A pattern language for security models

Eduardo B. Fernandez and Rouyi Pan
Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL 33431
ed@cse.fau.edu

Abstract
Security is a serious problem in the Internet and it is necessary to build new systems incorporating security as integral part of their design. The use of patterns is a good tool to help designers build secure systems. We discuss three patterns that correspond to the most common models for security: Authorization, Role-Based Access Control, and Multilevel Security. These can be applied in all the levels of the system and we show their use in the definition of a pattern for file authorization.

1. Introduction
Security is a very important aspect of any computing system, and has become a serious problem since institutions have opened their databases to the Internet. Most web systems in current use have not been designed with security in mind and patches have failed to make them more resistant to attacks. It is important to develop systems where security has been considered at all stages of design, which not only satisfy their functional specifications but also satisfy security requirements [Fer00a]. To do this we need to start with high-level models that represent the security policies of the institution. There are three models currently used by most systems: the access matrix, the Role-Based Access Control (RBAC) model, and the multilevel model.

Security encompasses all the architectural levels of a system [Fer99]. The Layers architectural pattern [Bus96] is therefore the starting point of this pattern language. This pattern provides a structure where we can define patterns at all levels that together implement a secure system. Its main idea is the decomposition of a system into hierarchical layers of abstraction, where the higher levels use the services of the lower levels. Here it provides a way to put things in perspective and to describe the mechanisms needed at each layer. We have discussed earlier [Fer99], why all these levels must be coordinated to assure security. Figure 1 shows the specific set of layers we consider. This figure shows some of the participants at each layer and their correspondence across layers.

Due to the complexity of the problem, we need a series of pattern languages, one per level. We start in this paper with a pattern language for abstract models; this includes the three basic models mentioned above. These models define security constraints at the highest architectural level for the applications, which are enforced by the lower levels. These models have been extensively studied by the security community [Pfl97, Sum97], and we do not attempt here to add new models or extend the

existing models. Our intent is to specify the accepted models as object-oriented patterns that can be used as guidelines in the construction of secure systems. We show also how these patterns can be applied to all the system levels.

We present first the Authorization pattern that describes access control for resources. We then describe in Section 3 the RBAC pattern as an extension of the Authorization pattern. The Multilevel Security pattern is shown in Section 4. After this we discuss how to apply the patterns to the lower levels and show a pattern for file access control as an example of a specialization of the Authorization pattern. We end with a discussion of general aspects common to these patterns.

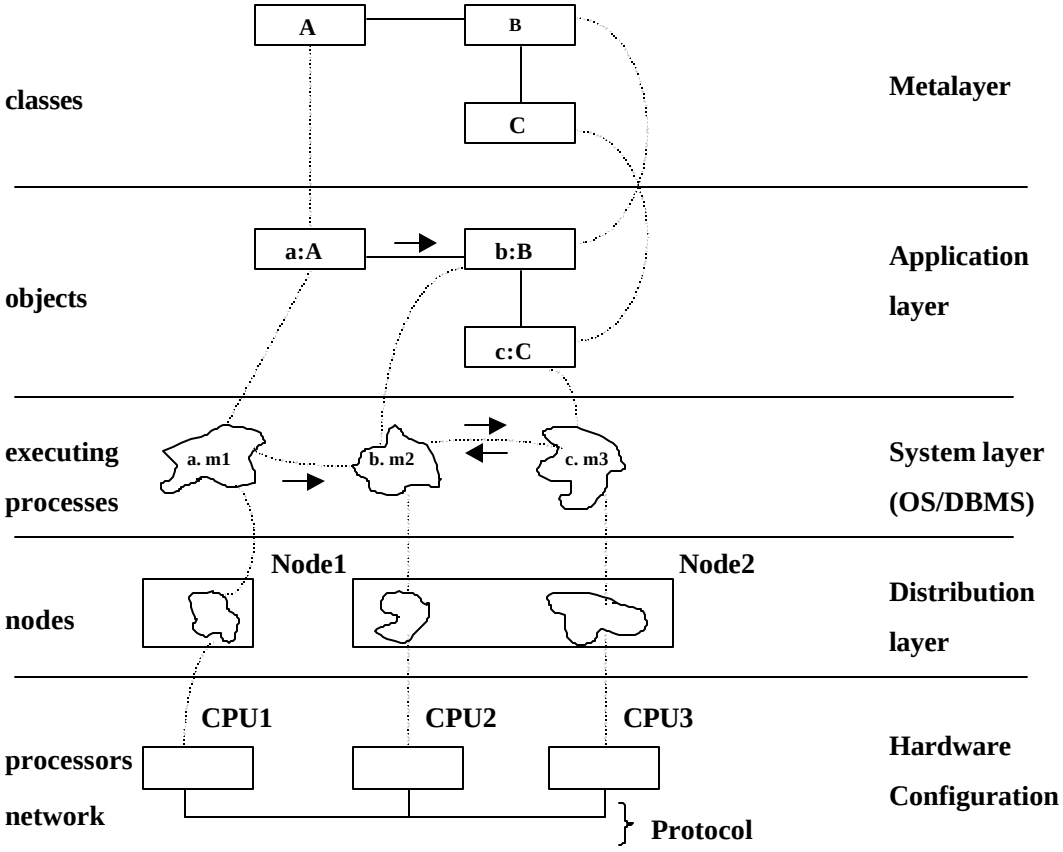

Figure 1. An instance of the Layers pattern

2. Authorization pattern

Context

Any computational environment where there are active entities that request resources whose access must be controlled.

Problem
How to describe allowable types of accesses (authorizations) by active computational entities (subjects) to passive resources (protection objects).

Forces
- The authorization structure must be independent of the type of resources, e.g., it should describe access by users to conceptual entities, access by programs to operating system resources, etc., in a uniform way.
- Predicates or guards may restrict the use of the authorization according to specific conditions.
- Some of the authorizations may be delegated by their holders to other subjects.

Solution
Represent the elements of an authorization rule as classes and associations. Class **Subject** describes the active entities, while class **ProtectionObject** describes the requested resource. An authorization rule is defined by an association between these two classes. An association class, **Right**, includes the type of access allowed (read, write,…), a predicate that must be true for the authorization to hold, and a copy flag that can be true or false indicating if the right can be transferred or not. An operation *check_rights* can be used in the subject or object to check the validity of a request. Figure 2 shows the elements involved.
The sequence diagram of Figure 3 shows a request validation. ActiveSubject is an actor, e.g., an executing process, that requests some resource. This request is intercepted by a Reference monitor.

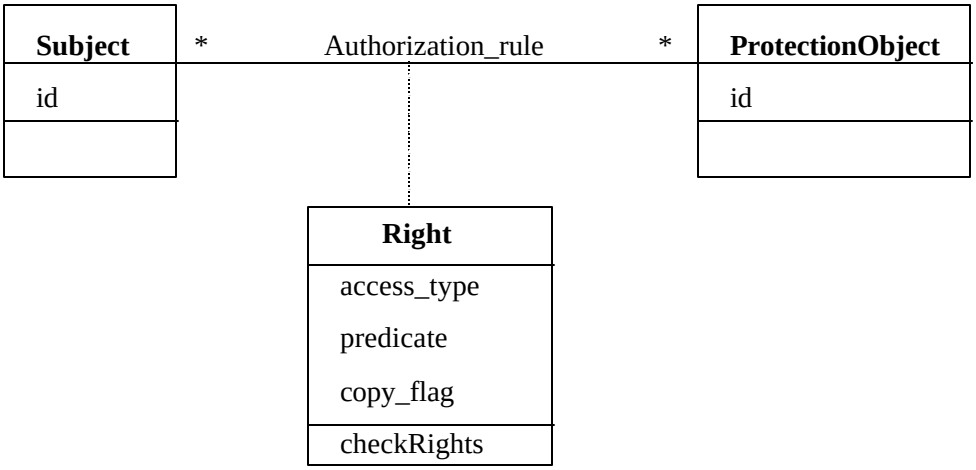

Figure 2. Authorization pattern

Consequences
The advantages of this pattern include:
- The pattern applies to any type of resource. Subjects can be executing processes, users, roles, user groups. Protection objects can be transactions, memory areas, I/O devices, files, or other OS resources. Access types can be read, write, execute, or methods in higher-level objects.

- The predicates in the rules are a general representation of any conditions that may restrict the application of a rule.
- The copy flag in the rule controls transfer of rights. However, some systems do not allow transfer of rights, i.e., their copy flags are always false.
- Some systems separate administrative authorizations from user authorizations for further security (principle of separation of duties) [Woo79].
- The request may not need to specify the exact object in the rule, this object may be implied by an existing protected object [Fer75]. Subjects and acess types may also be implied. This improves flexibility at the cost of some extra processing time (to deduce the specific rule needed).

Known uses

This model corresponds to the components of the access matrix, a fundamental security model [Sum97]. Its first object-oriented form appeared in [Fer93]. Subsequently, it has appeared in several other papers and products [Ess97, Kod01]. It is the basis for the access control systems of most commercial products, e.g., Unix, Windows, Oracle, and many others.

Related patterns

The RBAC pattern shown later is a specialization of this pattern. In Section 5 we show another specialization, a pattern for access control of files in operating systems.

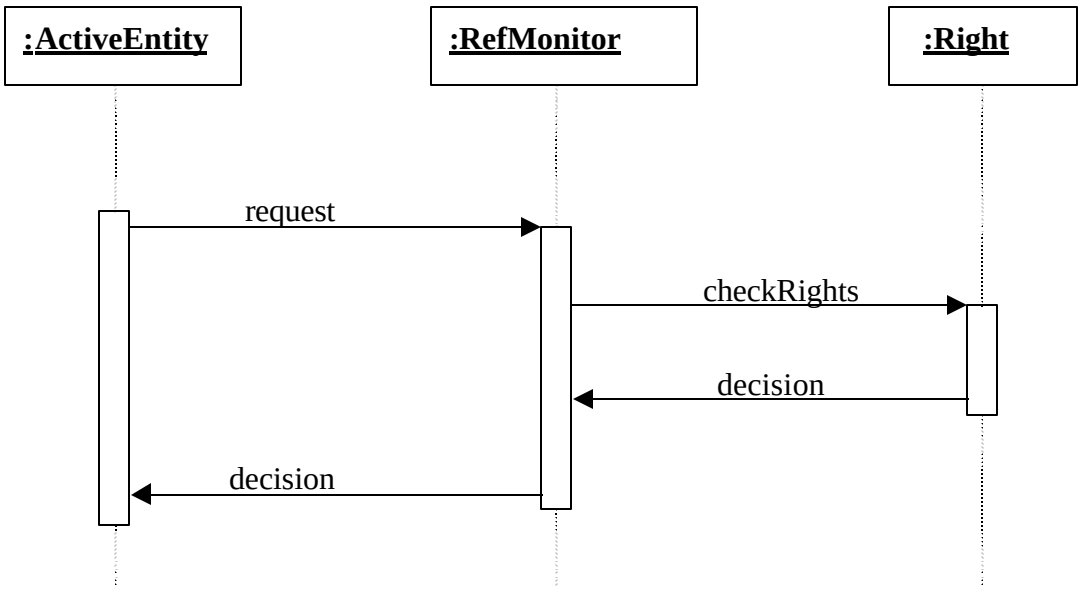

Figure 3. Request validation

3. Role-Based Access Control (RBAC) pattern

Context

Most institutions have a variety of job functions that require different skills and responsibilities. For security reasons users should get rights based on their job functions. This corresponds to the application of the need-to-know principle, a fundamental security policy [Sum97]. Job functions can be interpreted as roles that people play in performing their duties. In particular, web-based systems have a variety of users: company employees, customers, partners, search engines, etc.

Problem

How to assign rights to users according to their roles in an institution.

Forces

- People in institutions have different needs for access to information, according to their functions.
- We want to help the institution to define precise access rights for its members according to a need-to-know policy.
- Granting rights to individual users would require storing many authorization rules and it would also be hard for administrators to keep track of these rules.
- Users may have more than one role and we may want to enforce policies such as separation of duty, where a user cannot be in two specific roles in the same session.
- We may need to have hierarchies of roles, with inheritance of rights.
- A role may be assigned to individual users or to groups of users.

Solution.

Extend the idea of the previous pattern interpreting roles as subjects. A basic model for Role-Based Access Control (RBAC) is shown in Figure 4. Classes **User** and **Role** describe the registered users and the predefined roles, respectively. Users are assigned to roles, roles are given rights according to their functions. The association class **Right** defines the access types that a user within a role is authorized to apply to the protection object. In fact, the combination Role, ProtectionObject, and Right is an instance of the Authorization pattern. Accordingly, the predicate indicates content-dependent restrictions that can be used to select specific objects. The attribute copy_flag indicates a Boolean value (true/false).

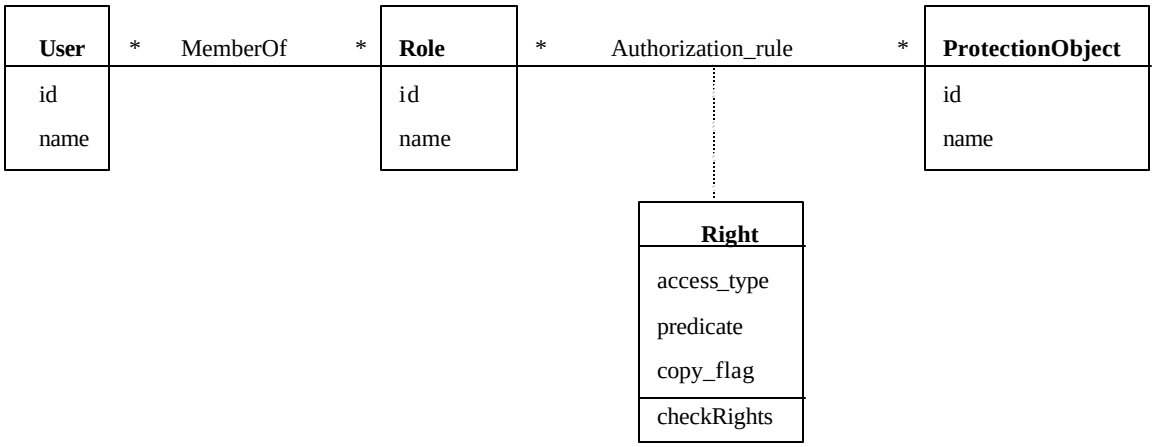

Figure 4. Basic RBAC pattern

The model of Figure 5 considers additionally composite roles (an application of the Composite pattern) and separation of administration from other rights (an application of the policy of separation of duties). The model also includes a constraint to enforce a separation of duties. The administrator has the right to assign roles to groups and users, he is a special user who can assign users to roles and rights to a role. Rights for security administration usually include:

- Definition of authorization rules for roles.
- Creation/deletion of user groups.
- Assignment of users to roles.

Figure 5 also includes the concept of **Session**, that corresponds to the way to use a role and that can be used to enforce role exclusion at execution time. Finally, class **Group** describes groups of users that can be assigned to the same role.

Consequences
Among the advantages of this pattern we have:
- It allows administrators to reduce the complexity of security, there are much more users than roles.
- Institution policies about job functions can be reflected directly in the definition of roles and the assignment of users to roles.
- Roles can be structured for further flexibility and reduction of rules.
- Users can activate more than one session at a time for functional flexibility (some tasks may require multiple views or different types of actions).
- We can add UML constraints to indicate that some roles cannot be used in the same session or given to the same user (separation of duty).
- Groups of users can be used as role members, thus further reducing the number of authorization rules and the number of role assignments.

Possible disadvantages include:
- Additional conceptual complexity (new concept of roles, assignments to multiple roles,…).

There are other possible structurings of roles [Fer94], which may be useful for specific environments. It is also possible to use roles to extend the multilevel model of Section 5.

Known uses
Our pattern represents in object-oriented form a model described in set terms in [San96]. That model has been the basis of most research papers and implementations of this idea [FBK99]. RBAC is implemented in a variety of commercial systems, including Sun's J2EE [Jaw00], Microsoft's Windows 2000, IBM's WebSphere, and Oracle, among others. The basic security facilities of Java's JDK 1.2 have been shown to be able to support a rich variety of RBAC policies [Giu99].

Related patterns
A simpler version (corresponding to Figure 3), appeared in [Fer93] and [Yod97], and a pattern language for its software implementation appears in [Kod01] (this doesn't consider composite roles,

groups, and sessions). The pattern of Figure 5 includes the Authorization pattern and the Composite pattern. Other related patterns are the Role pattern [Bau00], and the Abstract Session [Pry00].

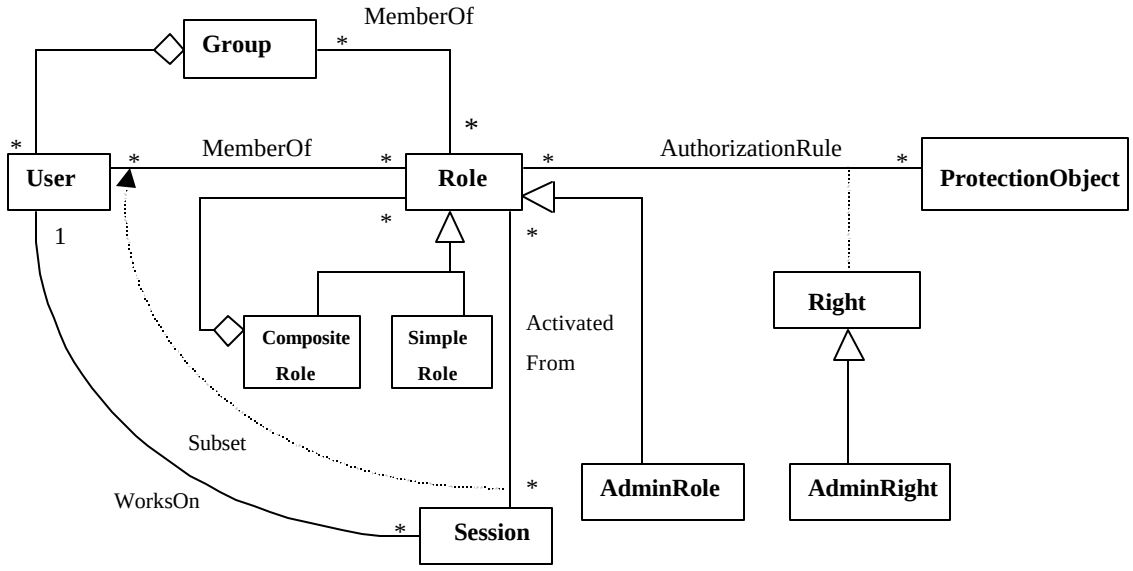

Figure 5. A pattern for RBAC

4. Multilevel Security pattern

Context

In some environments data and documents have sensitivity levels, e.g., secret, confidential. Users have clearances and can access documents based on their clearances.

Problem

How to decide access in an environment with security classifications.

Forces

- The model should protect the confidentiality and integrity of data based on its sensitivity.
- The model should be able to be used at any architectural level.
- There could be different sets of rules to decide access.
- There must be a convenient way to assign users and data to classification levels.

Solution

In this model users and data are assigned classifications or clearances. Classifications include levels (top secret, secret,…), and compartments (engDept, marketingDept,…). For confidentiality access of users to data is based on rules defined by the Bell- LaPadula model, while for integrity the rules are defined by Biba's model [Sum97]. Figure 5 shows its basic structure.

However, the basic model does not allow assignment to levels, a special set of trusted processes is needed to assign levels and modify classifications; that is, for all administrative functions.

Consequences
Advantages include:
- The classification of users and data is relatively simple and can follow institution policies.
- This model can be proved to be secure under certain assumptions [Sum97].

Possible disadvantages include:
- Implementations should use labels in data to indicate their classification. This assures security, if not done the general degree of security is reduced.
- We additionally need trusted programs to assign users and data to levels.
- Data should be able to be structured into hierarchical sensitivity levels and users should be able to be structured into clearances. This is usually hard to do or impossible in commercial environments.

Known uses
The model has been used by several military-sponsored projects and in a few commercial products, including DBMSs (Informix) and operating systems (Pitbull [Arg] and HP's Virtual Vault [HP]).

Related patterns
The concept of roles can also be applied here, role classifications can replace user classifications.

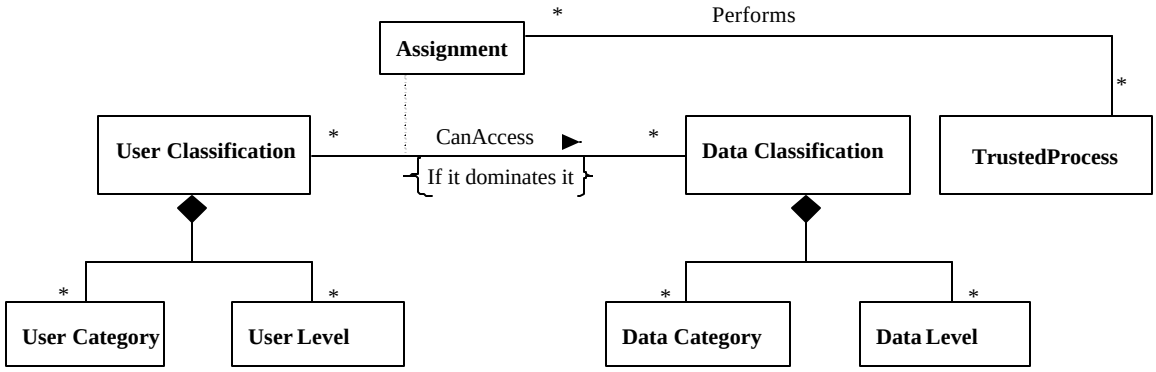

Figure 6. Class model for the Multilevel Security pattern

5. The lower levels

These patterns define abstract models and can be applied to all the levels of the system. At each level we can describe more concrete patterns that express the concepts of the abstract model in terms of

concrete entities of the level. To have a secure system it is necessary to define pattern languages for each level. This is left for future work. We show here an example.

The Authorization pattern of Section 3 can be applied to operating system users accessing files from workstations. This is a pattern for the system layer of Figure1.

5.1 File Authorization pattern

Context

The users of operating systems need to define files. These files can be accessed from different authorized workstations and access to the files should be restricted to authorized users.

Forces

The following forces apply to this case:

- There may be different categories of subjects, e.g., users, roles, and groups.
- Subjects may be authorized to access files, directories, and workstations.
- A subject has a home directory for each authorized workstation, but the same home directory can be shared among several workstation or among several subjects.
- Users may be grouped for access.
- Some systems may use roles instead or in addition to users as subjects.
- There are different implementations for the file systems of operating systems.

Solution

Specialize the Authorization pattern by replacing objects by files and rights for access permissions (Figure 7). This defines file access, while workstation access is defined by a similar application of the Authorization pattern. Files and directories are recursively structured. The result is a Semantic Analysis pattern (SAP), which combines two versions of the Authorization pattern with a Composite pattern. A SAP is an analysis pattern corresponding to a basic set of Use Cases [Fer00].

Consequences

Possible advantages of this pattern are:

- It can accommodate a variety of subjects, including users, roles, and groups.
- The access objects can be single files, directories, or recursive structures of directories and files.
- Implied authorization is possible; for example, access to a directory ay imply similar type of access to all the files in the directory.

Some disadvantages are:

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. Constraints could be added to the pattern to force all the instances of the pattern to conform to an access matrix model.

Other aspects include:

- Some systems may not use authorizations for workstations.
- Most operating systems use read/write/execute as access types. Higher level types of access are possible.

- In most operating systems there is the concept of owner, a special type of user with all rights on the files he creates.
- In some systems, files are mapped to the virtual memory address space. The pattern still applies to this case.

Known uses

This pattern represents the file systems of Unix, Windows, Linux, and most current operating systems.

Related patterns

This is a specialization of the Authorization pattern. If roles are used then the Role-Based access Control pattern is relevant. The file structure uses a Composite pattern.

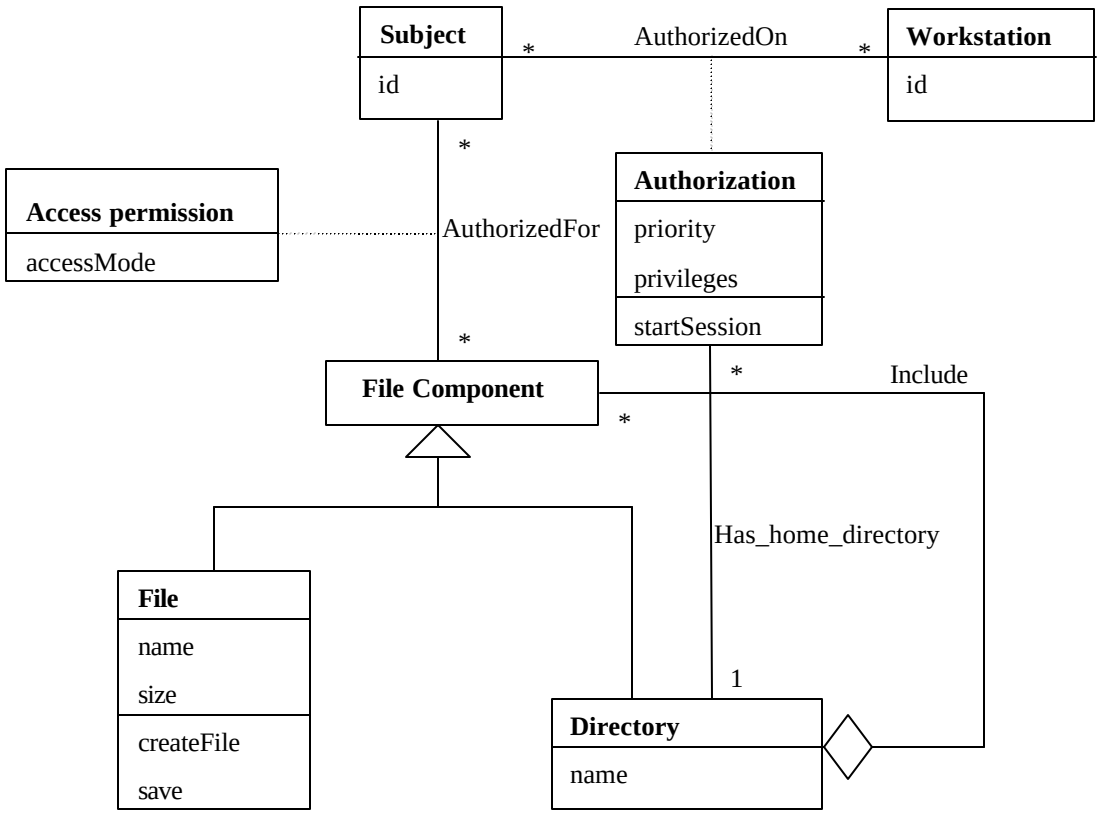

Figure 7. A pattern for file authorization

6. General discussion of the patterns

The actual implementation of these patterns depends on the architectural level where they are applied. An important issue here is how to control acess with low overhead. The Authorization pattern can be

implemented using Access Control Lists (ACLs), as is done in most operating systems. Another implementation approach is the use of capabilities, as it has been done at the hardware and operating system level to control resources. A recent paper describes their application to control access to classes [Fra99]. The use of metaclasses and reflection is another interesting way to implement these models [Wel99]. Requests for resources must be intercepted and validated with the information in the ACLs or capabilities; this requires a special program (another pattern) known as the Reference Monitor [Sum97]. A possible implementation of the Reference Monitor is the Interception pattern of POSA2 [Sch00].

Patterns for some of the architectural levels have appeared:

- Yoder and Barcalow [Yod97] describe the Single Access Point (a general secure system design pattern), Check Point (a kind of Reference monitor), Roles (as indicated in Section 3), and others.
- Fernandez discusses the relation between metadata and patterns [Fer00a]. The model for RBAC of Section 3 (Figure 5), appeared first here.
- Cryptographic patterns are described in [Bra00].
- A general discussion of security patterns including some network security patterns is presented in [Sch01].
- A framework for access control and filtering of distributed objects, combining several patterns, is described in [Hay00].
- Several security patterns for Java appear in [Jaw00].

Work is needed to add more patterns in each level and to collect and unify these patterns.

There are other security models, based on other policies or combinations of policies. Two important models are the Clark-Wilson model and the Chinese Wall model [Sum97]. The Clark-Wilson model emphasizes integrity and defines two basic principles:

- Well-formed transactions—to assure that changes to the data leave it in a consistent state.
- Separation of duty—changes must be approved by at least two parties.

The Chinese Wall is more properly a policy than a model. Information is grouped into conflict of interest classes and a person is allowed at most one set of information in each class.

There is a pattern for the Clark-Wilson model [Wel99], but none for the Chinese Wall model.

Acknowledgements

We thank our shepherd Jeff Barcalow for his insightful remarks that improved this paper considerably.

References

[Arg] Argus Systems Group, “Trusted OS security: Principles and practice”, http://www.argus-systems.com/products/white_paper/pitbull

[Bau00] D.Baumer, D. Riehle, W. Siberski, and M. Wolf, “Role Object”, Chapter 2 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. of PLoP’97*, http://jerry.cs.uiuc.edu/~plop/plop97

[Bra00] A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software", Chapter 16 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. of PLoP'98*, http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal., *Pattern- oriented software architecture*, Wiley 1996.

[Ess97] W. Essmayr, G. Pernul, and A.M. Tjoa, "Access controls by object-oriented concepts", *Proc. of 11th IFIP WG 11.3 Working Conf. on Database Security*, August 1997.

[FBK99] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn, "A Role-Based Access Control model and reference implementation within a corporate Intranet", *ACM Transactions on Information and System Security*, Vol. 2, No. 1, Feb. 1999, pages 34-64 .

[Fer75] E. B. Fernandez, R. C. Summers and T. Lang, ``Definition and evaluation of access rules in data management systems," *Proceedings 1st International Conference on Very Large Databases*, Boston, 1975, 268-285.

[Fer93] E.B.Fernandez, M.M.Larrondo-Petrie and E.Gudes, "A method-based authorization model for object-oriented databases", *Proc. of the OOPSLA 1993 Workshop on Security in Object-oriented Systems* , 70-79.

[Fer94] - E. B. Fernandez, J. Wu, and M. H. Fernandez, ``User group structures in object-oriented databases", *Proc. 8th Annual IFIP W.G.11.3 Working Conference on Database Security*, Bad Salzdetfurth, Germany, August 1994.

[Fer99] E.B.Fernandez, "Coordination of security levels for Internet architectures", *Procs. 10th Intl. Workshop on Database and Expert Systems Applications*, 1999, 837-841.

[Fer00] E.B. Fernandez and X. Yuan, "Semantic Analysis Patterns", *Procs. of the 19th Int. Conf. on Conceptual Modeling (ER2000)*, 183-195.

[Fer00a] E.B. Fernandez, "Metadata and authorization Patterns", Report TR-CSE-00-16, Dept. of Computer Science and Eng., Florida Atlantic University, May 2000.

[Fra99] G. Frascadore, "Java application server security using capabilities", *Java Report*, March 1999, 31-42.

[Giu99] L. Giuri, "Role-Based Access Control on the web using Java", *Procs. of RBAC'99*, ACM 1999, 11-18.

[Hay00] V. Hays, M. Loutrel, and E.B.Fernandez, "The Object Filter and Access Control framework", *Procs. Pattern Languages of Programs (PLoP2000) Conference*, http://jerry.cs.uiuc.edu/~plop/plop2k

[HP] Hewlett Packard Corp., Virtual Vault, http://www.hp.com/security/products/virtualvault

[Jaw00] J. Jaworski and P.J. Perrone, *Java security handbook*, SAMS, Indianapolis, IN, 2000.

[Kod01] S. R. Kodituwakku, P. Bertok, and L. Zhao, "A pattern language for designing and implementing role-based access control", *Procs. KoalaPLoP 2001*.

[Pfl97] C.P. Pfleeger, *Security in computing*, (2nd Ed.), Prentice-Hall 1997.

[Pry00] N. Pryce, "Abstract session: An object structural pattern", Chapter 7 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. of PLoP'97*, http://jerry.cs.uiuc.edu/~plop/plop97

[San96] R. Sandhu et al., "Role-Based Access Control models", *Computer* , vol. 29 , No2, February 1996, 38-47.

[Sch00] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-oriented software architecture, vol. 2 , Patterns for concurrent and networked objects*, J. Wiley, 2000.

[Sch01] M. Schumacher and U. Roedig, "Security engineering with patterns', submitted to PLoP 2001.

[Sum97] R. C. Summers, *Secure Computing: Threats and Safeguards*, McGraw-Hill, 1997.

[Wel99] I. Welch, "Reflective enforcement of the Clark-Wilson integrity model", *Procs. 2nd Workshop in Distributed Object Security*, OOPSLA'99, ACM, November 1999, 5-9.

[Woo79] C. Wood and E. B. Fernandez, ``Authorization in a decentralized database system," *Proceedings of the 5th International Conference on Very Large Databases*, 352-359, Rio de Janeiro, 1979.

[Yod97] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security". *Procs. PLOP'97*, http://jerry.cs.uiuc.edu/~plop/plop97 Also Chapter 15 in *Pattern Languages of Program Design*, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Addison-Wesley, 2000.