Inf2-SEPP:
Lecture 11: Design Patterns: MVC, Observer

Cristina Adriana Alexandru

School of Informatics
University of Edinburgh

Previous lectures

- Design
  - Class diagrams
  - Sequence diagrams

Both important for this lecture

This lecture

Design patterns

- Meaning, background and use
- Elements of a pattern
- Cautions on pattern use
- Architectural pattern: The Model View Controller (MVC)
- Detailed design pattern (behavioural): Observer
  - The problem
  - Details
  - Advantages
  - Disadvantages

Design Patterns

"Reuse of good ideas"

A pattern is a named, well understood good solution to a common problem.

- Experienced designers recognise variants on recurring problems and understand how to solve them.
- They communicate their understanding by recording it in design patterns
- Such patterns then help novices avoid having to find solutions from first principles.

Patterns help novices to learn by example to behave more like experts.

Patterns: background and use

Idea comes from architecture (Christopher Alexander): e.g. **Window Place:** observe that people need comfortable places to sit, and like being near windows, so make a comfortable seating place at a window.

Similarly, software design patterns address many commonly arising technical problems in software design, particularly OO design

Patterns also used in: reengineering; project management; configuration management; etc.

Pattern catalogues: for easy reference, and to let designers talk shorthand.

Elements of a pattern

A pattern catalogue entry normally includes roughly:

- Name (e.g. Publisher-Subscriber)
- Aliases (e.g. Observer, Dependants)
- Context (in what circumstances can the problem arise?)
- Problem (why won't a naive approach work?)
- Solution (normally a mixture of text and models)
- Consequences (good and bad things about what happens if you use the pattern.)

Cautions on pattern use

Patterns are very useful *if you have the problem they're trying to solve*.

But they add complexity, and often e.g. performance penalties too. Exercise discretion.

You'll find the criticism that the GoF patterns in particular are "just" getting round the deficiencies of OOPLs. This is true, but misses the point.

(GoF = "Gang of Four", authors of the first major Design Patterns book)

Model View Controller (MVC): the problem

Context: architectural design

Reminders:

- The more complex a system is, the less maintainable, harder to understand, error prone, less secure.
- Complexity can increase at a high speed: the more components, the even more relationships between them
- Related concept of coupling; The more relationships, the higher the coupling
- Design guidelines/principles:
  - Separation of concerns: components doing only one thing; grouping components with related functionality
  - Keeping coupling low

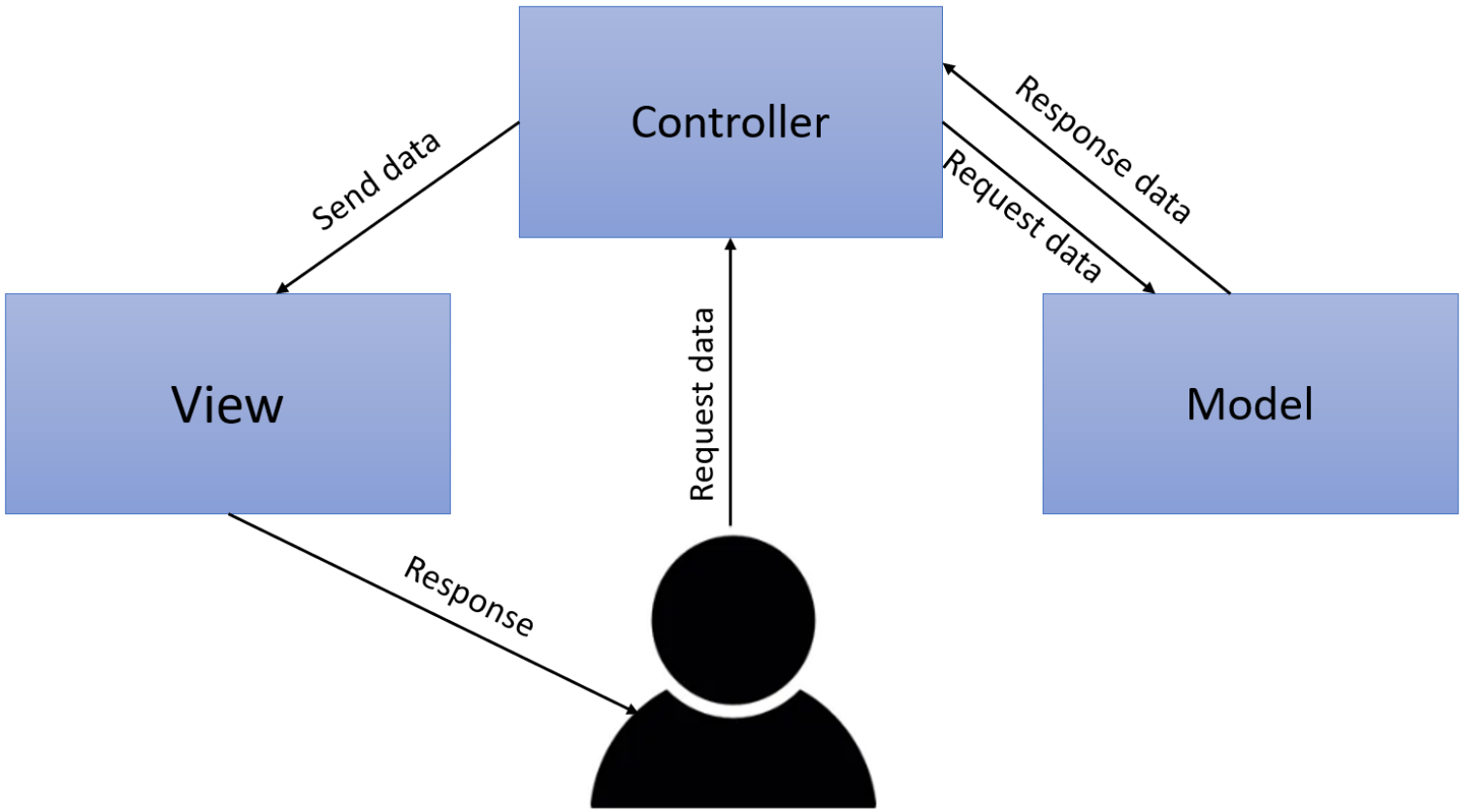
Especially a problem for large scale systems (over 100K LOC)

Model View Controller (MVC): the solution

Split the application into 3 components:

- Model: manages data and the domain logic of the application. Communicates with the controller. Usually interacts with a data source (database, input file, etc.). Can sometimes update the view (not in version from this course).
- View: defines and manages how data is presented to the users. There can be several views.
- Controller: receives input from the user, handles application logic, acts as middleman between model and view.

Model View Controller (MVC): an example interaction



Model View Controller (MVC): advantages and use

- Facilitates the separation of concerns, as each component has distinct responsibilities
- Decouples presentation (the view) from data and domain logic (the model)
- Multiple developers can work in parallel on the different components
- Easier to understand, maintain, less error prone
- Easier to test
- Supports multiple views, ideal for web applications

One of most popular architectures for web applications, used in numerous web frameworks: Ruby on Rails, Angular, Django, Flask.

Observer pattern: the problem

Context: detailed design

Problems:

1. Maintaining state consistency between a set of cooperating classes, i.e. *dependant* classes being informed about the state changes of *subject* classes.
2. Easily adding and removing dependants without changing the subjects (i.e. not knowing of who dependants are).

Example: Changing the way information on students (the subjects) is presented in a bar chart vs pie chart (dependants).

Observer is often used in event driven software, and in MVC pattern to represent the 'view' part.

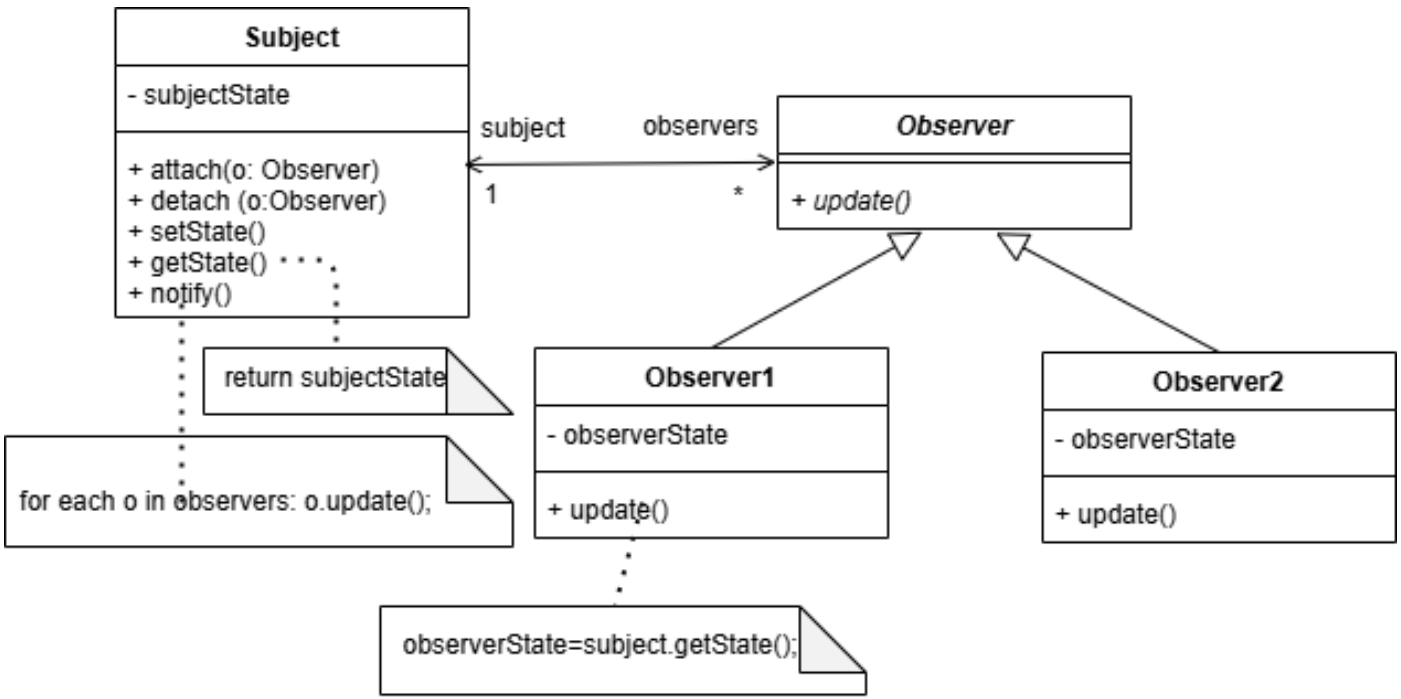
Observer pattern: the problem

Naïve solution for problem 1): associating each subject with each of its dependants

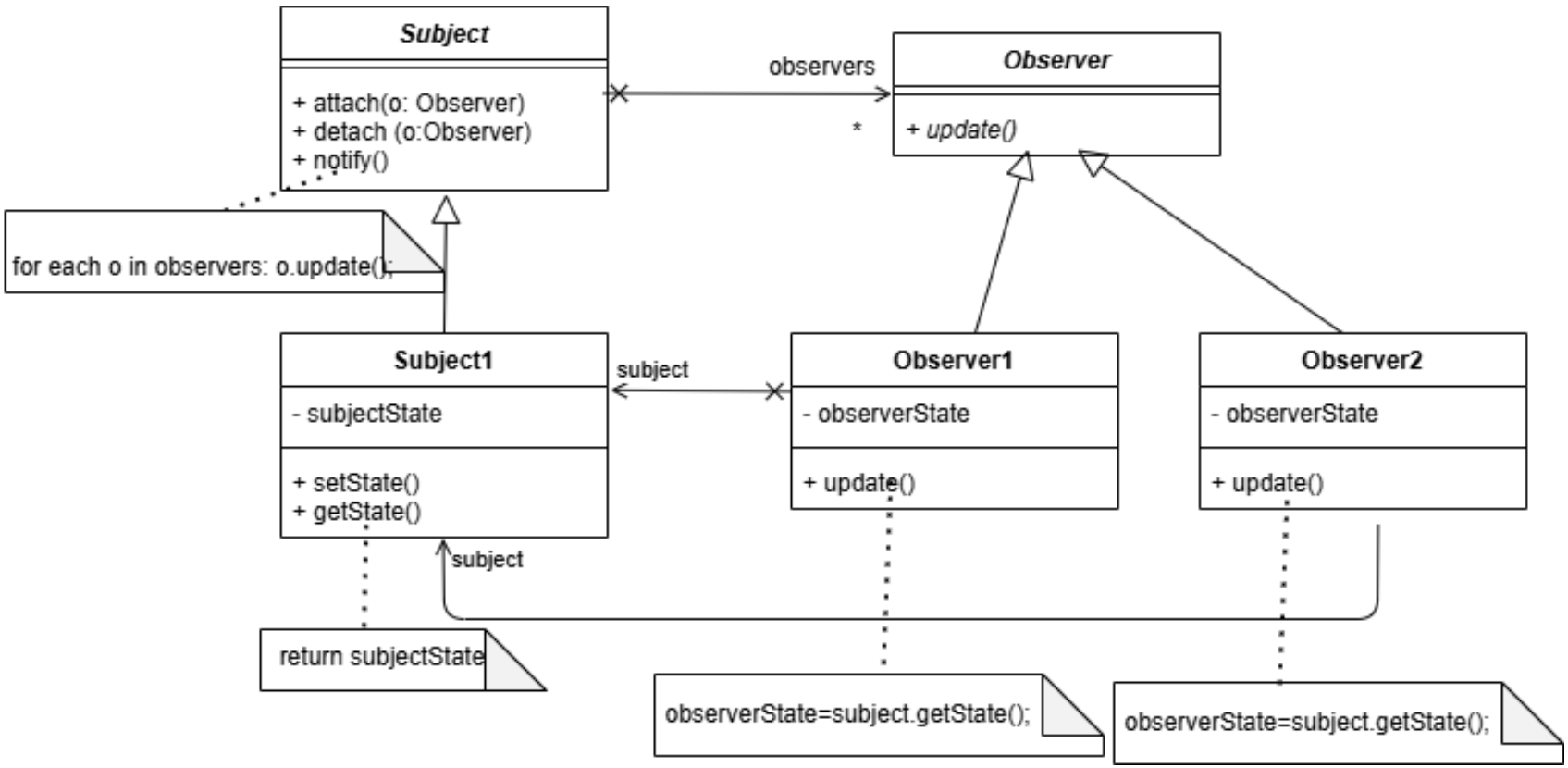
BUT this leads to *tight coupling*, not respecting problem 2):

- The subject must know of its dependants and their number
- The subjects may need updating when dependants are updated.

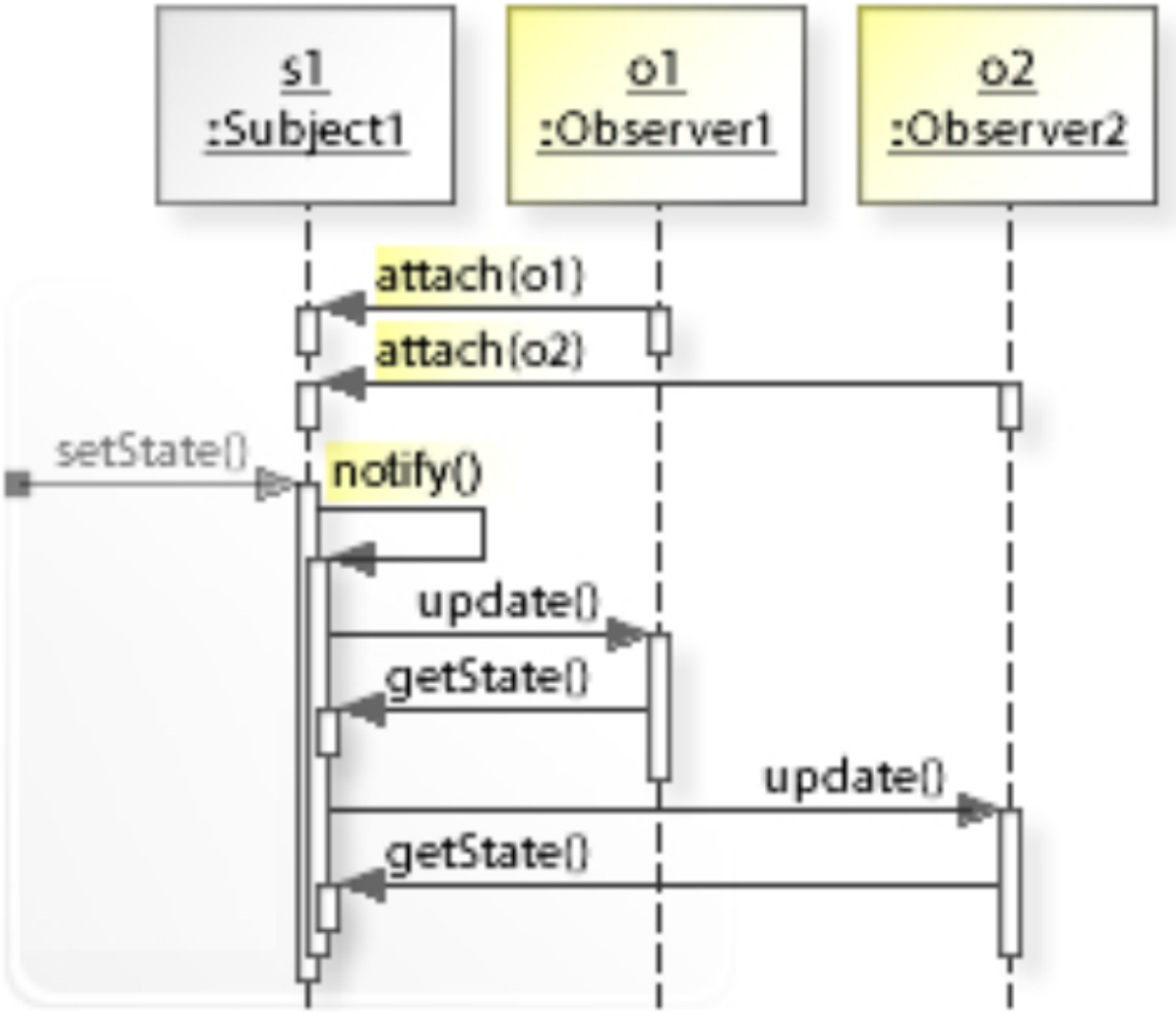
Observer pattern with one subject



Observer pattern with potentially more subjects



Observer pattern with potentially more subjects



Observer pattern: Advantages and disadvatages

Advantages:

- Abstract, minimal subject-observer class coupling
- Support for broadcasting, without the subject needing to know and inform each observer

Disadvantages:

- May lead to cascades of updates which are difficult to debug
- Costly in terms of space if many subjects and few observers; One solution: use of hash maps, costly in terms of time.
- Ending up with dangling references to deleted subjects/ observers; One solution: notifying when deleted
- Risk of having an inconsistent subject state before notification

Resources

Recommended: Read more on design patterns in general, e.g.

- Stevens: Ch18.2
- Sommerville: Look up *design patterns* in index
- `http://en.wikipedia.org/wiki/Design_Patterns`

Essential: Read on the MVC and Observer patterns:

- On the MVC architectural pattern: from YouTube
- If you can get a copy of Gamma, E., 1995. Design patterns: elements of reusable object-oriented software. Pearson Education India: p. 326-337 "Observer"
- On the Observer pattern: from Source Making and Wikipedia