Design Patterns for Role-Based Access Control

Martin Lason*
Martin.Lason@vsb.cz

Roman Szturc*
Roman.Szturc@vsb.cz

Abstract: The correct specification of user's permissions and their association to roles are significant assumptions of effective security management. This paper is aimed at early stage of roles definition process when first users' roles and privileges are defined. The paper introduces several schemes that could help security administrators to identify roles and their privileges. These schemes should be treated as design patterns in software development.

Keywords: RBAC, design patterns, security.

1 Introduction

Users have to log in to information system that contain data with restricted access in order to ensure security and privacy. In 1980s most systems had their own password database. As number of systems grows, user has to remember more and more passwords for different systems. In order to avoid this problem many systems have started to use LDAP (Lightweight Directory Access Protocol) for purpose of authentication in 1990s. Thanks to that users can share one user name and password among several systems. Still, there is a lot of systems unable to share an LDAP database of users accounts and permissions, for many reasons. Furthermore users have to enter passwords repeatedly due to authorization. Solution of this problem is called Single sign-on. Single sign-on (SSO) is mechanism whereby a single action of user authentication and authorization can permit a user to access all computers and systems where he has access permission, without the need to enter multiple passwords [8].

However next issue is authorization management. In a separated system users have sets of privileges for each system but application of an SSO enables centralization of their privileges. Using of RBAC greatly simplifies management of these privileges. Many companies and organizations use identity management to manage user accounts and their privileges. This includes, in addition to other functions, granting of access privileges to users according to their role in organization. The great advantage of such a solution is its ability to define global roles which determine users' privileges in the whole organization.

Although the most of users have the same or very similar privileges, definitions of their roles arise in most cases individually. Each situation is usually solved ad hoc with minimal connection to existing models of roles. Mapping of user's privileges to its roles and specification of relations among the roles are thus performed without any help of templates. The templates could help to solve common situations significantly.

In this paper we will try to define several basic schemes of roles that describe common problems and that could help to solve them in a uniform way. This work is aimed to context-aware role switching because it describes reality more precisely. Since assigning users to roles

* Department of Computer Science FEI VSB-TU Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba

and permissions to roles is time consuming task, usage of general patterns will reduce costs of development.

2 RBAC

Role-based access control (RBAC) is a powerful security model for the authorization management. The central notion of RBAC is that users do not have discretionary access to enterprise objects. Instead permissions are administratively associated with roles, and users are administratively made members of appropriate roles, thereby acquiring the roles' permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications.

Core RBAC recognizes five administrative elements: users, roles and permissions, where permissions are composed of operations applied to objects [3]. Figure 1 shows a pair of binary relations: one between Operation and Objects, referred to as a Permission, and the other between Role and permission.

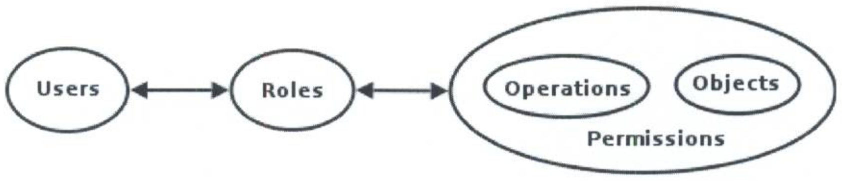

Fig. 1: RBAC elements relationships

2.1 Parametrized Roles

RBAC provides a mean of expressing access control in associating privileges with roles that is scalable to large numbers of principals. However, pure RBAC associates privileges only with roles, whereas applications often require more fine-grained access control. Parameterized roles extend the functionality to meet this need.

We have used parameterized roles in our models, because we need to satisfy access control requirements that are context-sensitive in large-scale systems. Hence we used OASIS model [2] to extend basic role model with parameters. There are also others models that are able to handle with parameters (e.g. [6]). But we chose OASIS model because in OASIS it is possible to test context predicates during role activation as well as at the time of access. Another reason why we chose OASIS is that there exists an implementation of OASIS based on XML Schema and Simple Object Access Protocol (SOAP), which provides access control enforcement for SOAP services [2]. Access control decision can be based on the actual parameter values of a SOAP request.

There is a formal approach using logic because adding parameters to privileges and roles adds a layer of complexity to the model. It is crucial, that all parameters are typed. Parameters may be of four types: the set of all roles, the set of all appointment certificates, the set of all privileges and the set of all environmental constraints. Environmental predicates are implemented by computational procedures such as database lookup that can be invoked when a rule is interpreted.

3 RBAC model in UML

There is a need of a reference model for our intended schemes of roles. We have chosen SecureUML model [1] to define RBAC elements using UML. Figure 2 presents the metamodel that defines the abstract syntax of SecureUML. The types User, Role, and Permission and the relations UserAssignment, PermissionAssignment, and RoleHierarchy are directly adopted from the proposed RBAC standard [4]. An AuthorizationConstraint is a logical predicate that is attached to a permission by the association ConstraintAssignment and makes the permission's validity a function of the system state. The AuthorizationConstraint is used by Context and Protectable interfaces that are described later. The types Resource and Action roughly correspond to the RBAC terms Operation and Object in [4] and integrate these elements into system modeling languages.

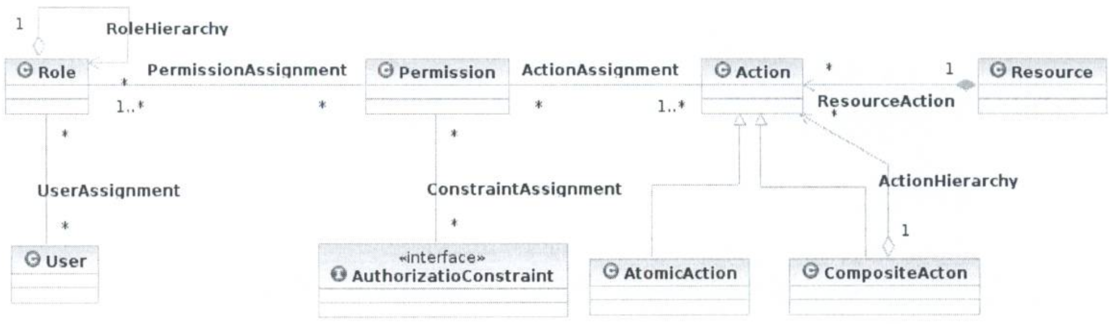

Fig. 2: SecureUML Metamodel

4 Patterns for RBAC

In this section we try to find abstraction imitating repeated parts of security models. As a result we will show some templates that should describe general solutions of common problems. The template (that could be compared to design pattern in software development) should help security administrator to define roles and create relationships between them and possible associations to users and permissions.

The main idea is based on the fact that there have to be plenty of similar activities that could be modeled in a uniform way. Such abstract solution will solve common problems that occur during definition of system's roles. Many of such roles in real life are context aware. *Context* plays important role in our template because it helps to characterize situations in real systems. Central idea is that user's functions differ in different contexts. Specifically user has different privileges in different contexts. These contexts are formally defined as parameters of user's role.

For example user may have permission to store data to a database. But it could be very dangerous to assign such permission to all users that need to make changes in database without any restriction. More secure solution is usage of permission that are restricted by context in which user is active. In particular user will gain permission to write to database, but only to her own entries or to entries of her department. The context may be also an application that is used to access the resource. Protected resource may be accessible only from selected application that are trusted.

4.1 Composit pattern

Let's start with very simple pattern. It is used in cases when user's roles are context aware and the context is some type of hierarchical structure. It should be noticed that it does not refer to groups of users. These groups are not related to roles because they are related to model of users.

Problem

Roles in the system are context aware and there exist many contexts. These contexts are represented by appropriate role parameters. There is a need to organize big amount of parameters in some way. Additionally the system should be able to handle with similar parameters uniformly.

Example 1. Suppose we are modeling project management system. Members of development team have access to some parts of a project according to their role. Each project consists of several independent parts assigned to different developers. Some developers have roles that include permission to read some kind of project information. It means that users will gain permission to read project's data. But in the pure RBAC model, user will gain permission to read data of all projects! Therefore permission to read project data are parameterized by particular project that is accessed. Additionally suppose that users have access to the project by means of Web services.

Example 2. Another example where hierarchical parameters could be met is parts of organization. Each organization can be usually divided in some organization units. For example university consists of faculties which consist of departments. We want to use RBAC model for granting privileges to persons that works in some part of the university. For example secretary role should have permission to change timetable. But if we use parameter, secretary will be able to change timetable only for department to which she is assigned to.

Solution

Solution of previous example is depicted in figure 3. The Context interface represents hierarchical parameter of the role. Users are assigned to roles based on their competencies and permissions are assigned to their roles. Each role can have parameters that are specifies context of permissions. The goal is to compose parameters that could serve as a context, into tree structure to represent part-whole hierarchies. Context's structure uses the *Composit* design pattern [5]. Context represents primitive objects—role parameters. Each parameter can be either instance of Leaf or instance of Composit. Leaf represents leaf of a tree in the context's hierarchy (e.g. particular indivisible organization unit) and Composit represents node that contains a collection of parameters.

The main idea of the Composite pattern is to provide an uniform interface to role parameters represented by different types in the same hierarchy, where instances are all components of the same composite complex object. For example, faculty (represented by Composit) consists of departments (represented by Leaf). It could have common interface OrganizationUnit (represented by Context) that enables to handle this role uniformly regardless of particular instances of the hierarchy. Employees that have permission to use network printer could be restricted by the context of their department because they can use only those printers that are associated to the same department as employee's one. All types of the context are represented by the same interface mentioned as a OrganizationUnit.

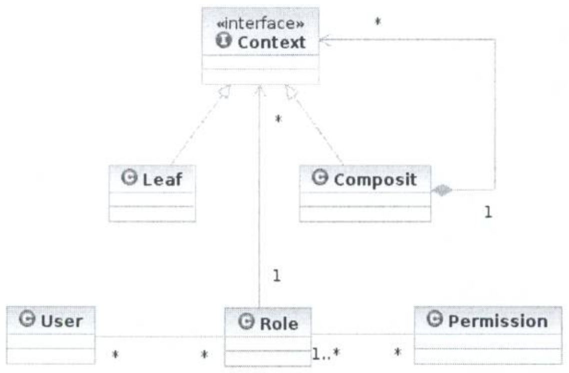

Fig. 3: Composit as a context

4.2 Proxy pattern

The name of this pattern originates from the Proxy design pattern. The aim is to provide a surrogate or representative for another object to control access to it.

Problem

Example 3. Typical example may be a web application (e.g. developed in PHP) that uses a database. Application users cannot access the database directly, but implementation provides a special user that has a permission to access the database. Requests of all users are controlled by application and go through the special user only. The special user acts as a *proxy* and fully controls access to the database.

The previous example is applied very frequently nowadays, but with growing popularity of Web services there is analogical approach defined. The problem is described as a Trusted Subsystem pattern [7]. Let's suppose a following situation. A client needs to access one or more Web services that are distributed across a network. The Web services are designed so that access to additional resources (such as databases or other Web services) is encapsulated in the business logic of the Web service. These resources must be protected against unauthorized access. The goal is to ensure that the client that is used to access the Web service cannot access the additional resources directly. The situation is depicted in figure 4.

Example 4. The bank client wants to change his account properties. On personal level the client has to contact bank clerk to change some data in his account. The clerk verifies his personal identity and new data and then she makes a change of his account. The clerk represents in this example proxy that restricts access to bank resources to ensure correctness of the data. Let us note that the clerk uses an bank application that acts as a second proxy.

Example 5. We can find another example in some systems. Secure systems don't allow to read users' passwords to anybody. There are only functions that compare passwords to database and

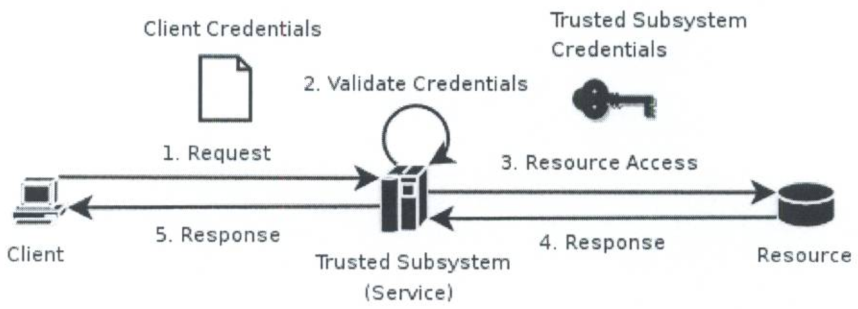

Fig. 4: Web services – Trusted Subsystem

that change password to a new one. User that wants to change her password has to enter her current password to prove her identity, in other words she sends the credentials that identifies herself. After that system verifies user's privilege to change password and if everything is correct, it sends request to protected password database. In this case system acts as a proxy that needs user's credentials to perform operations over protected resource.

Solution

Basic principles of the solution of this problem is depicted in figure 5. The RBAC elements such as users, roles and permissions have the same meaning as in previous occurrence. The central notion of this solution is separation of two types of roles: UserRole and Proxy. Proxy role has privileges to access protected resource. The resource is accessible via Protectable interface. Protectable acts in this case as a role parameter. Finally users assigned to role Proxy can access all object that implements interface Protectable in object oriented approach. The model implementation should use *appointments* that are described in detail in [2] for credentials representation.

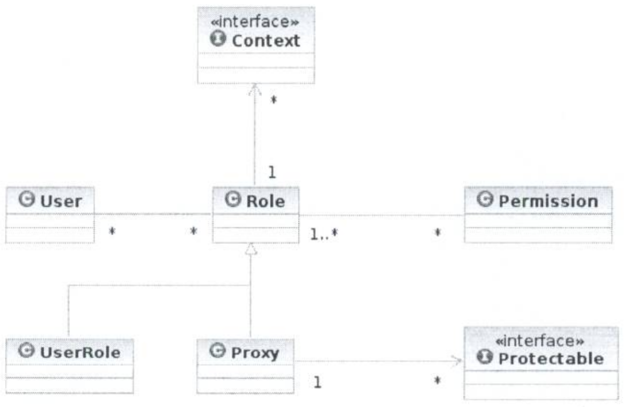

Fig. 5: Proxy

Interaction diagram depicted in figure 6 shows Trusted Subsystem example. The solution was implemented by Web service technology. We add RBAC model to the solution by using Proxy pattern. A person (represented by object alice) wants to use resource that cannot be accessed directly. She uses client that is associated to UserRole. UserRole contains privileges to send user's credentials (in RBAC model described as *Appointments*) to service (Proxy).

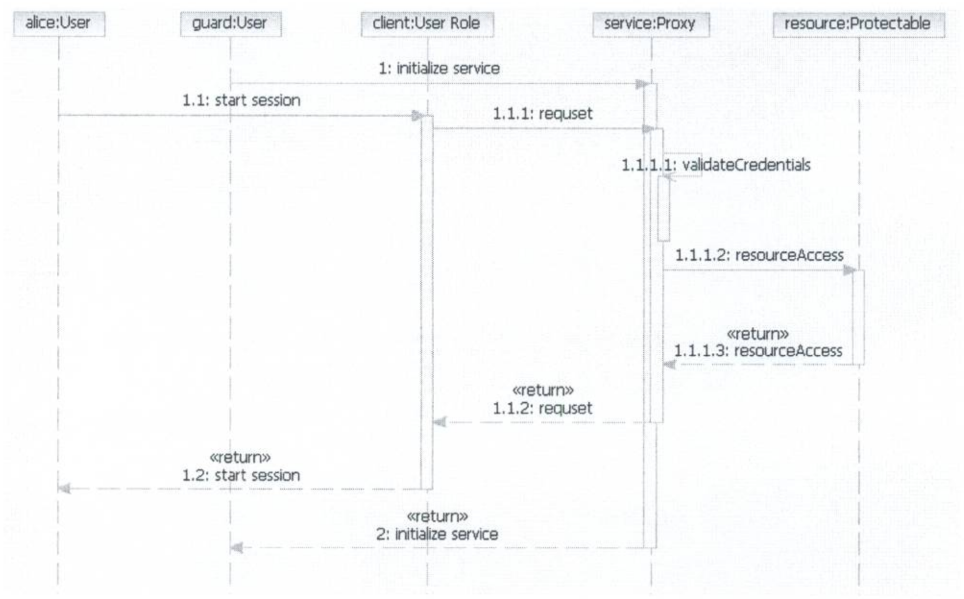

Fig. 6: Proxy – interaction diagram

Web service controlled by object guard enables access to protected resource. This object has role Proxy that has privileges to access protected resource directly. All other object have to use such broker that has Proxy role to be able to access the protected resource.

As soon as service object (that has Proxy's permissions) receives request from the client, it has to validate user's credentials. User's credentials are sent via the request method depicted in figure 6. If service decides that client is allowed to have access to resource, it creates Trusted Subsystem Credentials and sends request to resource (represented by Protectable interface). These credentials are sent via requestAccess message. Response from the resource is sent to service that replies the client.

5 Conclusions

In this work we have introduced design patterns for RBAC that should make role management more easier. Our work was motivated by attempt to reduce cost of a role definition task. We have shown patterns describing common problems encountered during association of permissions to roles. Design of the patterns should lead security administrators to solutions of their particular problems. This will reduce time of development of a role scheme.

This work is aimed mainly to parameterized roles because they reflect reality more precisely than pure RBAC model. We have demonstrated usage of these patterns in several real life examples. As shown above, many ordinary situations should be solved in a uniform way. Advantages of particular patterns were also mentioned. In a future we want to continue in searching for other patterns that will bring general solution of other common problems.

Bibliography

1. Basin, D., Doser, J., Lodderstedt, T. *Model driven security for process-oriented systems* Proceedings of the eighth ACM symposium on Access control models and technologies, ACM Press, New York, USA, 2003. ISBN:1-58113-681-1. Pages: 100–109.
2. Bacon, J., Moody, K., Yao, W. *A Model of OASIS Role-Based Access Control and Its Support for Active Security.* ACM Transactions on Information and System Security (TISSEC), ACM Press, New York, USA, 2002. ISSN:1094-9224. Pages: 492–540.
3. Ferraiolo, D. F., Kuhn, D. R., Chandramouli, R. *Role-Based Access Control.* Artech House Publishers, 2003, Boston. ISBN 1-58053-370-1.
4. Ferraiolo, D. F., Sandhu R., Gavrila S., Kuhn, D. R., Chandramouli R. *A Proposed Standard for Role Based Access Control.* [online] Aug 2001 [cited 23 Jan 2006]. `http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf`
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns.* Addison-Wesley Publishing Co., 1995; ISBN: 0201633612
6. Luigi Giuri, Pietro Iglio: *Role templates for content-based access control.* In Proceedings of the second ACM workshop on Role-based access control. ACM Press, New York, USA, 1997. Pages: 153–159
7. Microsoft Corporation. *Web Service Security - Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0* [online]. Dec 2005 [cited 23 Jan 2006]. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/wssp.asp`
8. The Open Group. *Single Sign-On* [online]. 2005 [cited 1 Feb 2006]. `http://www.opengroup.org/security/sso/`