D50: Advances in Software Engineering Design Patterns

Wolfgang Emmerich

Outline

- *Motivation*
- *What are Design Patterns?*
- *Observer: An Example Design Pattern*
- *Design Pattern Templates*
- *Suggested Design Pattern*
- *Summary*

Design Patterns

- *Good OO systems exploit recurring design structures.*
- *Design patterns capture, communicate and apply this structure.*
- *Must read: 'Design Patterns - Elements of Reusable Software' by Gamma, Helm, Johnson & Vlissides, Addison Wesley, 1995*

What is a Design Pattern?

"A pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

Christopher Alexander

Role of Design Patterns

- *To capture descriptions of good solutions to recurring design problems.*
- *To show how collections of classes and objects can be customised to solve a general design problem in a particular context.*

  (Patterns are not components.)

Philosophy of Design Patterns

- *Very deep!*
- *The "Quality Without a Name"*
- *Aliveness*
- *Familiarity*
- *Timelessness*
- *Self similarity*

Design Patterns and Software

- ***Compression (abstraction/classification)***
- ***About habitability and piecemeal growth.***
- ***Adaptable software***
- ***Repair, rather than replace***

Abstraction

"The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use."

Balzar et al

Abstraction cont...

“Abstraction facilitates separation of concerns: The implementation of an abstraction can ignore the exact uses or instances of the abstraction, and the user of the abstraction can forget the details of the implementation of the abstraction, so long as the implementation fulfills its intention or specification.”

Balance

- *Too little abstraction exposes too much detail to be understood.*
- *Too much abstraction creates highly interwoven and fragile “perfect” structures.*
- *To allow systems to adapt the abstractions must be malleable.*

Compression

- *Meaning of abstraction determined by the context it is in (cf poetry).*
- *Meaning of a part larger than the part by itself.*
- *Another way of describing abstraction and inheritance.*

Master Plans Don't Work

- *"It is simply not possible to fix today what the environment should be like in the future, and then to steer the piecemeal process of development toward that fixed, imaginary world." Christopher Alexander*
- *It is impossible to predict what will happen during the lifetime of a long-lived program.*
- *Use prototyping and patterns to promote adaptability.*

Design Pattern Structure

- *Name*
- *Abstract description of collaborations*
- *Issue(s) addressed*
- *Consequences of application*
- *Language and implementation independent - a “micro architecture”*

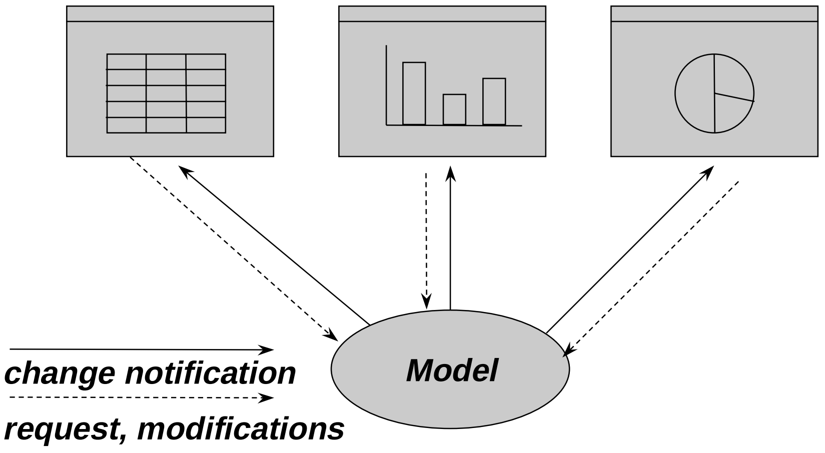
13

Design Pattern Properties

- *abstracts a recurring design structure*
- *comprises class and/or object*
  - *dependencies*
  - *structures*
  - *interactions*
  - *conventions*
- *names & specifies the design structure*
- *distils design experience*

14

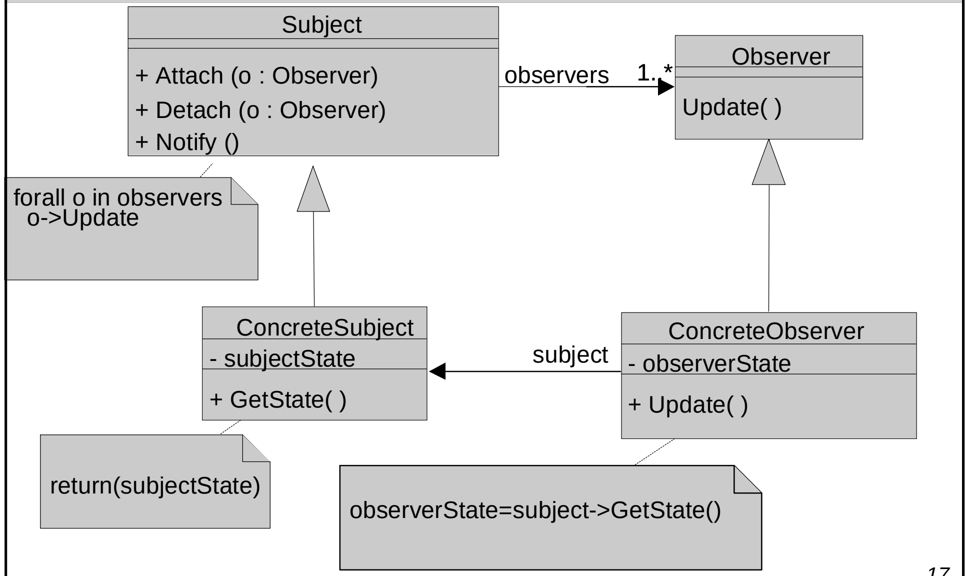
Example - Observer



Observer Pattern

- **_Intent:_**
  - **_define a one-to-many dependency between objects so that when one object changes state, all its dependants are notified and update automatically._**
- **_Applicability:_**
  - **_when an abstraction has two aspects, one dependent on the other._**
  - **_when a change to one object requires changes to others, and you don't know how many objects need to change._**
  - **_when an object should notify other objects without making assumptions about who those objects are._**

Observer Pattern



Observer Pattern

- ***Consequences:***
  - ***+ modularity: subject and observers may vary independently***
  - ***+ extensibility: can define and add any number of observers***
  - ***+ customisability: different observers provide different views of a subject.***
  - ***- unexpected updates: observers don't know about each other.***
  - ***- update overhead: might need hints***

Observer Pattern

- **Implementation**
  - *subject-observer mapping*
  - *dangling references*
  - *avoiding observer-specific update protocols*
  - *registering modifications of interest explicitly*
- **Known Uses**
  - *Smalltalk Model-View-Controller (MVC)*
  - *InterViews (Subjects and Views)*
  - *Andrew (Data Objects and Views)*

Design Pattern Goals

- **Codify good design**
  - *Distill and disseminate experience*
  - *Aid to both novices and experts*
  - *Abstract how to think about design*
  - *Provide a Design Language*

Design Language

- *Provide a common vocabulary*
- *Greater expressiveness*
- *More appropriate level of abstraction*
- *Design decisions can be articulated succinctly*
- *Documentation can be improved*

Design Pattern Template

- *Intent: short description of pattern and its purpose*
- *Also Known As: other names for pattern*
- *Motivation: motivating scenario showing pattern's use*
- *Applicability: circumstances in which pattern applies*
- *Structure: graphical representation of the pattern*
- *Participants: participating classes and/or objects and their responsibilities*

Template cont.

- ***Collaborations**: how participants co-operate to carry out responsibilities*
- ***Consequences**: the results of application, benefits and liabilities*
- ***Implementation**: pitfalls, hints or techniques, plus language dependency*
- ***Sample Code:** example implementations in OO language*
- ***Know Uses:** examples drawn from existing systems*
- ***Related Patterns:** discussion of other patterns that relate to this one.*

Suggested Patterns:

- ***Abstract Factory (87): Provide an interface for creating families of related or dependent objects without specifying their concrete classes.***
- ***Adapter (139): Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.***
- ***Bridge (151): De-couple an abstraction from its implementation so that the two can vary independently***

Suggested Patterns (cont'd)

- ***Builder (97): Separate construction of complex object from its representation so that same construction process can create different representations.***
- ***Chain of Responsibility (223): Avoid coupling request sender to receiver by giving more than one object chance to handle request. Chain receiving objects and pass request along chain until an object handles it.***

Suggested Patterns (cont'd)

- ***Command (233): Encapsulate request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.***
- ***Composite (163): Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly***

Suggested Pattners (cont’d)

- Decorator (175): Attach additional responsibilities to object dynamically. Provide flexible alternative to subclassing for extending functionality
- Facade (185): Provide unified interface to a set of interfaces in a subsystem. Facade defines higher-level interface that makes subsystem easier to use.
- Factory Method (107): Define interface for creating object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

Suggested Patterns (cont’d)

- Flyweight (195): Use sharing to support large numbers of fine-grained objects efficiently.
- Interpreter (243): Given a language, define representation for its grammar along with interpreter that uses the representation to interpret sentences in the language.
- Iterator (257): Provide a way to access elements of an aggregate object sequentially without exposing its underlying representation.

Suggested Patterns (cont'd)

- Mediator (273): Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interaction independently
- Memento (283): Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Suggested Patterns (cont'd)

- Observer (293) Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- Prototype (117) Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Proxy (207) Provide a surrogate or placeholder for another object to control access to it.

Suggested Patterns (cont'd)

- ***Singleton (127): Ensure a class only has one instance, and provide a global point of access to it.***
- ***State (305): Allow an object to alter its behaviour when its internal state changes. The object will appear to change its class.***
- ***Strategy (315): Define family of algorithms, encapsulate each one, and make them interchangeable. Let algorithm vary independently from its clients.***

Suggested Patterns (cont'd)

- ***Template Method (325): Define skeleton of algorithm in operation, deferring some steps to subclasses. Lets lets subclasses redefine steps of algorithm without changing algorithm's structure.***
- ***Visitor (331) Represent operation to be performed on the elements of an object structure. Define a new operation without changing the classes of the elements on which it operates.***

Summary

- Motivation
- What are Design Patterns?
- Observer: An Example Design Pattern
- Design Pattern Templates
- Suggested Design Pattern
- Next Session: Look at some Design Patterns in detail