Designing Application Authorizations

Leszek A. Maciaszek
Macquarie University, Australia
leszek@ics.mq.edu.au

Mieczyslaw L. Owoc
University of Economics, Poland
mowoc@manager.ae.wroc.pl

Abstract

Information systems must be protected from unauthorized access. ***Authorization*** *has been studied extensively as the main form of preserving the* security of databases. *Every database management system provides a sophisticated set of options aimed at protecting the database from unauthorized access. An important practical problem is how to take advantage of the database security options to ensure that a user is permitted to access the database through the* application program *but may not be allowed to access the database directly via database query tools. A related issue is how to extend the user privileges on the* client *part of the application so that only* authorized GUI controls *are available to the user.*

In this paper we propose a model for the design of necessary authorization settings *into both the* client *and the* server *parts of a database application. The settings are stored in an* ***Authorization Database (ADB)*** *to which the program connects to customize itself for the current user. The customization is based on an* application role *granted to the user. An application role is activated for a connection (user session). After the database server authenticates the user, the user login to the application role can be transparently obtained by the application from the ADB.*

Keywords: authorization design, database applications, security, client/server systems

Introduction

The security mechanisms must be built into the *client* (user interface) and the *server* (database) objects. The protection starts at the client. The ***client program*** must be able to disallow unauthorized access to GUI (graphical user interface) objects, such as menu items, action buttons, data fields, windows. The program should configure its client objects depending on the authorization level of the current user (***authenticated*** – as the minimum - by user id and password). The program turns off access to unauthorized GUI objects as needed.

In a well-designed authorization, the client should address as many security loopholes as possible. This avoids unnecessary and expensive trips to the database (perhaps to be informed that access to data or execution of operation is refused by the server). ***Server permissions*** (*privileges*) fall into two categories. A user may be given selective permissions to:

- access individual *server objects* (tables, views, columns, stored procedures, etc.),
- execute SQL statements (select, update, insert, delete, etc.).

In the database world, permissions for a user may be assigned directly at a *user level* or at a *group level*. The Security Administrator can assign permissions to a group of users in a single entry. A user may belong to none or to many groups. To allow greater flexibility with managing authorization, most database management systems introduce an extra level of authorization – the *role level*. The role allows the Security Administrator to grant permissions to all users who play a particular role in the organization. Roles can be nested – i.e. the permissions granted to different role names can overlap.

In larger IS applications, the authorization design is an elaborate activity. To properly handle the task, an ***Authorization Database (ADB)*** should be set up alongside the application database to store and manipulate the client and server permissions. The application program can then consult the database after the user's login in order to identify the user's authorization level and configure itself to that user.

Any changes to database permissions are then recorded and managed in the ADB – i.e. nobody, even the Security Administrator, is allowed to directly change the application database permissions without first updating the authorization database. The objective of this paper is to present a generic model for the authorization database and explain how such a database can be integrated in the overall design of client/server applications to control effective user permissions.

Designing Security Chains

The security management begins with granting a login ID and password to a user. The application will only allow a connection after the login ID and password are ***authenticated***. An authenticated user is granted access to the application at the level determined by the access permissions granted to that user. In modern information systems, the users obtain the permissions indirectly through their associations to user *groups* and user *roles* (Oracle8*i* (2000); SQL Server 7.0 (2000); Sybase 12.0 (2000))

A user can be allocated to a ***group***. The authorization settings defined for a group apply to all users of that group. Groups are defined hierarchically. A user of a lower-level group inherits the authorization settings of higher-level groups, in addition to the authorization settings defined for the group itself (and the authorization settings defined explicitly for the user account, if any). A user can belong to many groups.

Although users and groups can be directly granted/denied privileges (authorization settings), we assume in our model that application privileges are obtained exclusively via ***application roles***. Users and groups belong to roles. Client and server object permissions are assigned to application roles. Like groups, lower-level roles inherit permissions of higher-level roles. A user or a group can belong to more than one role.

Figure 1 represents the design of a part of the ADB schema concerned with the definition of the security chains that allow the user to obtain necessary authorization settings to work with the application (Maciaszek, 2001). The diagram shows also referential relationships to two tables (RoleToClientPermission and RoleToServerPermission) that link application roles to permissions assigned to them. In the following sections we show how the user obtains the effective permissions and how the specific client and server permissions are defined in the ADB.

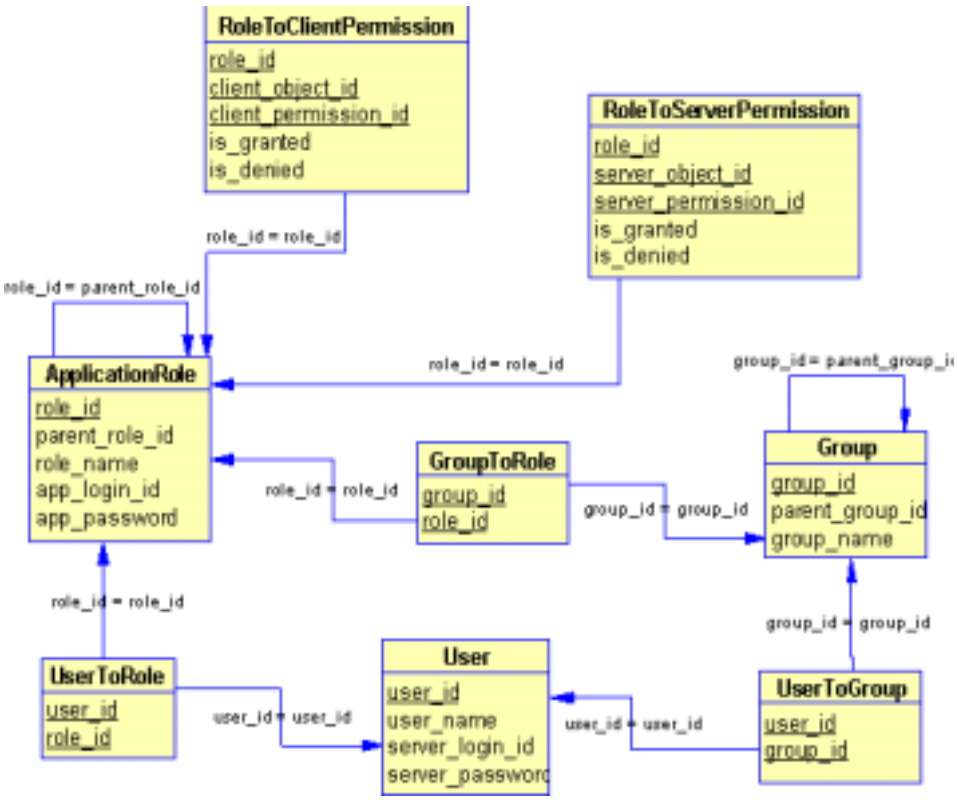

Figure 1. Design of security chains

Effective Permissions of Application Users

The ***effective permissions*** granted to a user are defined by the precedence rules along the security chains from the server login to the application role login. In general, the effective permissions of a user on an application object are the union of all granted, denied, or revoked permissions on an object assigned to the user and the groups and roles that that user belongs to. The precedence rules state that a denied permission at any level (user, group, or role) takes precedence over the same permission granted at another level.

In our model, most (if not all) permissions are maintained in the application roles. If Anne is a member of roles A and B, and the role A grants access to a client or server object but the role B denies the same access, Anne will be denied the access. This is because the denied access takes precedence over the granted access.

Our model assumes that a database server supports the DENY permission statement statements (as in SQL Server 7.0) in addition to commonly available GRANT and REVOKE. A ***denied permission*** applies across all authorization levels. A ***granted permission*** supersedes (removes) the revoked or denied permission but only at the level granted. Similarly, a ***revoked permission*** replaces the granted or denied permission at the level granted.

Figure 2 shows a UML Activity Diagram for effective permissions of application users. The user Mary attempts

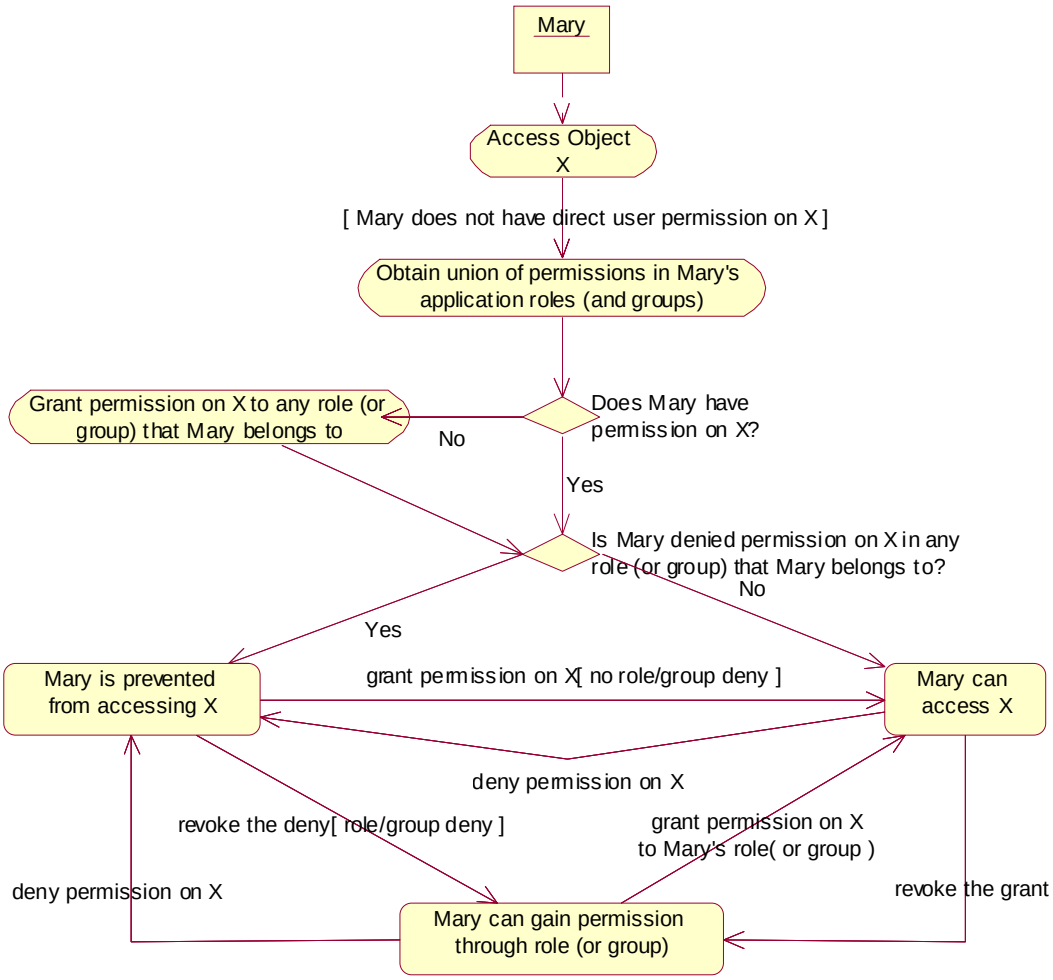

Figure 2. Activity diagram for permission

to access the object X. X can be a client object (such as menu item) or a server object (such as a select from a table). We assume that Mary does not have direct permission on X granted to her user account. To determine if Mary can access X, the system obtains the union of permissions that Mary inherits from her application roles and groups. If one of these permissions is granted on X *and* the permission on X is not explicitly denied in any other Mary's roles or groups, then Mary can access X. Otherwise, Mary is prevented from accessing X. The lower part of Figure 2 identifies three possible permission states and the transitions allowed between these states. If Mary is currently prevented from accessing X then this is because of one of two possibilities. The first possibility is that no Mary's role or group has been granted the permission and no Mary's role or group has been denied the permission. The transition "`grant permission on X`" changes Mary's state so that she can access X. The second possibility is that at least one Mary's role or group has been denied permission on X. The transition "`revoke the deny`" puts Mary into the state in which she can gain permission through role or group. Once such permission is granted Mary is again in the state in which she can access X.

Although our model allows taking the union of all user/group/role permissions, our recommendation is to control permissions in application roles only. The main reason for providing authorization settings via application roles (and not via groups or user accounts) is to ensure that a user *can only access the database through the application* without the possibility of gaining a back-door access to the database via SQL query tools. In our model, the database server login IDs and passwords ***and*** the login IDs and passwords for application roles are stored in the ADB.

It follows that our model still requires that the user connect to the database server using the predefined *server login* (however the server login alone may only grant the database connect privilege but no other database access privileges). Without the server login the auditing of specific user activities (via application) on the database would not be possible. Once connected, the *login to the application role* is transparent to the user and the effective permissions are obtained by the application from the ADB.

An application role is activated for a ***connection*** (user session). The authorization settings that the user obtained from the application role remain in effect until the user logs out of the application program (and, therefore, logs out of the database server).

Designing Client Authorizations

Our model assumes that the ADB is used by the application program to establish authorization settings with regard to both the client and the server objects. This is consistent with the requirement that an unauthorized user should be refused to perform unauthorized activities *on the client*. If that activity can compromise the integrity of the database then the second level of protection *on the server* is also enforced.

The ***client permissions*** relate to windows and window controls. A user may be prevented from:

- CRUD (Create, Read, Update, Delete) operations on primary windows, secondary windows, window panes and other recognizable parts of a window
- CRUD operations on window fields, including text, combo and spin boxes
- Activating window controls, including:
    1) Drop-down menu items
    2) Pop-up menu items
    3) Toolbar buttons
    4) Command buttons
    5) Picklist selections
    6) Keyboard keys
    7) Function keys
    8) Accelerator keys
    9) Screen cursor movements

The ADB stores the catalogue of all application ***client objects*** that are subject to authorization settings. The catalogue includes application windows, window fields and window controls. Any hierarchical relationships between client objects are also maintained (such as panes in a window, fields in a window, or menu hierarchy).

The ***permissions*** that can be granted for client objects are:

- `can_create`

  Permits a user to open a window for insert operations and to insert data values in the window fields.

- `can_read`

  Permits a user to only view information in a window, window field or window control.

- `can_update`

  Permits a user to open a window for update operations and to update data values in the window fields.

- `can_delete`

Permits a user to open a window for delete operations and to delete all or selected data values in the window fields.

- `can_activate`

  Permits a user to activate a window control

Each client object can be granted or denied more than one permission. The ***revoked permissions*** are not stored in the ADB - they are not needed to determine the authorization settings for the current user (a revoked permission is simply a permission that has not been granted).

The allowed mappings of permissions to client objects are stored in the ADB table called `ClientObjectToPermission` (Figure 3). An *application role* is then assigned its permissions on application's client objects. The assignment can be either that the permission *is granted* or that it *is denied*. This information is stored in the ADB table `called RoleToClientPermission` (the attributes `is_granted` and `is_denied`).

Because of possible hierarchical relationships between client objects, complex *integrity rules* are implemented in the ADB to prevent inconsistencies between granted/denied permissions. For example, it does not make sense to grant permission on a submenu if the permission is denied on a menu that contains it. Similarly, a denied permission for can_read precludes a granted permission for can_update. The integrity rules governing the *effective permissions* to a user belonging to multiple roles are resolved according to the strategies discussed in previous sections.

Designing Server Authorizations

Ultimately the security and integrity of the database is the responsibility of the database itself, not the client applications accessing the database. The client authorizations can only eliminate some security breaches early in the process and can free the database from duplicated and unnecessary checks. Not all potential security breaches can be addressed in the client. For example most integrity constraints implemented in database triggers can only be enforced once a client-authorized insert, delete or update operation hits a database table.

The ***server permissions*** relate to database objects - tables, views, columns, stored procedures, etc. A user may be prevented from:

- CRUD (Create, Read, Update, Delete) operations on persistent database objects

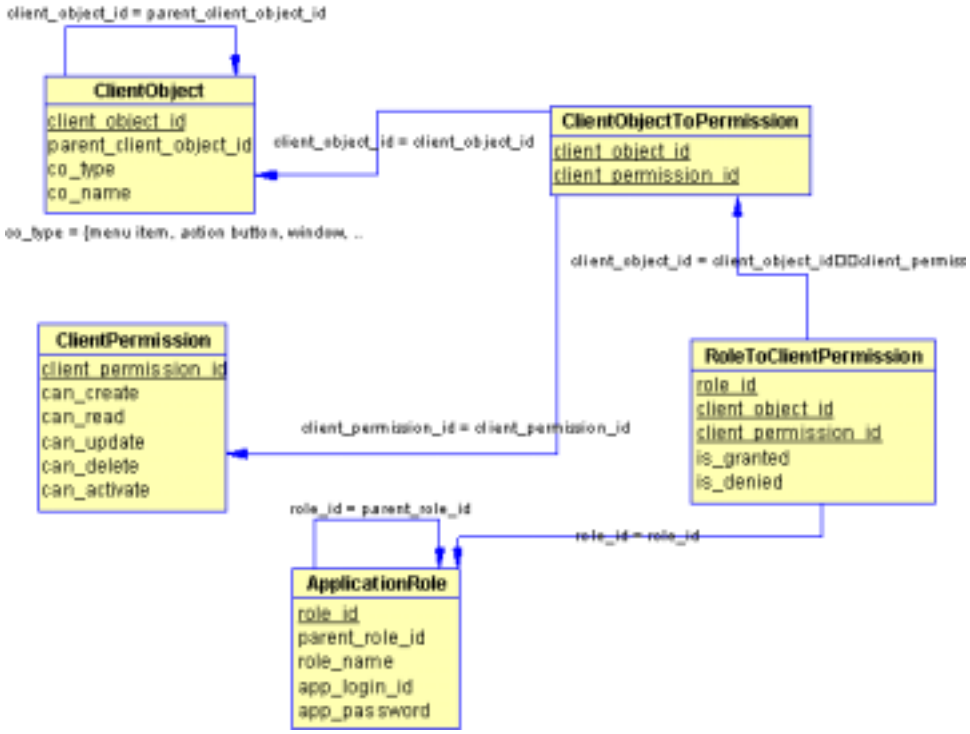

Figure 3. Client Authorizations

- reading from views
- inserting foreign key references
- persistent objects
- executing SQL statements
- executing stored procedures

The ADB stores the catalogue of all application ***server objects*** that are subject to authorization settings. The catalogue includes tables, views, stored procedures and any individual table columns that must be protected from indiscriminate access. Hierarchical relationships between server objects are maintained (such as columns in tables).

The ***permissions*** that can be granted for server objects are:

- `can_select`

  Permits a user to retrieve data from one or more columns of a table or view.

- `can_update`

  Permits a user to update data in one or more columns of a table or view (subject to view updateability principles adopted by a database system).

- `can_insert`

  Permits a user to insert rows containing data for one or more columns into a table.

- `can_delete`

  Permits a user to delete rows of data from a table.

- `can_reference`

  Permits a user to insert a row into a table that has a foreign key referencing another table to which the user may not have a granted permission.

- `can_execute`

  Permits a user to execute a stored procedure.

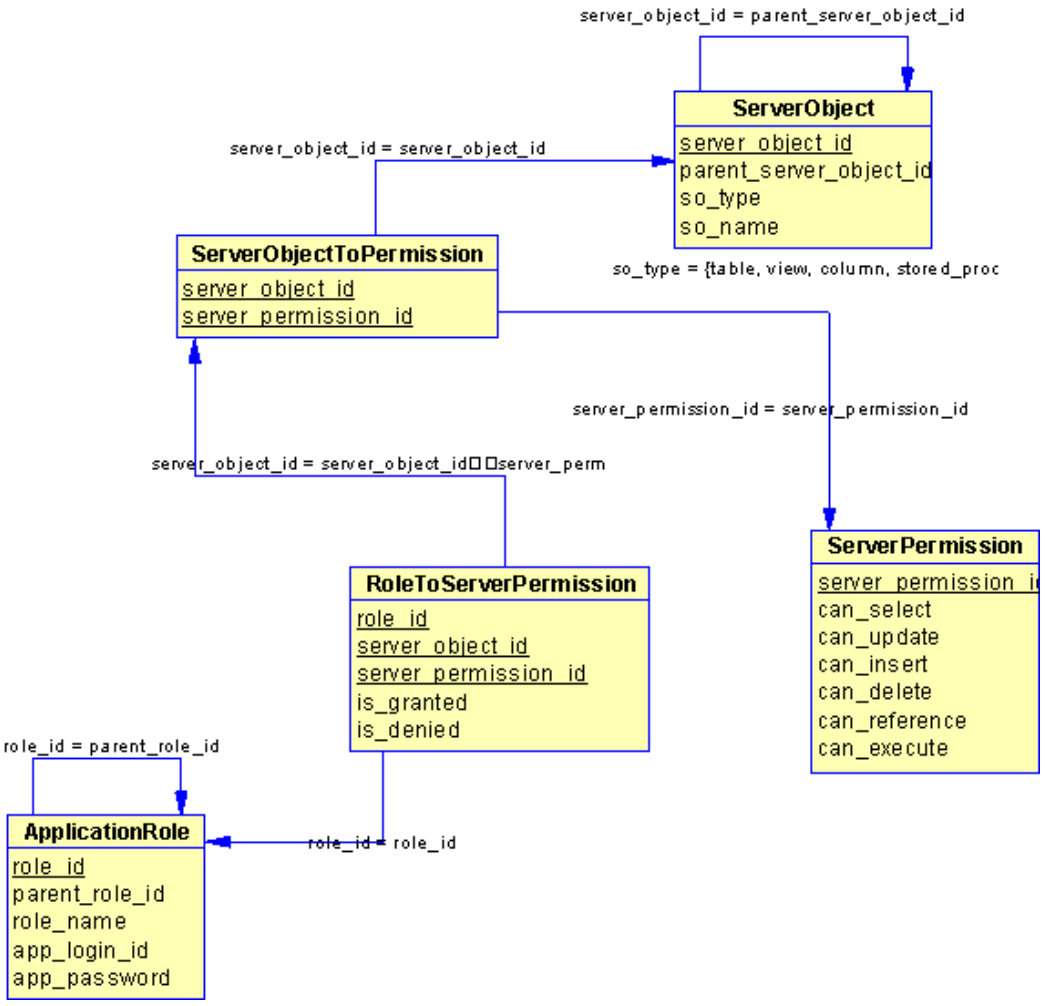

Figure 4. Security Authorizations

Like with client objects, each server object can be granted or denied more than one permission. The ***revoked permissions*** are not stored in the ADB.

The allowed mappings of permissions to server objects are stored in the ADB table called `ServerObjectToPermission` (Figure 4). An ***application role*** is either granted or denied permissions to server objects to which the role is linked via referential relationship. This information is stored in the ADB table called `RoleToServerPermission` (the attributes `is_granted` and `is_denied`).

Conclusions

Database applications are normally accessible to large number of users with varying security clearings. It is therefore important that application authorizations are carefully designed. In this paper, we have proposed that an Authorization Database (ADB) - separate from the application database - is set up. The ADB is then used to verify the current users' permissions and to customize the application GUI to correspond to these permissions (such as dimming unavailable menu items).

The application is customized on user's login. The login is performed in two stages - the login to the server is followed by the login to the application role (the latter can be transparent to the user). The user gains permissions based on his/her application roles. Other permissions that the same user may have on the database may be invalidated for the duration of the user's connection. The server login allows auditing of user activities on the database.

We believe that the model presented in this paper fills a gap in the literature dealing with application security issues. The paper presents a pragmatic solution for authorization design that can be used as a blueprint for controlling authorizations in large client/server systems. A variant of the proposed design has been used by a large international market research company and implemented in an advertisement monitoring system.

References

Maciaszek, L.A. (2001): Requirements Analysis and System Design. Developing Information Systems with UML, Addison-Wesley.

Oracle8i (1999): Oracle 8i CDROM Documentation.

SQL Server 7.0 (2000): SQL Server Books Online.

Sybase 12.0 (2000): Sybase Adaptive Server Enterprise 12.0 CDROM Documentation.

Ackowledgements

The authors would like to thank Bozena Cioch-Maciaszek for implementing and validating the ADB models discussed in this paper.

Biographies

Leszek Maciaszek is Associate Professor of Computing at Macquarie University, Sydney, Australia. His research has been in databases, object-oriented technology, software engineering and large-scale business information systems. Professor Maciaszek has authored close to 100 publications including the books *Database Design and Implementation* (Prentice Hall, 1990) and *Requirements Analysis and System Design. Developing Information Systems with UML* (Addison-Wesley, 2001).

Mieczyslaw Owoc is Lecturer at University of Economics, Wroclaw, Poland. His research has been in knowledge validation and verification, knowledge management, expert systems, artificial intelligence, database technology, distance and open learning. Dr Owoc has published numerous papers, articles and book chapters.