10.8 File Authorization

This pattern describes how to control access to files in an operating system. Authorized users are the only ones that can use a file in specific ways. Apply AUTHORIZATION (245) to describe access to files by subjects. The protection object is now a file component that may be a directory or a file.

Example

Jim is an application programmer in a bank. He has a user account and some files in the bank's operating system. He realizes that the same system also stores files with customer data. These files have no authorization controls. Jim reads several of these files and finds customer information such as SSNs and credit card numbers. He uses this information to charge some items bought at mail-order shops.

Context

The users of operating systems need to use files to store permanent information. These files can be accessed by different users from different workstations, and access to the files must be restricted to authorized users who can use them in specific ways. Because of the needs of the organization, some (or all) of the files must be shared by these subjects. Use cases for a file system include creation and deletion of files, opening and closing of files, reading and writing files, copying files, and so on. A subject has a home directory for each authorized workstation, but the same home directory can be shared among several workstations or among several subjects. The home directory is used to search the files for which a subject has rights. Files are organized using directories, usually in a tree-like structure of directories and files. This facilitates the search for specific files.

Problem

Files may contain valuable information and access to them must be controlled carefully. In several recent attacks, hackers obtained lists of credit card numbers by accessing customer data illegally. Because files need to be shared, security becomes harder to enforce.

The solution to this problem must resolve the following forces:

- There may be different types of subjects, for example users, roles, and groups. The rights for users in groups or roles are derived from the group or role rights

(they are implicit rights). Groups of groups are possible, which makes deducing access rights even harder. All these subjects must be handled uniformly.

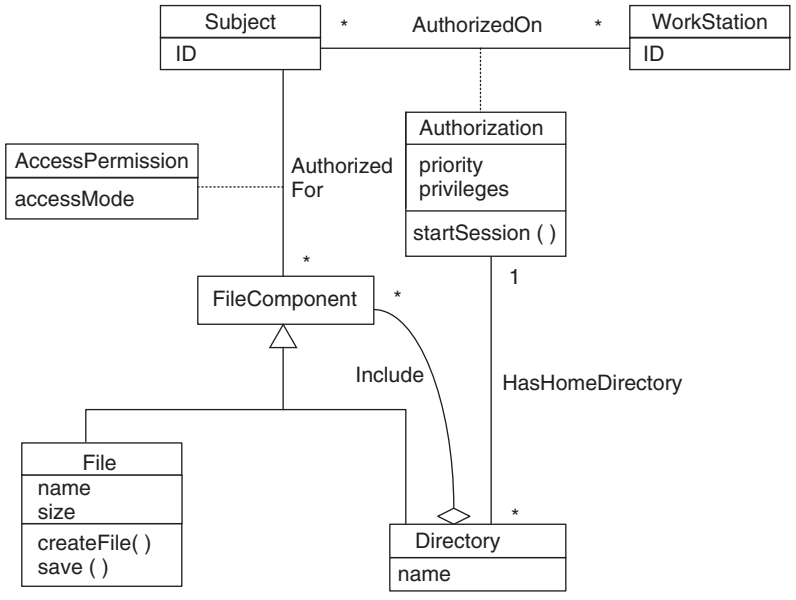
- Subjects may be authorized to access files or directories, and to exercise their file rights from specific workstations. To prevent illegal actions, we may need ways to apply these two types of authorization.
- Each operating system implements file systems in a different way. We need to abstract implementation details.
- Not all operating systems use workstations, groups, or roles. We need a modular system in which features not used can be cut easily from the model.

Solution

We apply AUTHORIZATION (245) first to describe access to files by subjects. Typically, file systems use Access Control Lists that are sets of authorizations. The protection object is now a file system component that may be a directory or a file. To reflect the fact that files may be accessed only from some workstations, we use AUTHORIZATION (245) again with the same subject and with workstations as protection objects. The tree structure of files and directories can be conveniently described by applying COMPOSITE [GoF95].

Structure

The figure below combines two versions of AUTHORIZATION (245) with COMPOSITE. File access is an extension of AUTHORIZATION (245) 1 by replacing `ProtectionObject`

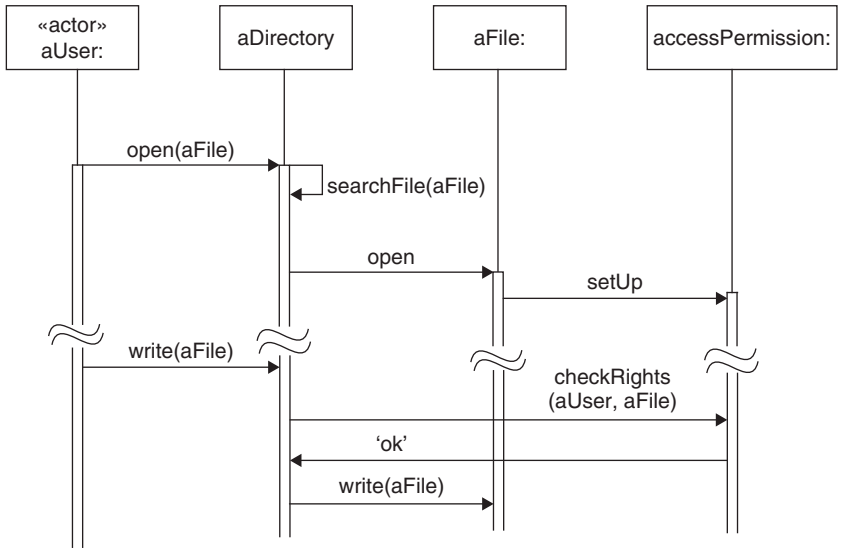


Class diagram for FILE AUTHORIZATION

by `FileComponent` and `Right` by `AccessControlListEntry` (ACLE). Workstation access is defined by a similar application of AUTHORIZATION (245).

Dynamics

The next figure shows the opening and writing of a file. A user actor opens the file, the directory locates it, and when found, opens it. Opening results in the file access permission being set up for future reference. When the user later tries to write to the file, their rights to write the file are checked and the write operation proceeds if authorized.



Sequence diagram for opening and writing to a file

Implementation

Typically directories are organized in a tree or directed graph structure [Sil03]. A file control block (FCB) describes the characteristics of each file, including its access permissions.

Example Resolved

A new operating system is installed that has authorization controls for its files. Now a need-to-know policy is set up in which only users that need access to customer files are given such access. This is the end of Jim's illicit activities.

Known Uses

This file system pattern can be found in most current operating systems such as Windows, Unix, and Linux. Not all of these systems uses all the concepts of the pattern.

Consequences

The following benefits may be expected from applying this pattern:

- The subjects can be users, roles, and groups by proper specialization of the `Subject` class. Roles and groups can be structured recursively [FP01]—that is, there can be role and group hierarchies that permit more flexibility in the assignment of rights.
- The protection objects can be single files, directories, or recursive structures of directories and files.
- Most operating systems use read/write/execute as access types, but higher-level types of access are possible. For example, a file representing students in a university could be accessed with commands such as list, order alphabetically, and so on.
- Implied authorization is possible: for example, access to a directory may imply a similar type of access to all the files in the directory [Fer94]. This approach allows an administrator to write fewer authorization rules, because some access rights can be deduced from others.
- Workstation access is also controlled and workstations can be homes for directories.
- This is a conceptual model that doesn't restrict implementation approaches.
- Workstation authorization is separated from file authorization, and systems that do not need workstation authorization can just ignore the relevant classes.
- In some operating systems, for example Inferno [Rau97], all resources are represented as files. Other systems represent resources by objects with ACLs. This means that this pattern could be used to control all the resources of the operating system.

The following potential liabilities may arise from applying this pattern:

- Implementations of the pattern are not forced to follow the access matrix model. For example, Unix uses a pseudo-access matrix that is not appropriate for applying the need-to-know policy. However, constraints can be added to the pattern to force all the instances of the pattern to conform to an access matrix model.
- Typically, access permissions are implemented as Access Control Lists (ACLs).

- [Gol99] and [Sil03]. These are data structures associated with a file in which each entry defines a subject that can access the file and its permitted access modes. The pattern models the entries of the ACLs but not the fact that they are associated with the file components.

Other aspects of using this pattern include:

- Some systems use the concept of Owner, who has all rights on the files they create. The Owner in this model corresponds to a special type of subject. When roles are used, there are no owners and when groups are used, ownership is not inherited in subgroups.
- In some systems, files are mapped to the virtual memory address space. The pattern still applies to this case, although a more uniform solution is then possible (see CONTROLLED VIRTUAL ADDRESS SPACE (339)).
- In some systems, the directory is not strictly a tree, because it is possible to have links between files in different subtrees [Sil03]. Modeling this case would require adding some associations to the model shown in the figure AUTHENTICATOR (324).

See Also

This pattern uses AUTHORIZATION (245). If roles are used, ROLE-BASED ACCESS CONTROL (249) is also relevant. The file structure uses COMPOSITE [GoF95]. It can use CONTROLLED EXECUTION ENVIRONMENT (346) for implementation.