when the implementation changes. This is in consistency with the information hiding property of object-oriented systems.

Sequences of use cases can be used to define a workflow that requires a specific set of authorizations for different roles. For example, a shop order can only be created by an Order Entry employee, cut and picked by a Materials employee, and completed by a Manufacturing employee. This complete workflow could be authorized as a unit.

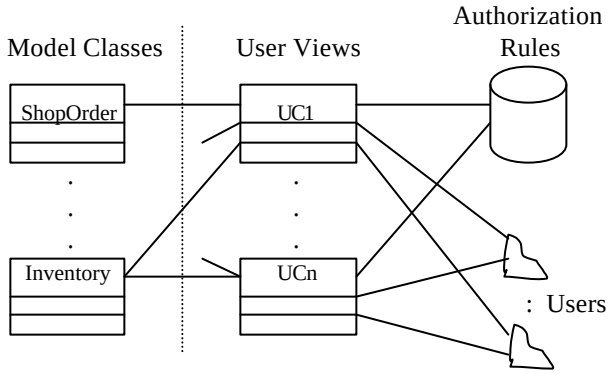

Figure 5. Authorization enforcement

6. Conclusions

We have proposed use cases as a convenient way to originate authorization rights for enterprise roles. This is consistent with the principle of defining authorization at the highest possible level, where their semantics are explicit [Fern93]. This way of defining authorization is based on enterprise needs and is in sharp contrast to "computer-oriented" policies, e.g. ownership, where a user creating a file gets all rights for it; we believe, policies of that type are poor from a security point of view. Notice that use cases are needed not only to develop new systems but to reengineer corporations around the object model [Newm95] and to define new architectures for legacy systems using distributed objects [Henn96].

The security administration and use of such a system should be much easier than current systems, where authorizations are not originated in enterprise modeling and are based on computer application perspectives. The need to define rights in this way has already been expressed a while ago [Moff88], but almost no commercial system follows this approach.

Future work includes development of more details and more formalization for the proposed approach. For example, this authorization structure can be described using objects. Another interesting aspect is the application of this approach to workflow systems [Kapp95].

References

[Fern93] E. B. Fernandez, M. M. Larrondo-Petrie, and E. Gudes, "Object-oriented database authorization: A method-based model", *Proc. OOPSLA Workshop on security of object-oriented systems, 1993.*

[Fern95a] E. B. Fernandez and R. B. France, "Formal specification of real-time dependable systems," *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems* '95, pp. 342-348.

[Fern95b] E. B. Fernandez, J. Wu, and M. H. Fernandez, "User group Structures in Object-Oriented Database Authorization," in *Database Security VIII: Status and Prospects*, J. Biskup et al., (Eds.), North-Holland Elsevier, 1995, pp. 57-76.

[Fern97] E. B. Fernandez and M. M. Anwar, "Using sequences of use cases and activities in object-oriented analysis", *FAU Technical Report TR-CSE-97-10*, February 1997.

[Ferr92] D. Ferraiolo and R. Kuhn, "Role-based access controls", Proc. *15th NIST-NCSC Nat. Computer Security Conf.*, NIST 1992, pp.554-563.

[Ferr95] D. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-Based Access Control (RBAC): Features and motivations", *Proc. 11th Comp. Sec. Applications Conf.*, 1995.

[Hawk97] J. C. Hawkins and E. B. Fernandez, "Extending use cases and interaction diagrams to develop distributed system architecture requirements", *FAU Technical Report TR-CSE-97-47*, May 1997.

[Henn96] P. Hennessey, R. Scheid, and J. R. Kirkley, III, "An integration framework for distributed systems", *Object Magazine*, March 1996, 36-42.

[Jaco92] I. Jacobson. *Object-Oriented Software Engineering*, Addison-Wesley, Reading, MA, 1992.

[Jaco95a] I. Jacobson and M. Christerson, "A growing consensus on use cases," *Journal of Object Oriented Programming*, March-April 1995, pp.15-19.

[Kapp95] G. Kappel et al., "Workflow management based on objects, rules, and roles", *Data Engineering*, vol. 18, No 1, March 1995, pp.11-18.

[Losa97] F. Losavio and A. Matteo, "Use case and multiagent models for object-oriented design of user interfaces", *Journal of Object-Oriented Programming*, May 1997, pp. 30-40.

[Moff88] J. D. Moffett and M. S. Sloman, "The source of authority for commercial access control", *Computer*, vol. 21,No 2, February 1988, pp. 59-69.

[Neum86] P. G. Neumann, "On hierarchical design of computer systems for critical applications," *IEEE Transactions on Software Engineering*, September 1986, pp. 905-920.

[Newm95] D. S. Newman, "Transforming information systems organizations through class-based reengineering", *Object Magazine*, March-April 1995, pp.43-49 and 87.

[Nyan94] M. Nyanchama and S. Osborn, "Access rights administration in role-based security systems", in *Database Security VIII*: *Status and Prospectus*, J. Biskup et al. (Eds.), North Holland-Elsevier, 1994, pp.37-56.

[Rumb9l] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Rumb94a] J. Rumbaugh, "Getting started: Using use cases to capture requirements", *Journal of Object-Oriented Programming*, September 1994, pp.8-12, 23.

[Rumb94b] J. Rumbaugh, "Modeling models and viewing views: A look at the model-view-controller framework", *Journal of Object-Oriented Programming*, May 1994, pp. 15-20 and 29.

[Rupi94] W. Rupietta, "Organization models for cooperative office applications", *Proc. DEXA '94*, pp.114-124.

[Sand96] R. S. Sandhu, E. J. Coyne, H.L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *Computer*, February 1996, pp.38-47.

[UML97] Unified Modeling Language notation, version 1.0, Rational Software Corp., Santa Clara, CA, 1997. http://www.rational.com