8.5 Role Rights Definition

'Least privilege' is a fundamental principle for secure systems. Roles can directly support the least privilege principle, but a systematic approach to assigning only the required rights to each role is required. This pattern provides a precise way, based on use cases, of assigning rights to roles to implement a least-privilege policy.

Example

Multitronics is a company that sells on-line digital media such as video, sounds, or images. They have been advised that for security reasons they should use a ROLE-BASED ACCESS CONTROL (249) approach, in which they can apply a least-privilege policy. For this they need to first identify the roles required to perform the business functions. In this system a manager administers the items on sale, deciding what is to be sold, at what prices, and so on. He can also order items for future sale. Subscribers register and create accounts so that they can purchase copies of digital items and download them to a mobile device such as a cellular phone. Subscribers can also reserve items not yet in stock. A salesperson maintains a catalog of items for sale and bills the subscribers for their purchases. To apply the required policy, we need a systematic way to assign rights to these roles.

Context

Applications composed of a variety of roles in which it is not easy to assign proper rights to the roles.

Problem

The ROLE-BASED ACCESS CONTROL (249) model is used now in many systems. However, the different component frameworks (.NET, J2EE) provide support only to define roles and to write authorization rules, and do not say anything about where the rights come from. It is not easy for system designers or for administrators to define the required roles and their corresponding rights.

How can we assign appropriate rights to the roles when we want to implement a least privilege policy?

The solution to this problem must resolve the following forces:

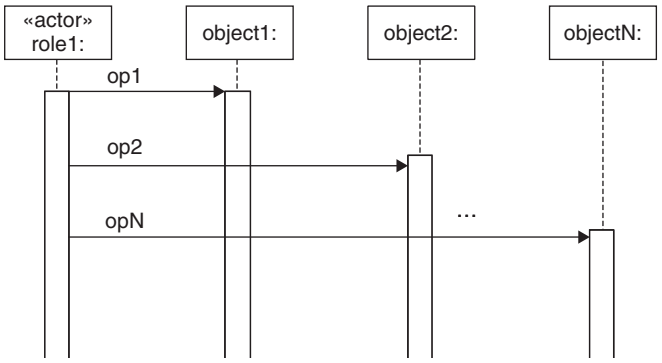
- Roles correspond to functional tasks in an organization, and we need to assign to these tasks sufficient rights to perform their work.

- Rights should be assigned according to the need-to-know (least privilege) principle, in which each role gets only the rights required to perform their duties.
- New roles appear and some roles may not be needed any more: changes to roles and their rights should be easy to perform.
- The assignment of rights should be independent of the system implementation.

Solution

Define the use cases of the system. The design of object-oriented systems always starts this way, but even systems that use other methodologies often define use cases as part of the requirements stage. As use cases define the interactions of actors with the system, we can interpret actors as roles. The roles that appear in a use case must be authorized for all the operations initiated by the role, or the role could not perform its functions. If we collect all the operations performed by a role over all use cases, they define the necessary rights for this role. To make this approach more detailed and systematic, we should build a use case diagram that displays all the use cases for the system, and sequence diagrams that show the interactions of roles with the system for each use case.

The figure below shows a generic sequence diagram indicating that actor role1 must use operations op1, op2, … opN to interact with the system. This means that role1 should be given the rights to apply these operations to the system.



Generic sequence diagram to obtain rights for a role

Implementation

Consider the following steps in order to implement the solution:

1. Start by building a use case diagram to display all the use cases of the system. The actors in this diagram correspond to roles and we can capture all the required roles in this way.

2. Build sequence diagrams for each use case. There is a sequence diagram for the main flow and a few more diagrams for alternate flows [Lar05].
3. Analyze all the sequence diagrams to see what operations the actors (roles) need to apply to interact with the system. These operations correspond to the role rights. In fact, these rights could be generated automatically from the use cases—tools such as Rational Rose can keep track of use cases, and they could be extended to generate the required authorization rules. One can also find all this information in the textual descriptions of the use cases, but it is harder to see the interactions, the sequence diagrams make the interactions more explicit.
4. From the use case exceptions the administrator implements the actions needed for security violations.
5. Addition or deletion of authorization rules is only necessary if a use case is added or deleted, or some of the actions of a use case are changed.

In a centralized system, authorizations could be enforced at the user interface, while in a distributed system, authorization could be enforced in a centralized system component such as the application server. Object-oriented systems use approaches based on model-view separation, for example the MVC or PAC architectures [POSA1]. These two models separate the conceptual model objects—a digital item 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.

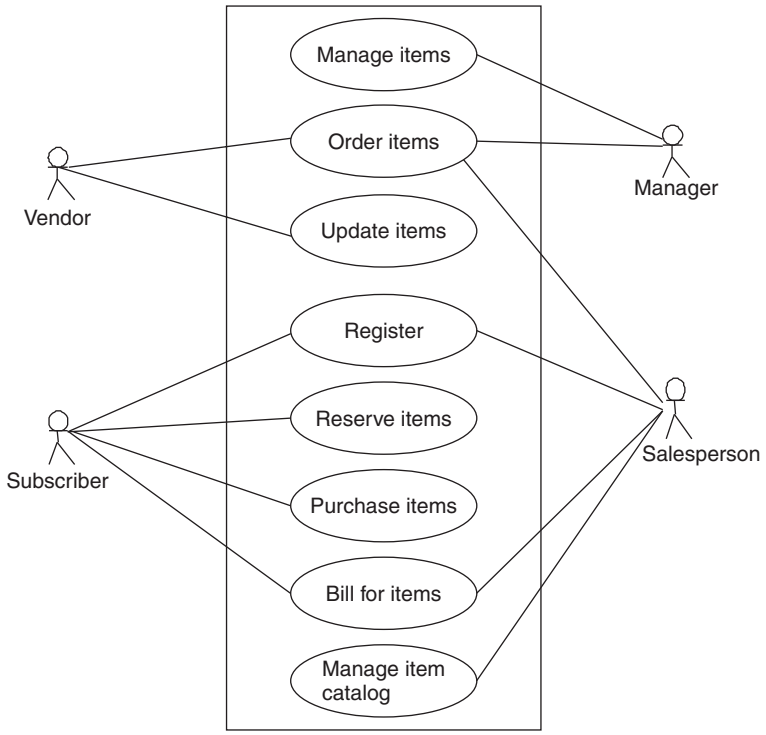
Sequences of use cases can be used to define a workflow that requires a specific set of authorizations for different roles. For example, a digital item can only be added by the vendor, released by the administrator, purchased, and downloaded by the subscriber, in that order. This complete workflow could be authorized as a unit.

Example Resolved

The figure below shows a use case diagram for the Multitronics on-line digital item vending system, including the roles defined earlier.

A subscriber participates in four use cases. Any user, once authenticated, has the right to register, but only registered users have the right to reserve and purchase items. These are all the rights needed for a subscriber in this system. A salesperson registers users in the system, bills users for their purchases, and maintains the catalog of products. He can also order new items from vendors according to customers' requests. A manager manages items and approves ordered items. Vendors have the right to upload the ordered items.

However, this is not the whole story. As indicated in the solution, a use case may include several actions that may be performed by different roles. To capture all the



Use case diagram for a digital item management system

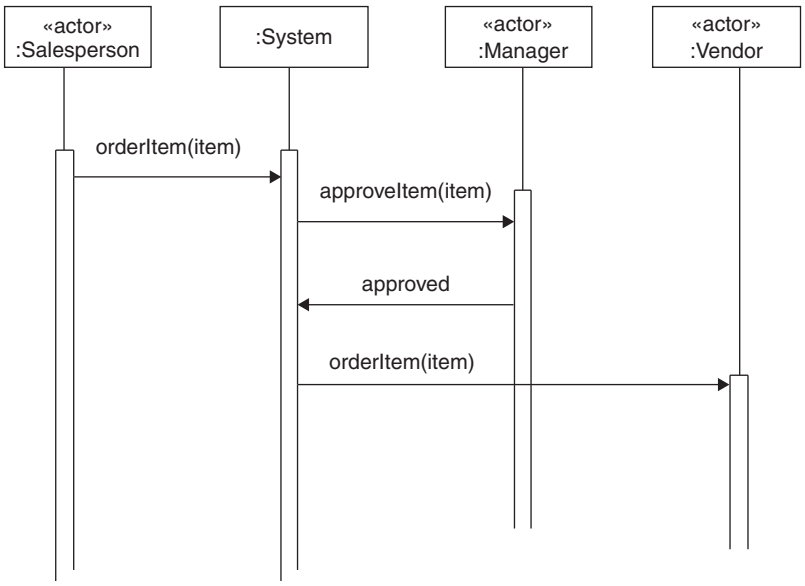
required rights, we need to look into the details of the use case or its corresponding sequence diagram. The figure below shows a sequence diagram to order an item. We can see that the salesperson initiates this use case and needs a right to order items. The manager has the right to approve the purchase, after which the salesperson has the right to send the order to the vendor.

The sequence diagram on page 263 shows the purchase of a digital item by a registered user.

From the two use cases shown, we can deduce that a salesperson role needs a right to order items, a manager role needs a right to approve orders, a registered subscriber role has the right to purchase and download an item. Sequence diagrams for the remaining use cases would provide the complete set of rights for all the roles.

Known Uses

Every complex object-oriented application using the RBAC model needs to define rights for its roles. Databases, for example Oracle, support user roles. Most modern frameworks, for example .NET and J2EE, support roles. Modern operating systems,



A sequence diagram for the use case Ordering a digital item

for example Trusted Solaris 7 and higher versions, also support roles. ROLE RIGHTS DEFINITION (259) indicates how to define rights for the roles in those systems.

Consequences

The following benefits may be expected from applying this pattern:

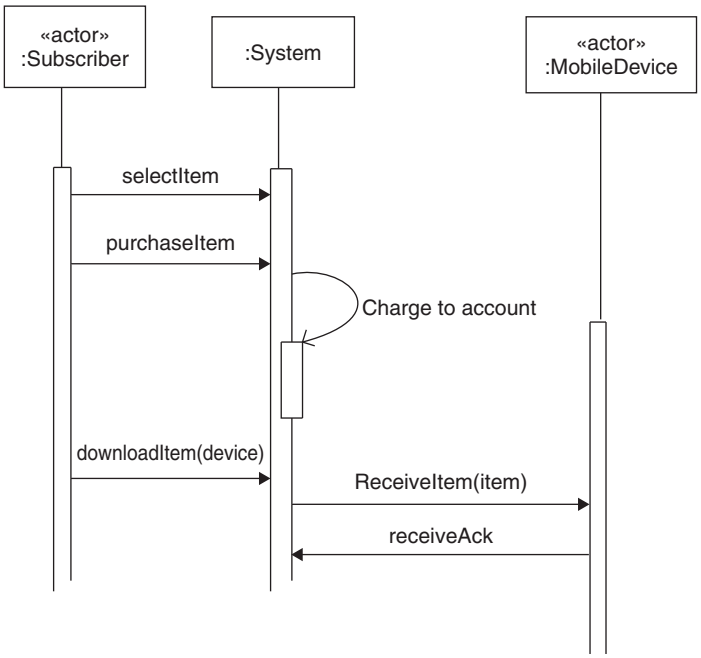
- Because roles correspond to functional tasks, their rights are defined according to the needs of the tasks.
- If these are the only rights given to the tasks, we have implemented a least privilege policy.
- Since all the use cases define all the interactions with the system, all the necessary rights can be generated in this way.
- A new use case just defines new rights that can be easily added to the existing set of rights.
- 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 long as the external view of the system does not change, there is no need to change authorization rules when the implementation changes. This is consistent with the information-hiding property of object-oriented systems.

The following potential liability may arise from applying this pattern:

- Building use cases requires specialized expertise, which may not be available in the organization.

See Also

ROLE-BASED ACCESS CONTROL (249) defines the structure of the security system, using classes such as `User`, `Role`, and others. ROLE RIGHTS DEFINITION (259) complements it, by providing a way to define the specific rights needed in a particular system.



Purchasing a digital item