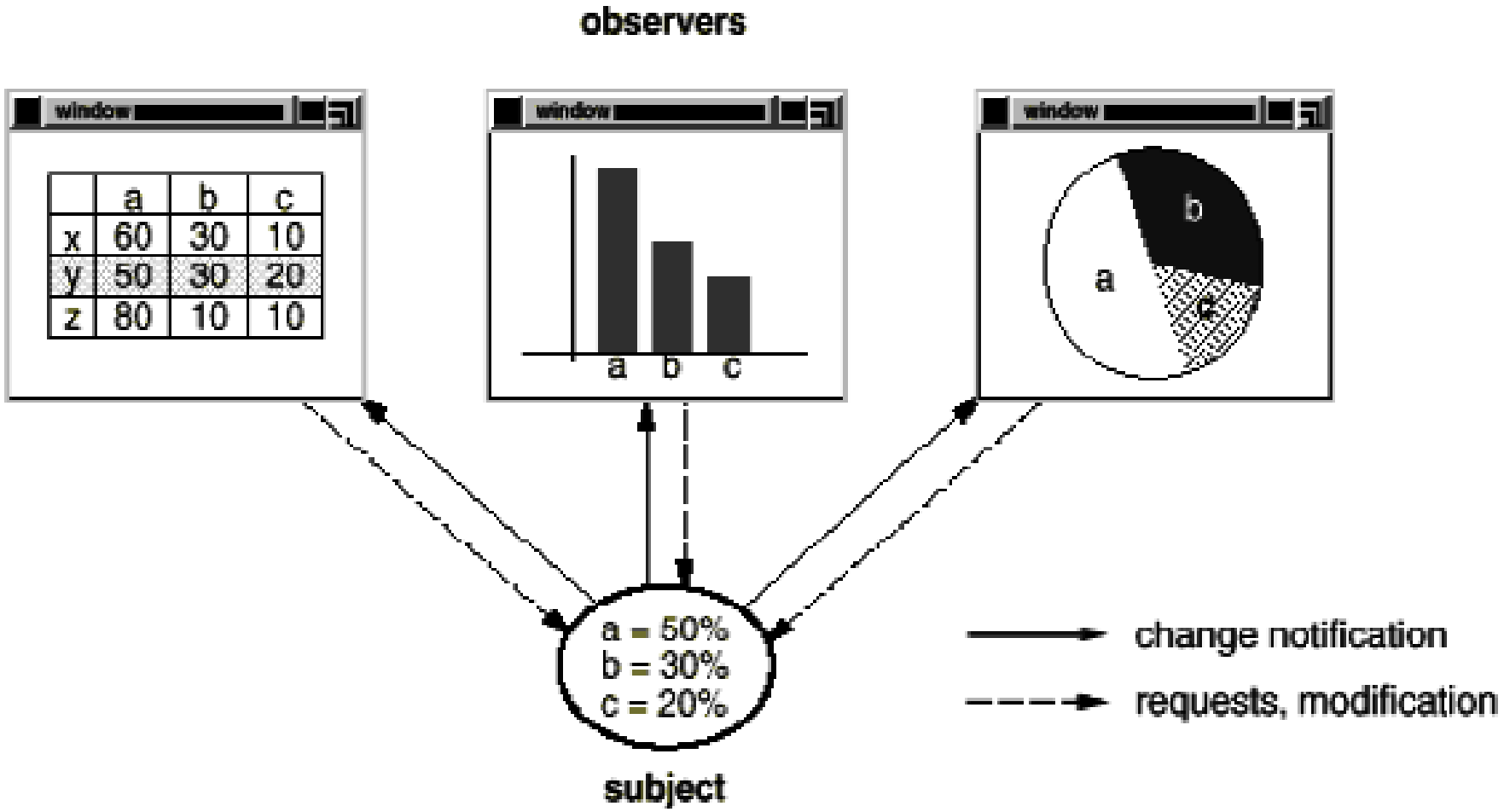
Tecniche di Progettazione: Design Patterns

GoF: MVC e Observer

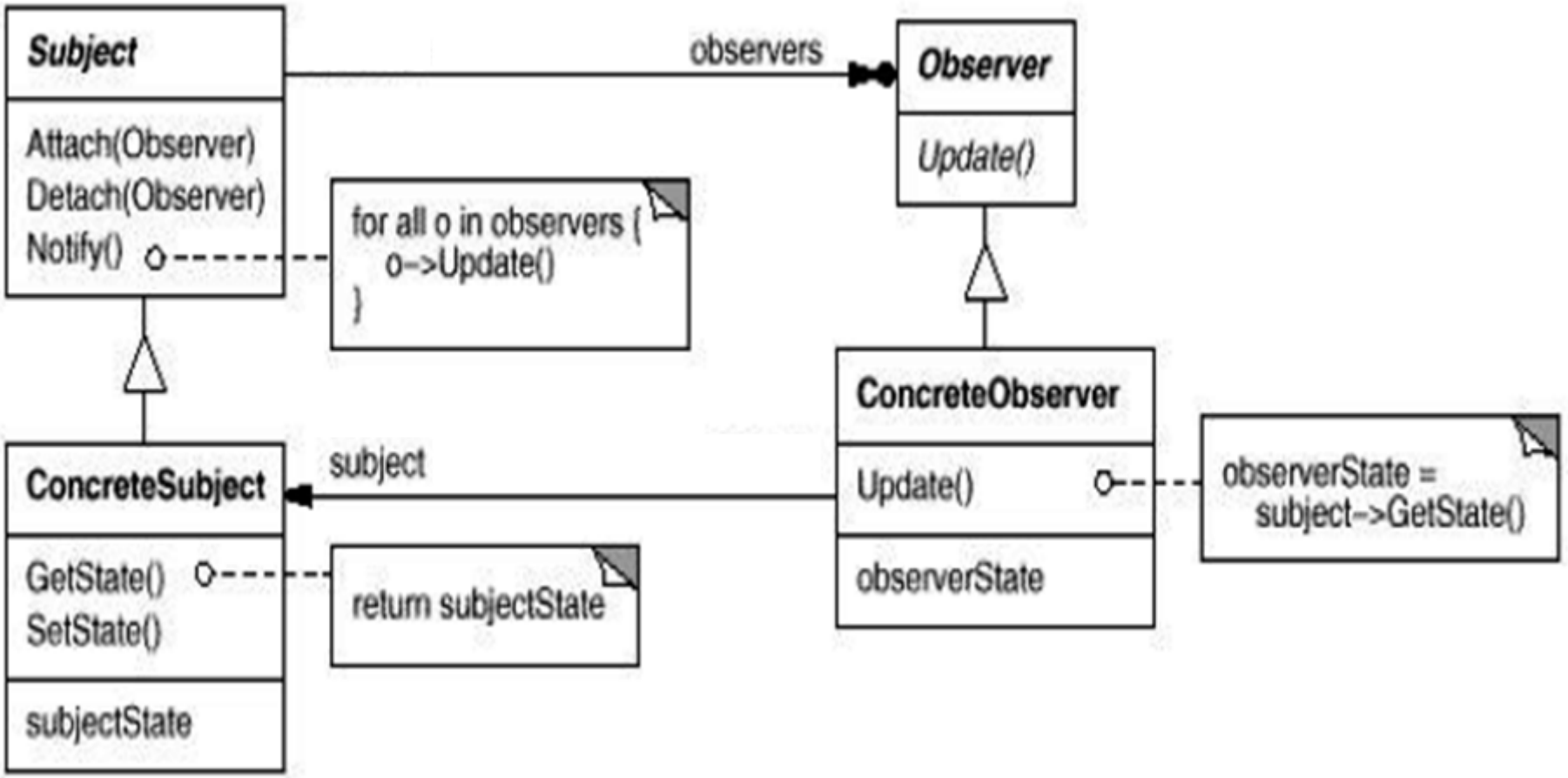
The Observer Pattern

- Intent
  - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- AKA
  - Dependents, Publish-Subscribe, Model-View
- Motivation
  - The need to maintain consistency between related objects without making classes tightly coupled

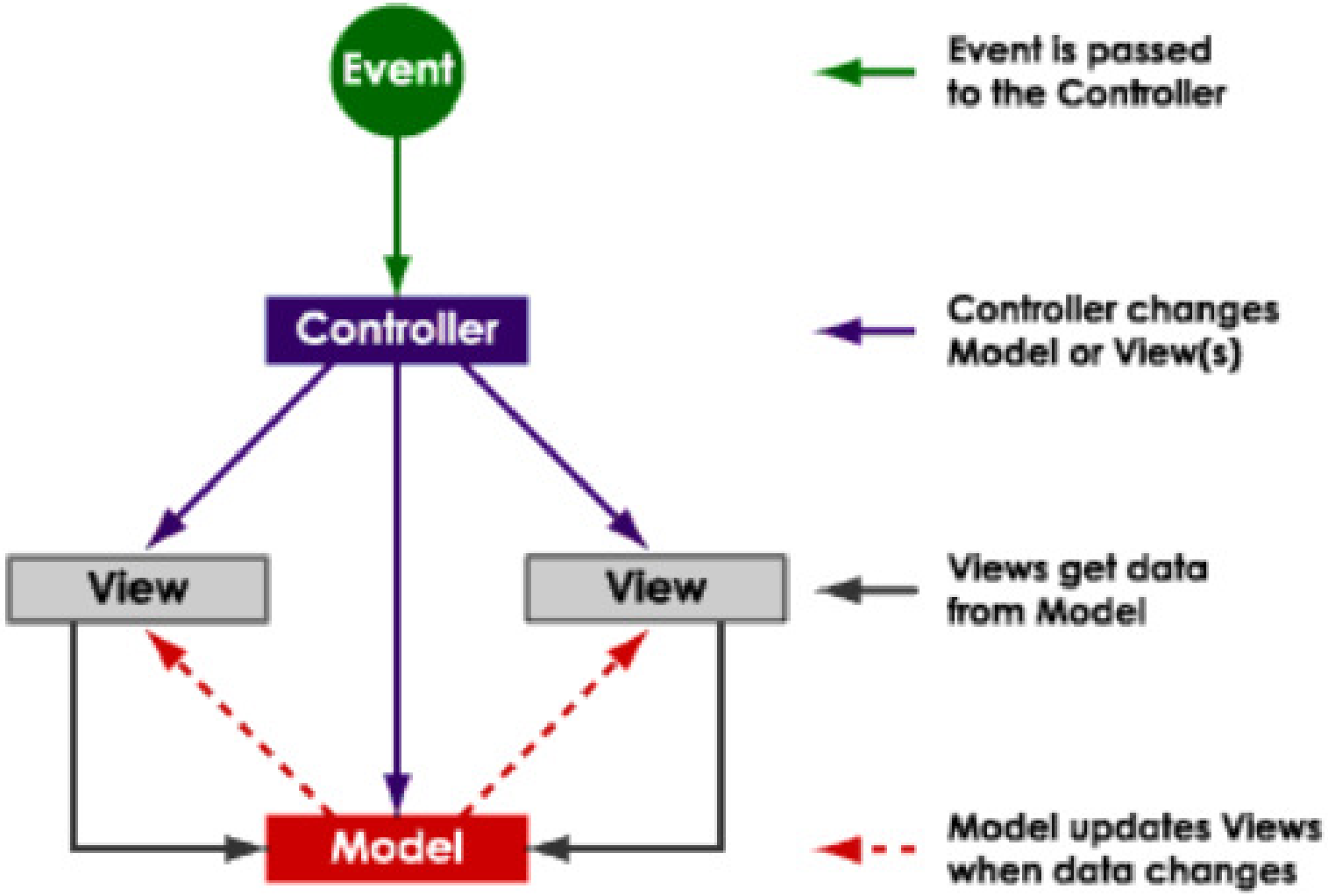
Example


observers
window
window
window
a b c
x 60 30 10
y 50 30 20
z 80 10 10
b
a
c
a = 50%
b = 30%
c = 20%
subject
change notification
requests, modification

Structure

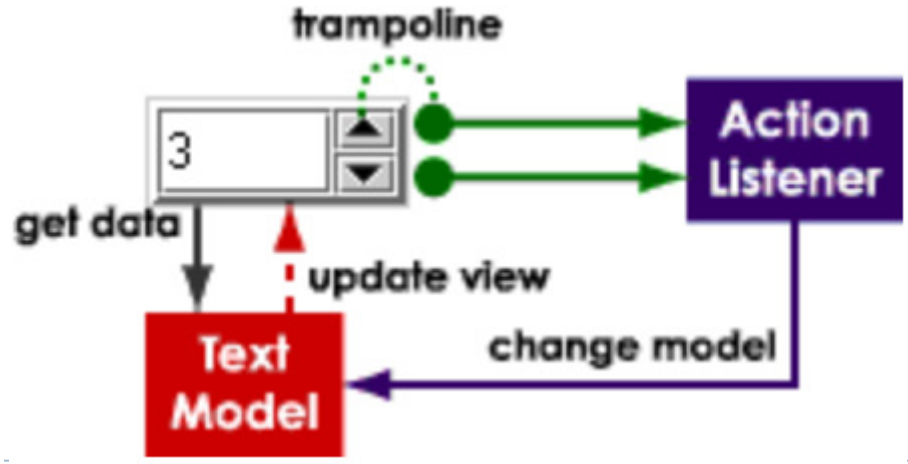

Subject
Attach(Observer)
Detach(Observer)
Notify()
observers
Observer
Update()
for all o in observers {
o->Update()
}
ConcreteObserver
Update()
observerState
observerState =
subject->GetState()
ConcreteSubject
GetState()
SetState()
subjectState
subject
return subjectState

Model View Controller



MVC: counter

- Counter component which consists of a text field and two arrow buttons that can be used to increment or decrement a numeric value shown in the text field.
- The counter's data is held in a model that is *shared* with the text field. The text field provides a view of the counter's current value. Each button is an event source, that spawns an action event every time it is clicked. The buttons can be hooked up to trampolines that receive action events, and route them to an action listener that eventually handles that event. Recall that a trampoline is a predefined action listener that simply delegates action handling to another listener.



MVC1: CounterGui

```
/**
* Class CounterGui demonstrates having the model and view in the same class
*/

import java.awt.*;
import java.awt.event.*;

public class CounterGui extends Frame
{
    //the counter (the model!)
    private int counter = 0;

    //the view/
    private TextField tf = new TextField(10);

//…… see project
```

MVC2: CounterView & Counter

- This example shows the model and the view in separate classes. First the view class:
- /**
- * Class CounterView demonstrates having the model and view
- * in the separate classes. This class is just the view.
- */
- public class CounterView extends Frame {
- // The view.
- private TextField tf = new TextField(10);
- // A reference to our associated model.
- private Counter counter;

MVC2: CounterView & Counter

- public CounterView(String title, Counter c) {
- super(title);
- counter = c;
- Panel tfPanel = new Panel();
- tf.setText(counter.getCount()+ "");
- tfPanel.add(tf);
- add("North",tfPanel);
- Panel buttonPanel = new Panel();
- Button incButton = new Button("Increment");
- incButton.addActionListener(new ActionListener() {
  - public void actionPerformed(ActionEvent e) {counter.incCount();
    - tf.setText(counter.getCount() + "");

  } });
- buttonPanel.add(incButton);

MVC3: applying Observer

```
/**
* Class ObservableCounter implements a simple observable
* counter model.
*/
public class ObservableCounter extends Observable {

/**
* Class ObservableCounterView demonstrates having the model
* and view in the separate classes. This class is just the view.
*/
 ….. counter.addObserver(new Observer() {
                public void update(Observable src, Object obj) { …….
```