Full View With Errors

Alias:

Full View With Exceptions
Reveal All and Handle Exceptions
Notified View

Motivation:

Once an officer is allowed on a military base, he or she could go to any building on the base. In effect, the officer has a full view of the buildings on the base. If the officer tries to enter a restricted area without proper clearance, either someone would stop and check them noting that they are not allowed in the restricted area, or alarms would sound and guards would show up to arrest the officer.

Graphical applications often provide many ways to view data. Users can dynamically choose which view on which data they want. When an application has these multiple views, the developer must always be concerned with which operations are legal given the current state of the application and the privileges of the user. The conditional code for determining whether an operation is legal can be very complicated and difficult to test. By giving the user a complete view to what all users have access to can make teaching how to use the system easier and can make for more generic GUIs.

Problem:

Users should not be allowed to perform illegal operations.

Forces:

- Users may be confused when some options are either not present or disabled.
- If options pop in and out depending upon *Roles*, the user may get confused on what is available.
- Users should not be able to see operations they are not allowed to do.
- Users should not view data they do not have permissions for.
- Users do not like being told what they cannot do.
- Users get annoyed with security errors, permission denied, and illegal operation messages.

Solution:

Design the application so users see everything that they might have access to. When a user tries to perform an operation, check if it is valid. Notify them with an error message when they perform illegal operations.

This pattern is very useful when a user can perform almost any operation. It is easier to show the user everything and just provide an error message when an illegal operation is attempted.

The solution for this pattern is simple when only a few error message need to be displayed. Just display the error message to standard error or in a dialog box. If many error messages are spread throughout the application, a separate error reporting mechanism may be useful. This mechanism could also be used for error logging.

Typically, an error-reporting framework would have two principal components. The log event object has a message describing the error condition and a severity level indicating if the event is a warning, an error, or just user information. When a log event is created it can automatically register itself with the logger. The logger is a *Singleton* that automatically receives and processes log events. The logger can be configured to display dialogs or write to a file depending on the severity of the event.

Example:

One example is an Oracle database. When you are using SQLPlus to access the data, you can execute any command. However, if you try to access data you don't have permission to see, an appropriate error message will be displayed.

Consequences:

- ✔ Training materials for the application are consistent for each type of user.
- ✔ Retrofitting this pattern into an existing system is straightforward. Just write a GUI that will handle all options and whenever a problem happens with an operation, simply exit the operation and open an error dialog.
- ✔ It is easier to dynamically change privileges on an operation because authorization is performed when the operation is attempted.
- ✔ It is easier to implement since you don't have to have multiple views on the data.
- ✘ Users may get confused with a constant barrage of error dialogs.
- ✘ Operation validation can be more difficult when users can perform any operation.
- ✘ Users will get frustrated when they see options that they cannot perform.

Related Patterns:

- *Limted View* is a competitor to this pattern. If limiting the view completely is not possible, this pattern can fill in the holes. The "Putting It All Together" section at the end of this paper compares these two patterns and describes when to use each.
- *Checks* [Cunningham 95] describes many details on implementing GUI's and where to put the error notifications.
- *Roles* will be used for the error notification or validating what the user can and can not do.

Known Uses:

- Login windows inform users when they enter incorrect passwords.
- SQLPlus, used to access an Oracle database, allows you to execute any and displays an appropriate error message if illegal access is attempted.
- Most word processors and text editors, including Microsoft Word and vi, let the user try to save over a read-only file. The program displays an error message after the save has been attempted and has failed.
- Reuters SSL Developers Kit has a framework for reporting error, warning, and information events. It can be configured to report errors to standard error, a file, or a system dependent location such as a dialog.

Non-security Known Uses:

- Assertions [Meyer 92] are a form of this pattern used within a programming interface. Since a class' interface cannot change, preconditions ensure that each method is used within a legal context.
- The Java programming model of throwing exceptions follows this pattern. Instead of designing methods to only be used in the proper context, exceptions are thrown forcing the rest of the application to deal with the error.