Observer Pattern – Behavioural
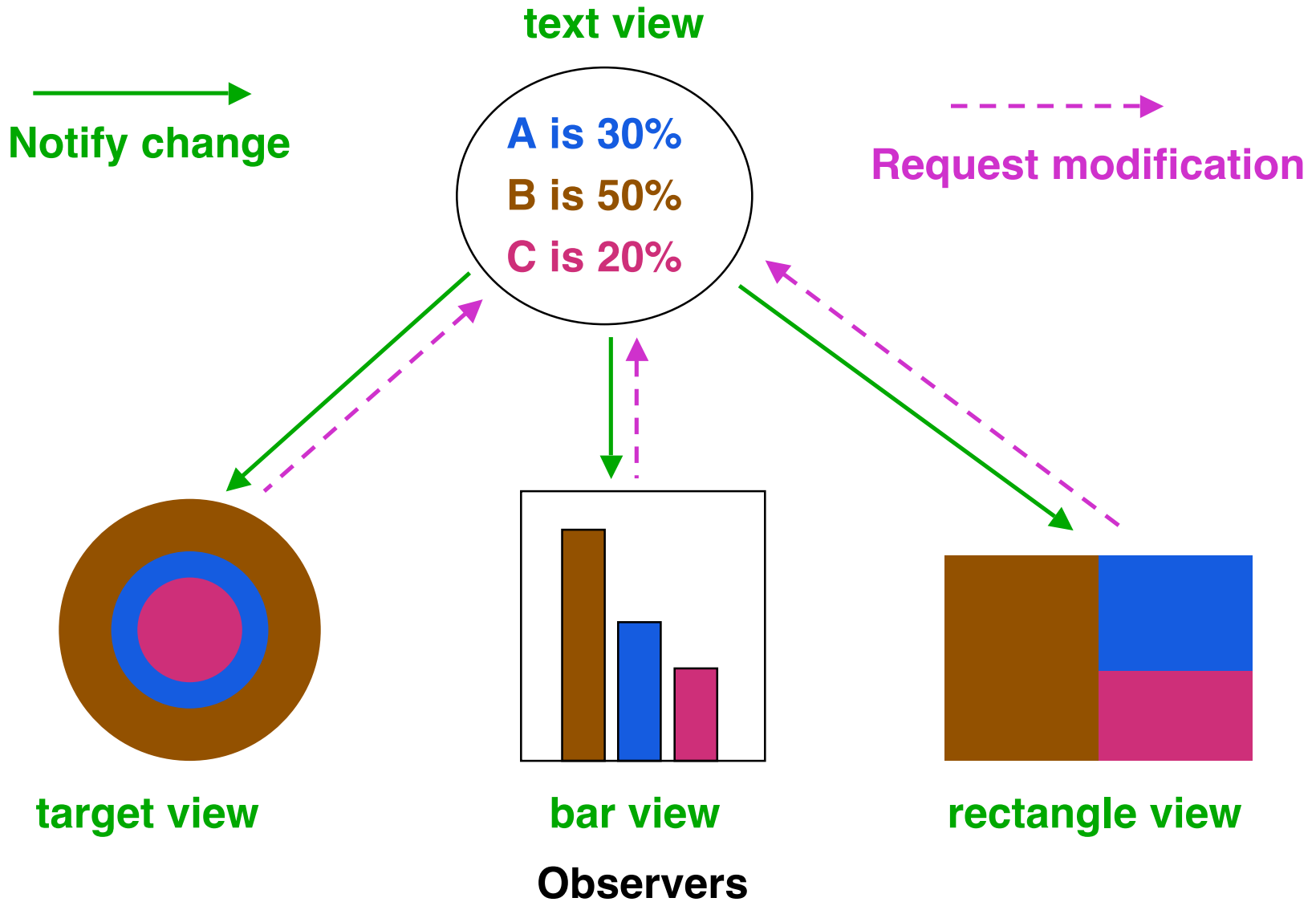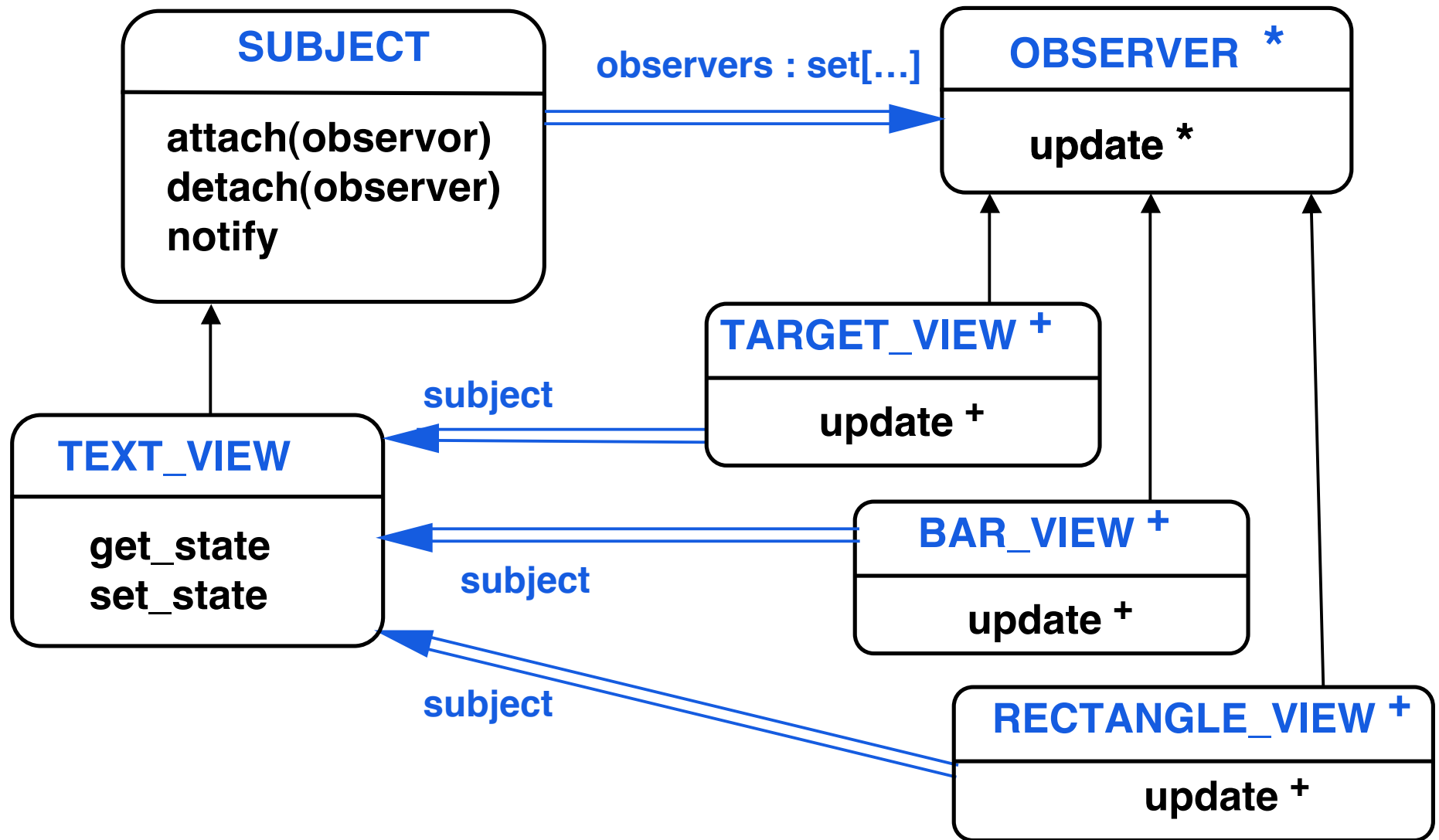
- Intent

  » **Define one-to-many dependency**

    > **When one subject changes state, all observers (dependents) are notified and correspondingly updated**
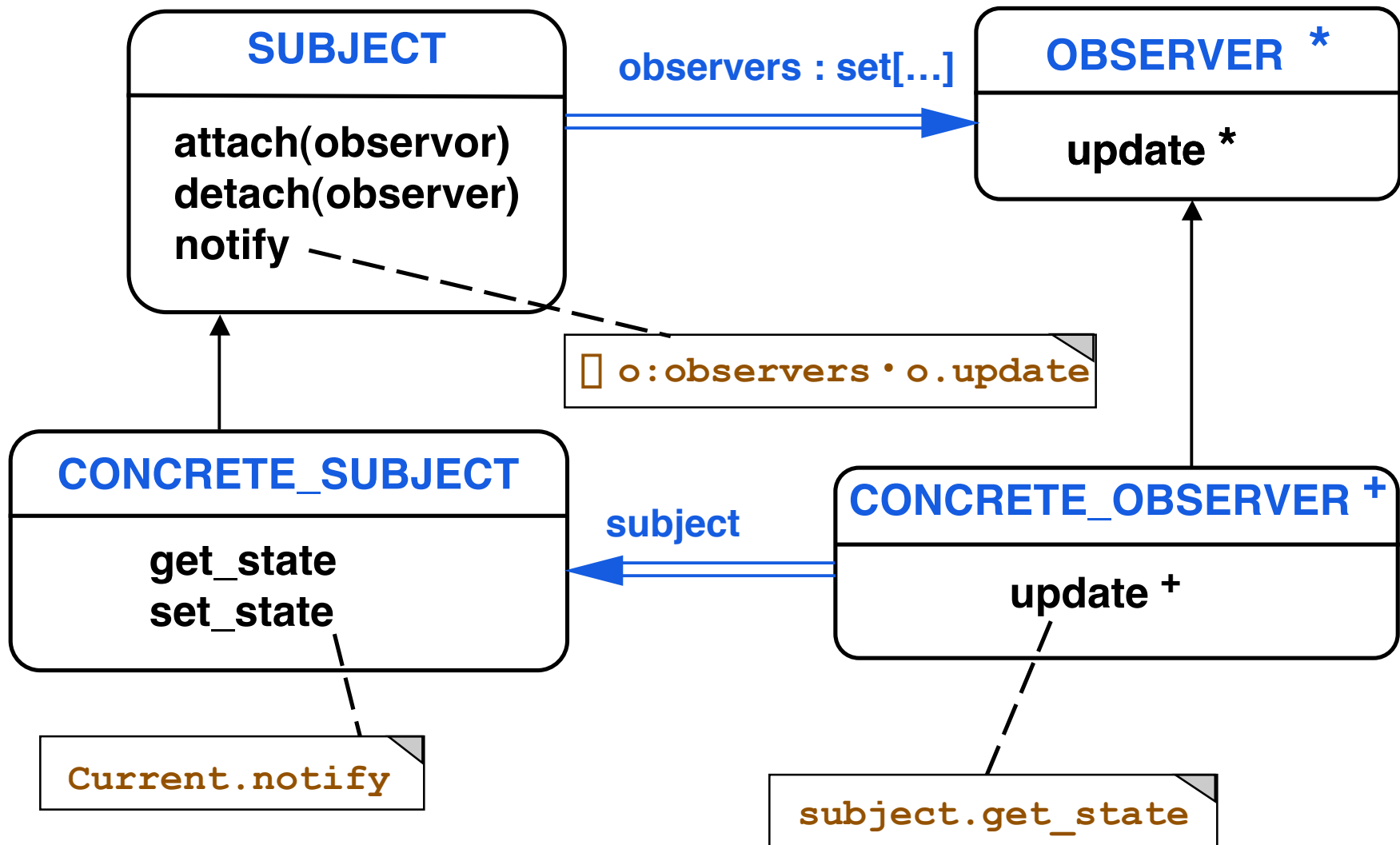
- Also known as

  » **Dependents and Publish-Subscribe**

Observer –!Motivation

text view

A is 30%
B is 50%
C is 20%

Notify change

Request modification

target view

bar view

rectangle view

Observers

Observer Architecture – Example

SUBJECT

attach(observor)
detach(observer)
notify

observers : set[...] →

OBSERVER *

update *

TARGET_VIEW +

update +

subject

TEXT_VIEW

get_state
set_state

BAR_VIEW +

update +

subject

RECTANGLE_VIEW +

update +

subject

© Gunnar Gotshalks

Observer-3

Observer – Abstract Architecture

SUBJECT

attach(observor)
detach(observer)
notify

observers : set[...] →

OBSERVER *

update *

∀ o:observers•o.update

CONCRETE_SUBJECT

get_state
set_state

subject ←

CONCRETE_OBSERVER +

update +

Current.notify

subject.get_state

Observer – Applicability

- When an abstraction has two aspects, one dependent upon the other

  - » **Encapsulating each aspect as a separate object means you can change and use them independently**

- When changing one object requires changing an indeterminate number of corresponding objects

- When an object needs to notify other objects without making detailed assumptions about those objects, to reduce coupling

Observer – Participants

- Subject

  » **Knows its observers**

  » **Provides interface for attaching, detaching and notifying its observers**

- Observer

  » **Defines an updating interface for observers**

Observer – Participants –!2

- Concrete subject

  » **Stores state of interest to concrete observers**

  » **Notifies observers when state changes**

- Concrete observer

  » **Maintains a reference to its concrete subject**

  » **Stores state that corresponds to the state of the subject**

  » **Implements Observer updating interface**

Observer –!Scenario

- Concrete subject updates all observers, when state is changed by a client

Scenario: **Update observers**
1 set_state
2 notify
3 update
4 get_state

CLIENT

1

2

CONCRETE_SUBJECT

3

4

CONCRETE_OBSERVER
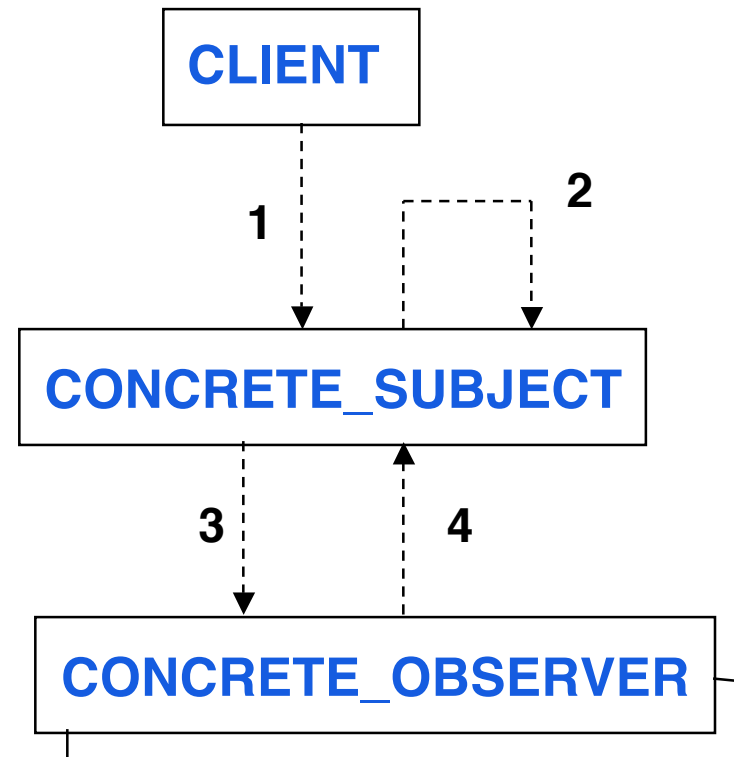
Observer –!Consequences – 1

- Abstract coupling between subject and observer

  » **Permits changing number of observers dynamically**

  » **Subject and observer can belong to different layers**

  > **If they are in one class, then the object spans system layers, which can compromise abstraction by layering**

- Supports broadcast communication

- Can have observers depend upon more than one subject

Observer –!Consequences – 2

- Observers may also change the state

  » **Can be expensive as observers are unaware of each other**

- Need additional protocol to indicate what changed

  » **Can have spurious updates**

  > **Not all observers participate in all changes**

  » **Can have clients notify, instead of subject, as clients understand better when updates are needed**

  > **Leads to errors as clients can forget to update**

Observer –!Consequences – 3

- Dangling references when subject is deleted

  » **Notify observers when subject is deleted**

  > **Cannot delete observers as other subjects may depend upon them**

- Update only when subject state is consistent with respect to observer

  » **Could be violated when subclasses invoke inherited operations**

Observer – Related Patterns

- Mediator pattern is used for change managers

  » **Change manage mediates between subjects and observers**

- Singleton pattern is can be used to make a change manger unique and globally accessible