Patterns and Refactoring

FreeCol

MileStone 4 - SOEN6471

Ede Cameron 2952270

Patterns

There is one Pattern within FreeCol that is apparent just from the names given to the FreeCol Packages. In the Server Packages there is the Model Package and the Control Package. In the Common Packages there is also a Model Package, and in the Client there is a Control Package. It can be assumed that these Packages are intentionally named to suggest the use of the Design Pattern, Model-Control-View (MVC) which is a widely used Design Pattern for Client - Server based applications.

The View is the visual aspect of an application accessible to the user and generally refers to the Graphical User Interface (GUI). Within the MVC Pattern is also the Controller Pattern which delegates diverse commands from the View to the different implementation rules of an application aspects contained within the Model.

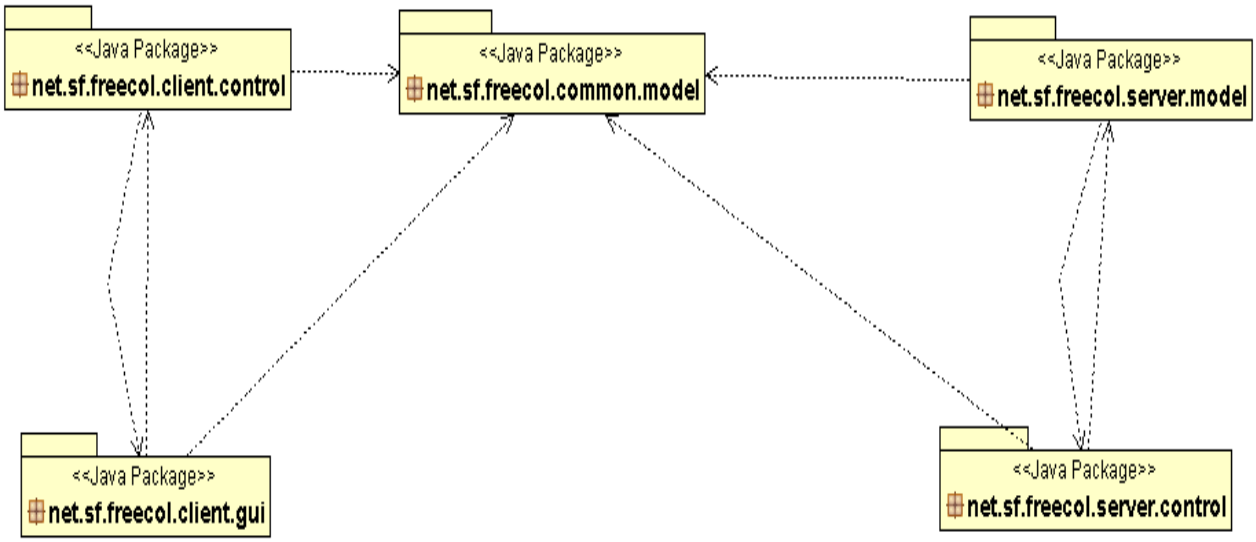

Figure 4.1 FreeCol Model Control and GUI (View) Packages - Showing dependencies between the Packages.

The Diagram above shows how the developers have implemented the MCV Pattern. There are some discrepancies between the Pattern description and the implementation,

there is a relationship between the GUI and the Common Model that shouldn't be there, but this dependency is to set up international language features for the game, but this seems to be the only noticeable dependency, the rest of the implementation seems to follow the Model Control View Pattern.

The main advantage to using an MCV Pattern is that the View can be decoupled from the Model and the Controller. In other words the Model should have no knowledge of the View. This means that any change to the GUI (View) should never affect the model and any change to the Model should have no affect on the user's view of an application though will affect behavior which is the responsibility of the model.

In general terms this means that a button represented in the GUI could initially be set to add one to an already set value and the new value is displayed on the screen.The model is written at some point to add two instead of one, because the view and model are decoupled the only code that needs to be changed will be in the model the view will stay the same but display a different number.

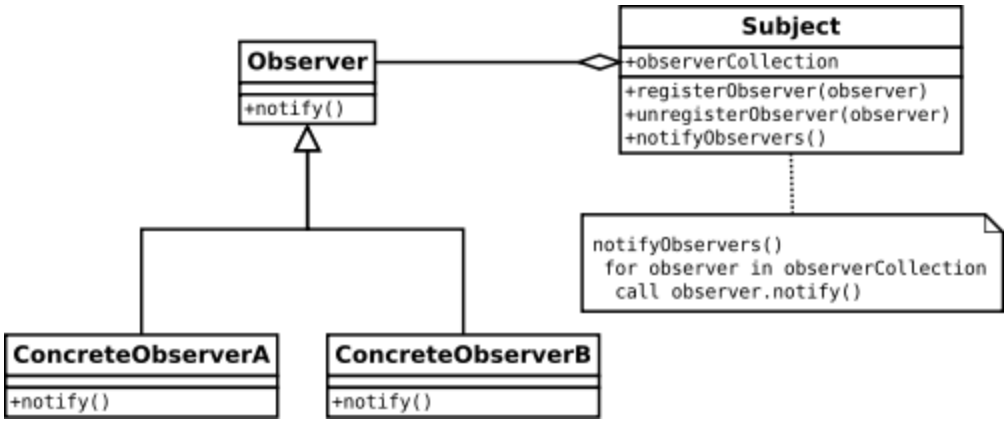

Figure 4.2 The Observer Pattern [2]

"The Observer pattern defines and maintains a dependency between objects. The classic example of Observer is in Smalltalk Model/View/Controller, where all views of the model are notified whenever the model's state changes." Design Patterns

There is however another Pattern which is integral to the MCV Pattern and is also used in FreeCol, and that is the Observer Pattern. in fact the Java GUI Library is based partially on the Observer Pattern because the Java GUI is Action Event based, in other words the GUI provides listeners (Observers) that react to events that are triggered through user-GUI interaction. This is an integral aspect of how the events triggered in the View are translated

by the Controller to invoke methods and pass data to the Model which in turn will update the view. These listeners are called Action- Event Listeners and are generally window event listeners triggered by a mouse or keyboard in the case of the FreeCol user interface.

```
public void fireActionEvent(ActionEvent event){
    ActionListener[] listeners = listenerList.getListeners(ActionListener.class);
    for(int i=0; i < listenerList.getListenerCount(); i++){
        listeners[i].actionPerformed(event);
    }
```

The Code above is one one of FreeCo'sl implementation of the Java Event Listeners.
In this case from the TerrainCursor Class in the Client GUI Package. The cursor represents the mouse of the Player, is then added to the MapViewer Class. The cursor when it receives input from the game (there are several other Classes involved in triggering an event) will fire events that are in the ActionListener List this very much the Observer Pattern, the listeners been the subscribers and the Terrain GUI been the publisher. These events are separated from the publisher because of the Model Control View architecture the publisher in this case controls when the events are published, but has no idea of what these events may be.

Refactoring FreeCol - Large Class

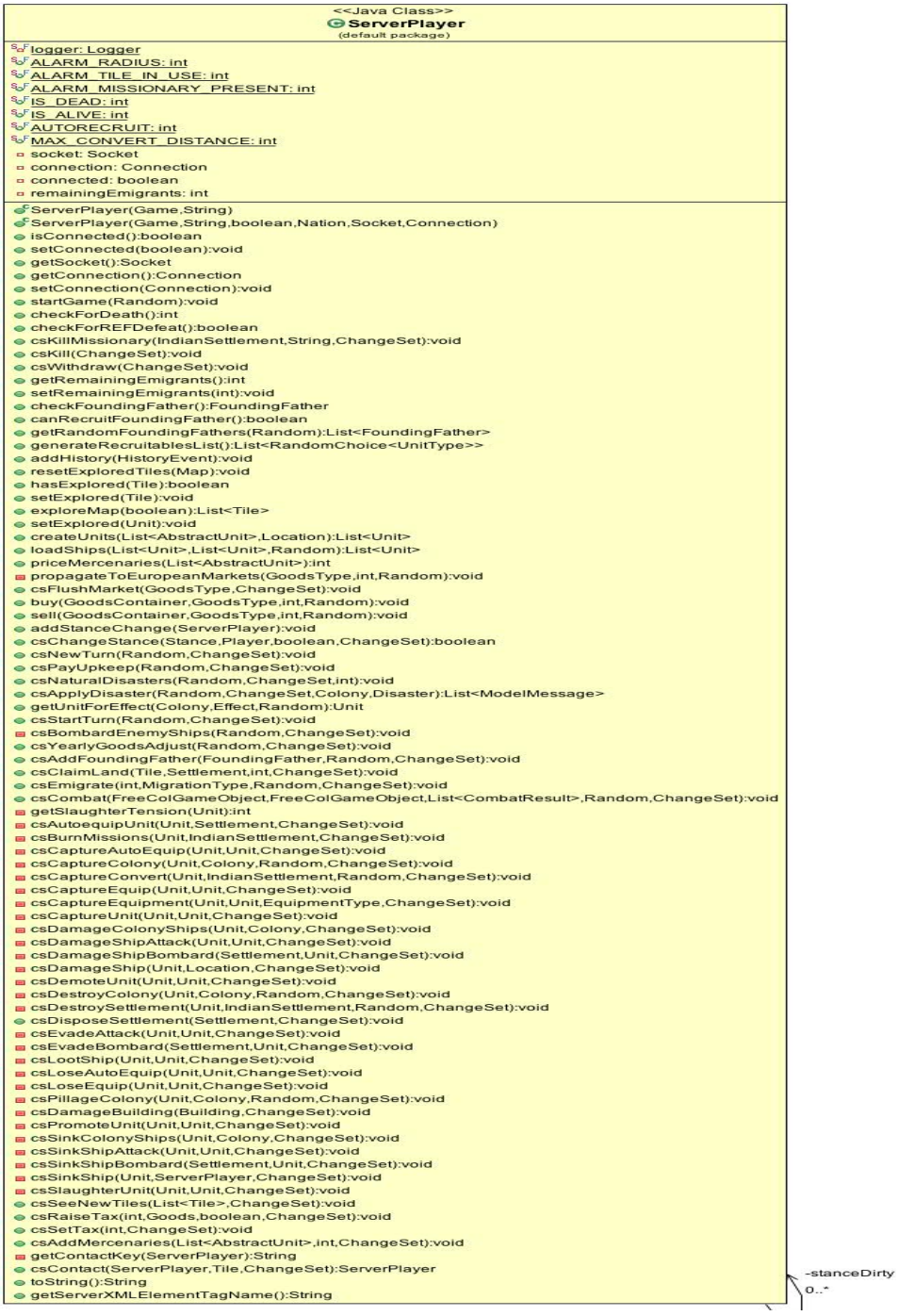

Figure 4.3 The ServerPlayer Class in the FreeCol Server Model package.

The Server Packages of FreeCol from a design perspective are,from the Package view, well organized, but within these packages there are numerous problems. First and foremost are the overly large Classes, from Milestone Three we saw several Classes that

were well over 1500 lines of code. One Class specifically the ServerPlayer Class is over 3000 lines and contains over 70 methods. These methods are all connected to the Player Object, but are very diverse in nature, and some of these methods are over 400 lines long. Although Extract Class and Extract method could rectify this problem, there rests the question of where to start.

It should be noted that this refactoring is quite long there were numerous reasons to do all these refactorings but the primary reason was to show that although daunting, cleaning up this class was possible. There are three basic refactorings here each one done once or twice to show its relevancy. Extract Class from ServerPlayer, move related methods into new class, ServerPlayerCombat and within this class extract methods from large method csCombat and finally extract classes from ServerPlayerCombat in polymorphic subclasses. It should also be noted that when starting these refactorings the possible refactorings in Milestone 3 were set aside in order to pursue the refactoring methods found during this milestone

Patchset 1/15 Refactor Class ServerPlayer

Create new Class ServerPlayerCombat

There are themes within the ServerPlayer Class such as combat, or trading. The ServerPlayer Class has several long methods. one method specifically is csCombat. This method has more than 500 lines of code and seems a good place to start the refactoring in that it represents the combat theme. In order to move this method into its own Class the Class needs to be created. The first step was to add the Class ServerPlayerCombat.java to the Package server.model, the same location as ServerPlayer.java

Patchset 2/15 Extract Method

Move Method csCombat into the ServerPlayerCombat Class

We were told to think small but moving the method csCombat out of the ServerPlayer Class broke numerous dependencies. This commit moves the method into the new Class. Eclipse fortunately auto generated most of the import statements but the errors in the New Class method were numerous. The commit reflects the move and the broken dependencies

Patchset 3/15 Handling Dependencies

Start recognizing and handling dependencies between ServerPlayer and ServerPlayerCombat

Although there are three steps in this refactoring they follow the logic of calling the new class from ServerPlayer and what the new class requires from the old class, in the hierarchy chain. The next refactorings recognize some interesting dependencies and try to rectify these problems

The first step in the refactoring was to add a constructor to the ServerPlayer Class that would call the ServerPlayerCombat class. There was never the intention of trying to separate the two Classes completely but have the csCombat method in ServerPlayer call the delegated method in ServerPlayerCombat. The inheritance hierarchy was then added so ServerPlayerCombat would extend the ServerPlayer and implement the ServerModelObject, like the parent class. Then the two methods getServerXMLElementTagName and csNewTurn were overridden after implementing the ServerModelObject in the new class.

PatchSet 4/15 Changing method visibility

The next refactoring was changing the methods called from ServerPlayerCombat that were still in the ServerPlayer Class from private to protected. After this was done a build was attempted and succeeded, but on close inspection of this refactoring, it seemed that this fix didn't resolve the inherent Large Class problem. It also was apparent atfer looking at the code in ServerPlayerCombat that the classes called from ServerPlayerCombat were only called from ServerPlayerCombat.

Patchset 5/15 Small fixes

Several small fixes were made to the ServerPlayerCombat Class including making a global instance of the ServerPlayerCombat Class in the ServerPlayer instead of a local instance in the method body, removing and adding some comments. It was at this stage that a build of FreeCol was made and succeeded. It was at this point that removing other methods from ServerPlayer commenced.

Patchset 6/15 Reducing the size of the ServerPlayer Class

Extract Method GetSlaughterTension from ServerPlayer to ServerPlayerCombat.

If the idea of this refactoring process was to reduce the size of the ServerPlayer Class then changing methods from private to protected doesn't really solve the problem and in fact creates a new code smell Feature Envy. The now protected classes which for the most part were now been called from ServerPlayerCombat.

This Patch represents the initial test to extract these methods. The method GetSlaughterTension was extracted from ServerPlayer to ServerPlayerCombat. The build was tested and suceeded.

The methods now in ServerPlayer whose visibilty was changed from private to protected really belong either in the ServerPlayerCombat Class or in Classes extended from ServerPlayerCombat. In fact the refactoring process will be to extract the classes into ServerPlayerCombat and then using polymophism move them into related Classes extended from ServerPlayerCombat

This one method move of GetSlaughterTension was to see if there were dependencies involved when moving the method into the ServerPlayerCombat Class. There were no errors reported, so decided to extract all relevant methods from ServerPlayer into ServerPlayerCombat.

Patchset 7/15 Massive Moves

Since the the refactoring in 6/17 worked with few problems all other protected methods changed during the refactoring of 4/17 were moved into the ServerPlayerCombat Class, and although this commit has over 1400 lines in the patch file (Patch7-15.patch), the moves were done one by one and tested but the commit condenses what would have been numerous repetitious commits into one large but tested commit. The ServerPlayer Class is reduced by over 500 lines of code but this now means that ServerPlayerCombat has increased by 500 lines.

Patchset 8/15 JUnit testing

Change constructor in ServerPlayerCombat to resolve JUnit Test Failure.

Although FreeCol was successfully building with what seemed no errors, the JUnit tests that are included in the FreeCol Source were failing and up until this point there seemed to be little point in using these tests until after the massive move of methods the JUnit tests seemed warranted. Of course on first run there were several errors and the JUnit Tests became a required step in building and testing after this and I admit should have been done from the start.

Error thrown from logger duplicate ID. Solved by changing the constructor for ServerPlayerCombat to only pass the game and not the Id (each class needs a unique id, but couldn't be passed directly). Retested and recompiled

Patchset 9/15 Delete Old Method csCombat From ServerPlayer

After Unit Testing the new class ServerPlayerCombat the old method csCombat and other methods alresdy moved into ServerPlayerCombat (although already commented out) were deleted from ServerPlayer - again passed JUnit test and recompiled.

Patchset 10/15 - Extract Remaining Methods

All other protected methods moved to ServerPlayerCombat
Final move of all remaining protected methods called in ServerPlayerCombat from ServerPlayer into ServerPlayerCombat. Tested, build suceeded

Patchset 11/15 Extract Method from csCombat in ServerPlayerCombat

Create new smaller method handleStanceAndTension to handle same method body code.

These next two refactorings move some of the code from the method csCombat into their own methods, in an attempt to reduce the size of csCombat. The object is just to show the refactorings needed within the long method.the visibilty changed of several booleans and integer values for tension from local to method to private global methods in ServerPlayerCombat Class. Again Unit tested

Patchset 12/15 Extract Method

moveAttacker Method created - same process as handleStanceAndTension

Patchset 13/15 Create Class

Start replacing enumerated types with polymorphism
Created Class SlaughterUnit

These final refactorings are to now start reducing the size of the class ServerPlayerCombat. There are numerous calls to methods within the ServerPlayerCombat Class form enumerated types these enums are been sent from the client side classes of FreeCol and would be complex to refactor but each method call within the enumerations can be moved into separate classes to reduce the size of ServerPlayerCombat.
Replaces method csSlaughterUnit moved into ServerPlayerCombat from serverPlayer and creates the class SlaughterUnit to handle the same

Patchset 14/15 Create Class

Created Class BurnMissions
Same process as Create Class SlaughterUnit with the same justifications

Patchset 15/15 Delete Method

Delete method burnMissions from ServerPlayerCombat - run JUnit tests and compile.

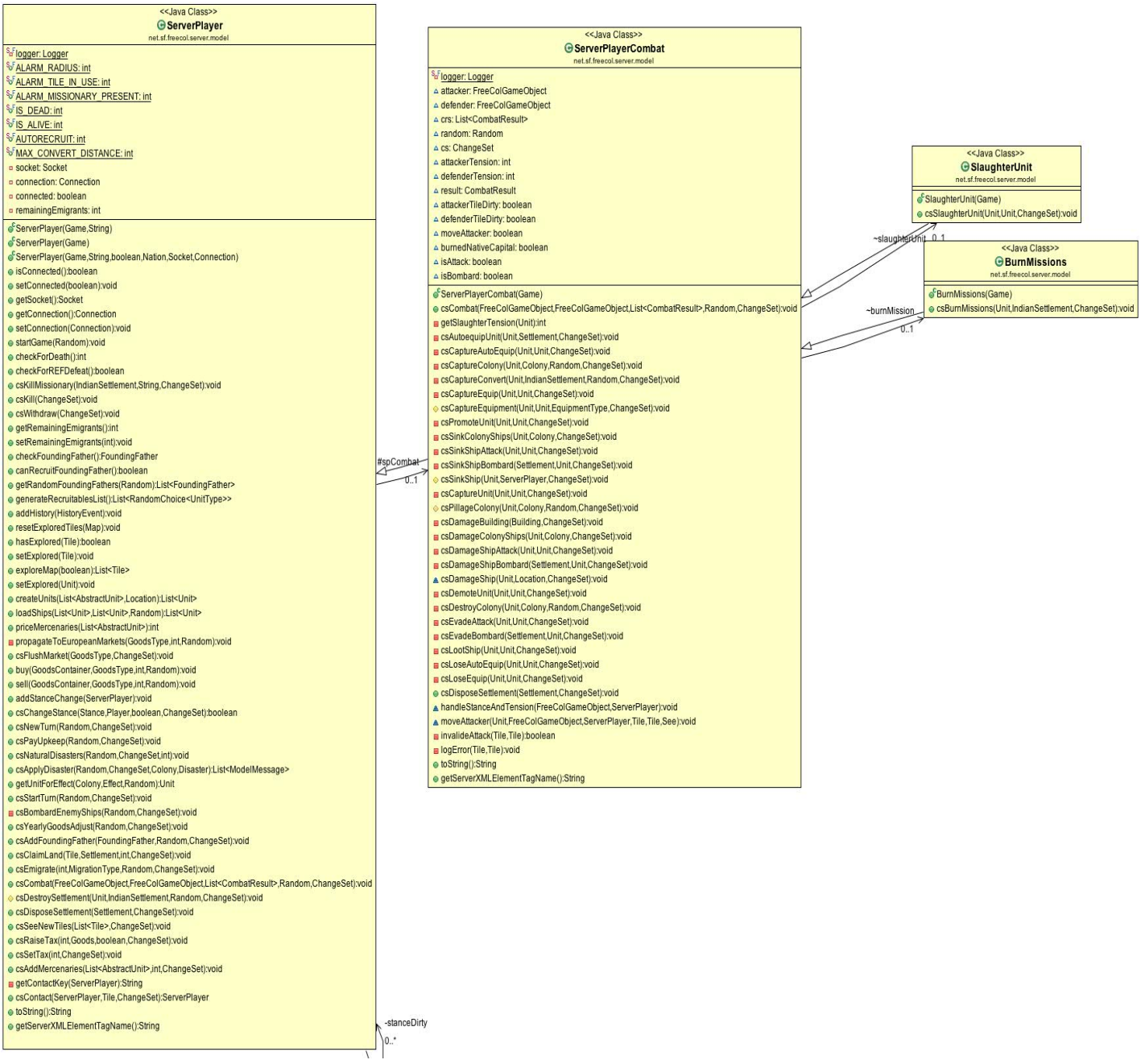

Figure 4.3 - The Refactored Class ServerPlayer. If the refactoring logic was continued we would see more smaller classes in the hierarchy but the principles of this refactoring

process have already reduced the size of both classes

The refactoring would of continue this way until there would be numerous small classes refactored from ServerPlayerCombat and more methods extracted from csCombat. The refactoring would then start again in the ServerPlayer to remove more common code into other Classes. These refactoring techniques would not only make the Server side code more readable it would also allow for an more extensible code where new rules could be added as classes and not affect the core code base of the ServerPlayer.

Bibliography

[1] Erich Gamma.. [et al.]. Design Patterns: Elements of Reusable Object-Oriented Software, Boston, Addison-Wesley, 1994

[2] http://en.wikipedia.org/wiki/Observer_pattern