How did we get the presented system structure?

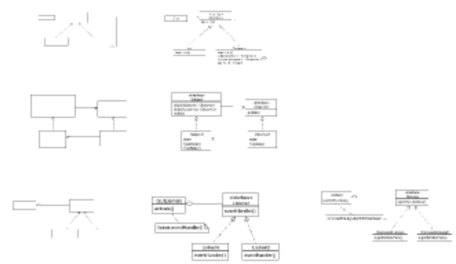
Where do other elements that start to occur in the class diagram come from?

- Aggregation/composition
- Dependency
- Association roles
- Interface realization and use
- Separating the user interface from the application logic

(Rational) Unified Process

entity - control - boundary

Where should we look for Model-View-Controller in a class diagram?



MVC = Observer + Strategy + Composite

Observer: the Model-View relationship (Model - Subject, View - Observer)

Strategy: the View-Controller relationship

Composite: nested Views (GoF)

Unified Process class stereotypes: a preparation for MVC, but controller is control

Software architecture

A part of the software system structure can be derived from use cases

Some decisions about the structure are generic (given by the realization domain)

A more stable part of the software system structure comes from the application domain

The software system structure copies the communication paths among the people who are developing this system

fiit.stuba.sk/~vranic/dot

Use cases

Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Conway's law

Lecture 3:

Software Architecture and Use Cases

Valentino Vranić

Ústav informatiky, informačných systémov
a softvérového inžinierstva

STU FIIT

vranic@stuba.sk fiit.sk/~vranic

MSOFT 2019/20
8. 10. 2019

How did we get the presented system structure?

A part of the software system structure can be derived from use cases

Software a

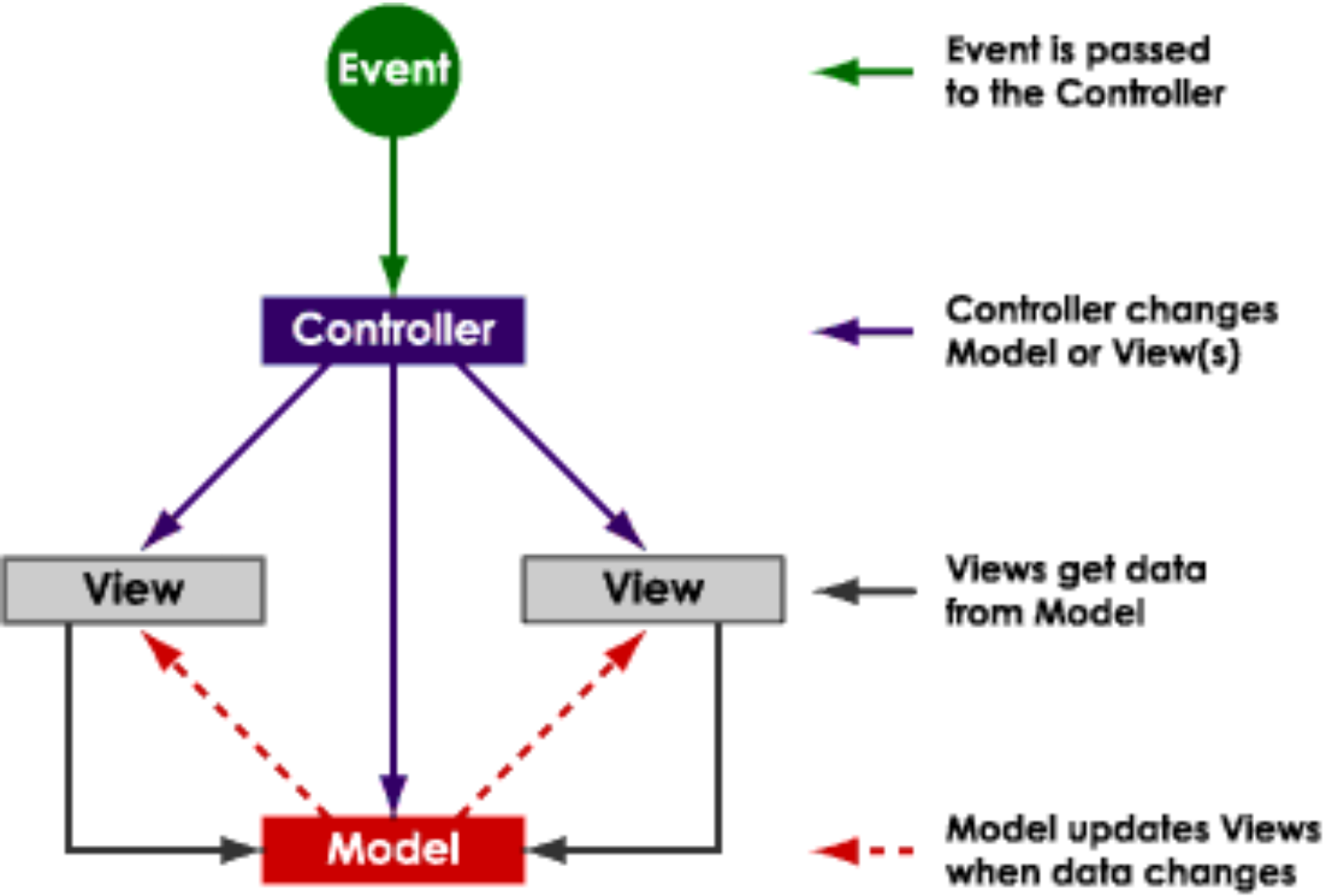
Where do other elements that start to occur in the class diagram come from?

A more stable part of the software system structure comes from the application domain.

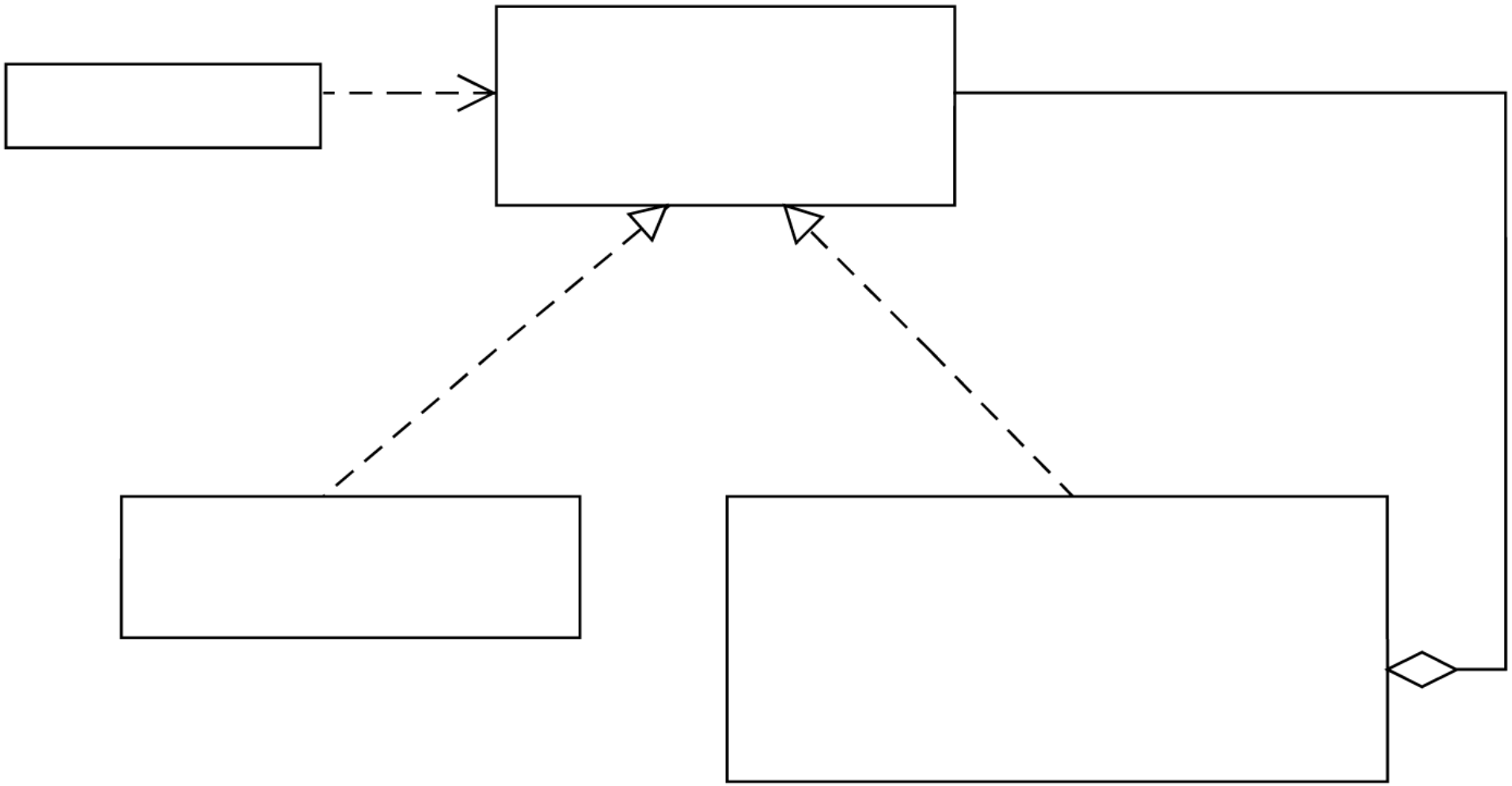
> Agregation/attribute
> Dependency
> Asociation roles
> Interface realization and use
> Separating the user interface from the application logic

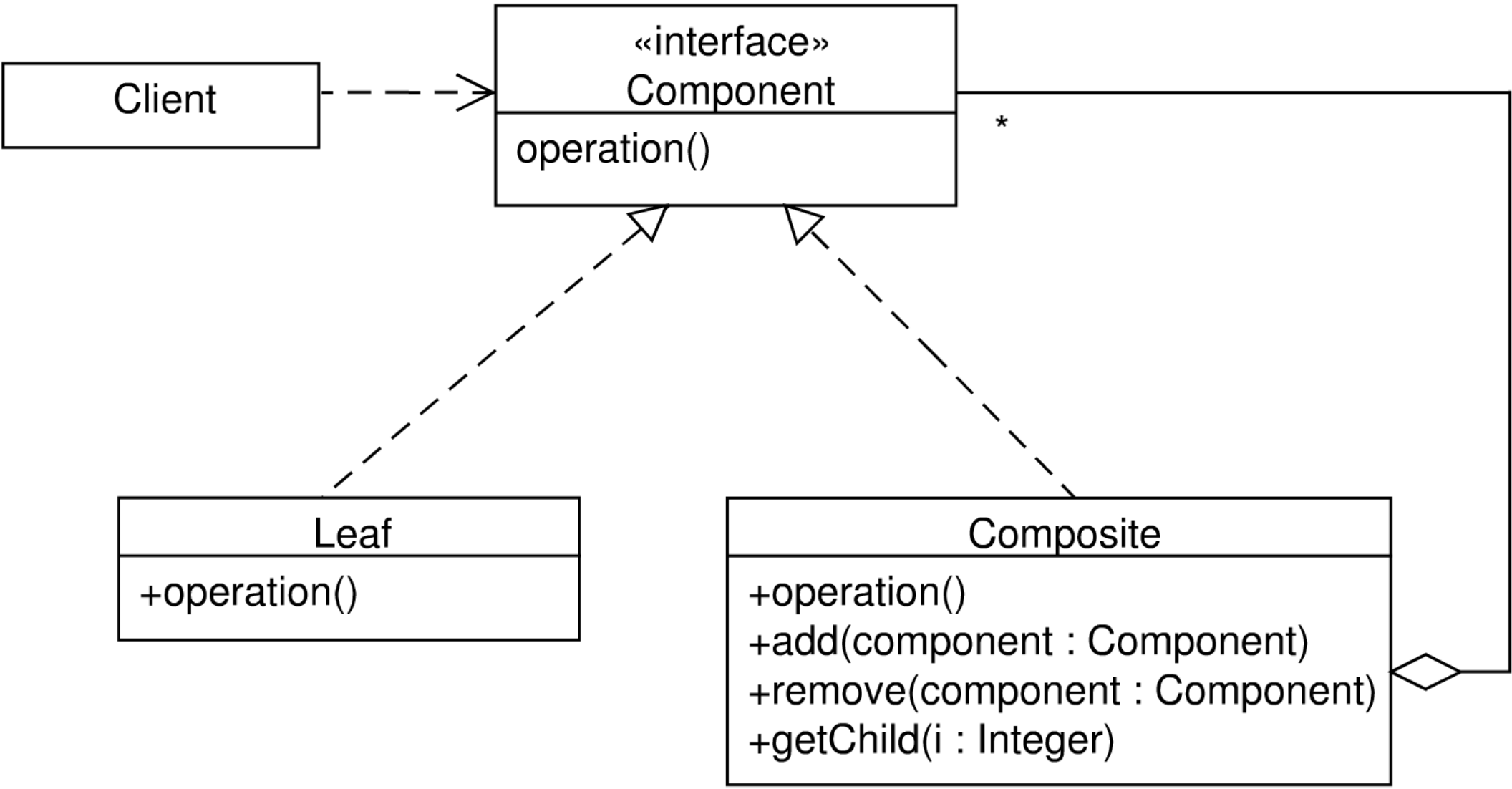
(Rational) Unified Process:

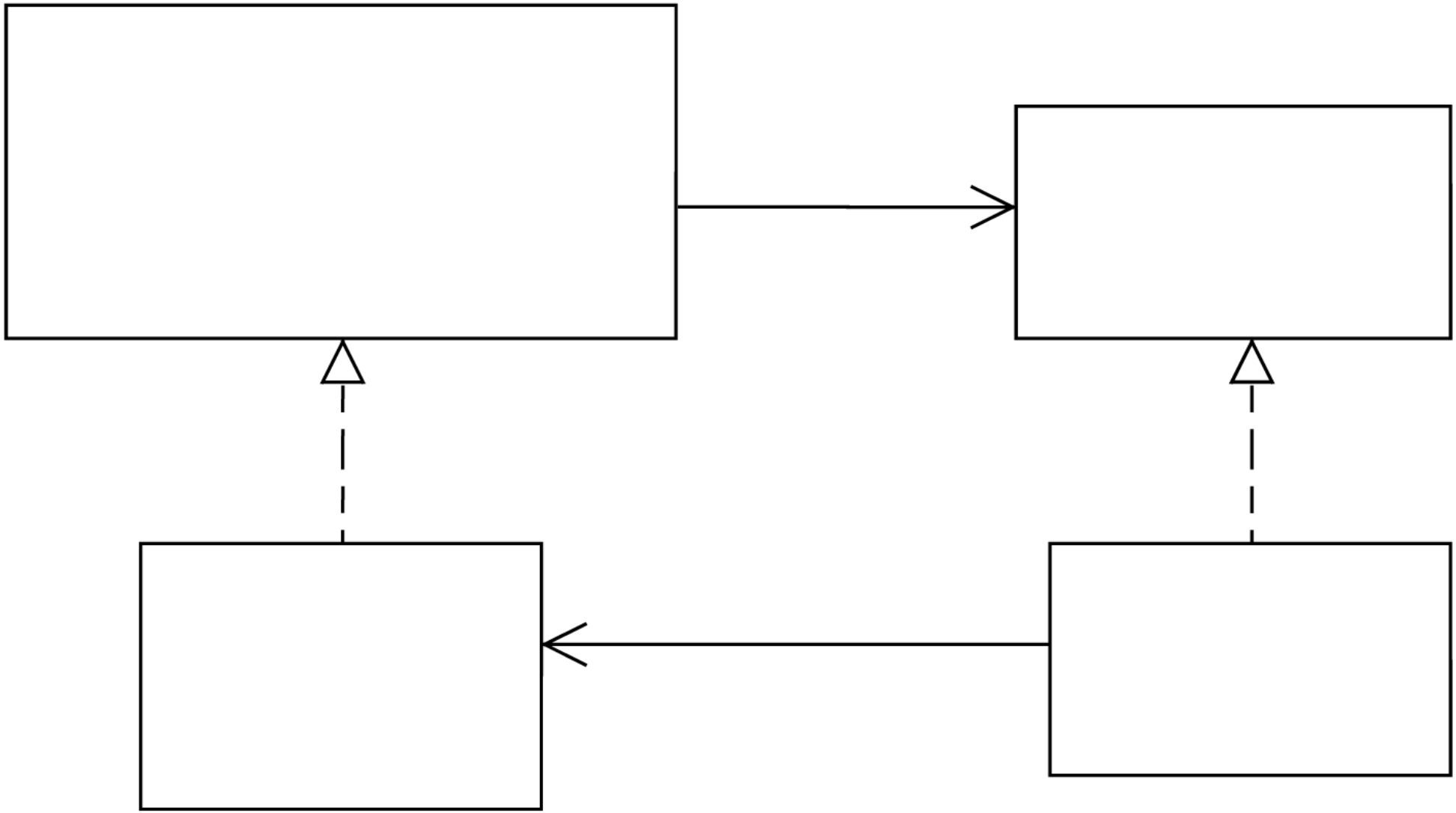
entity - control - boundary

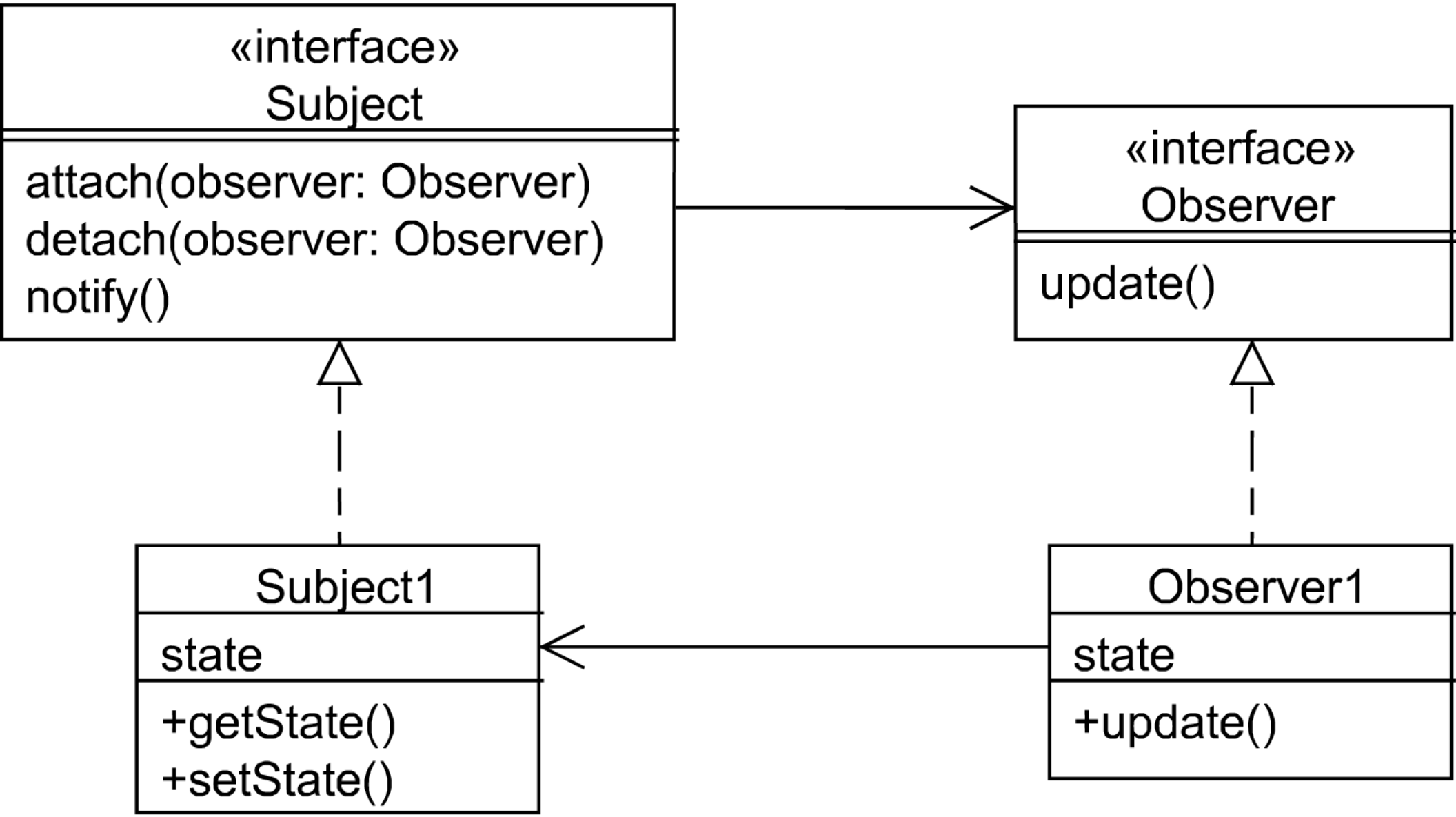

Event
Controller
View
View
Model
Event is passed to the Controller
Controller changes Model or View(s)
Views get data from Model
Model updates Views when data changes

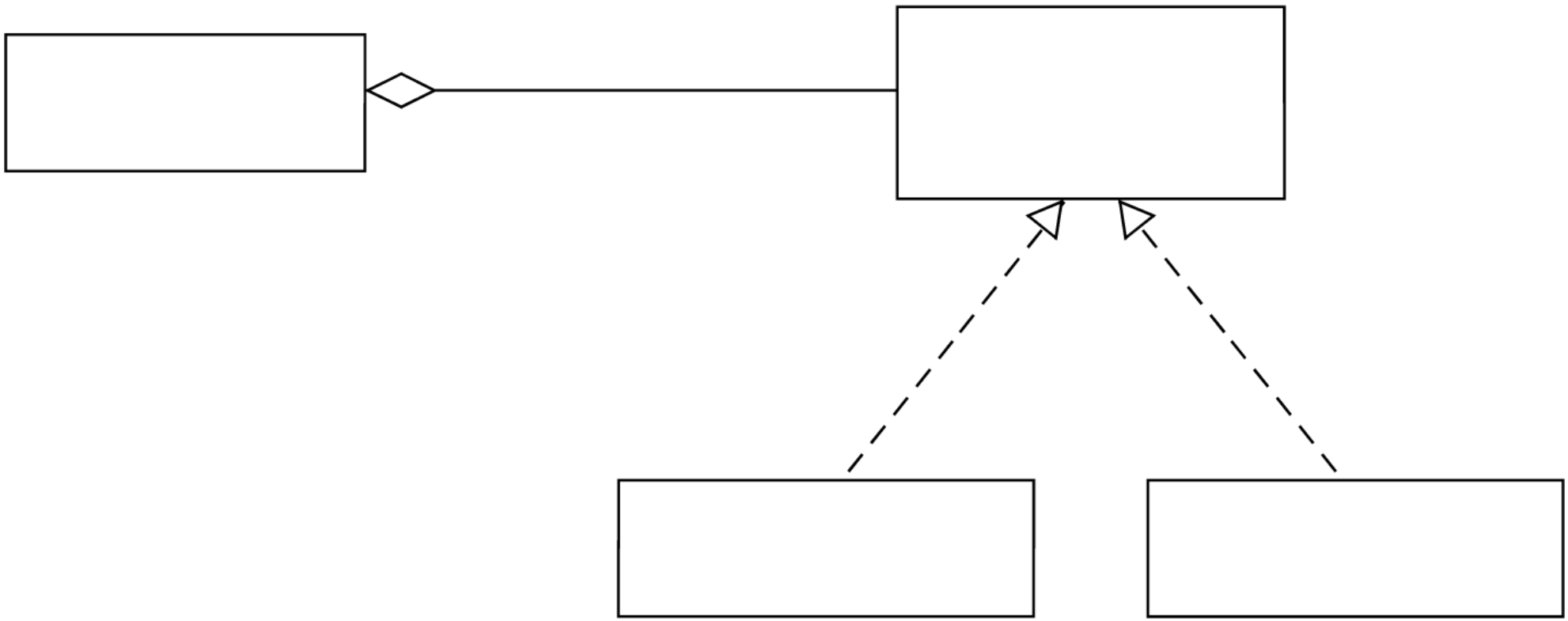
Where should we look for Model-View-Controller in a class diagram?

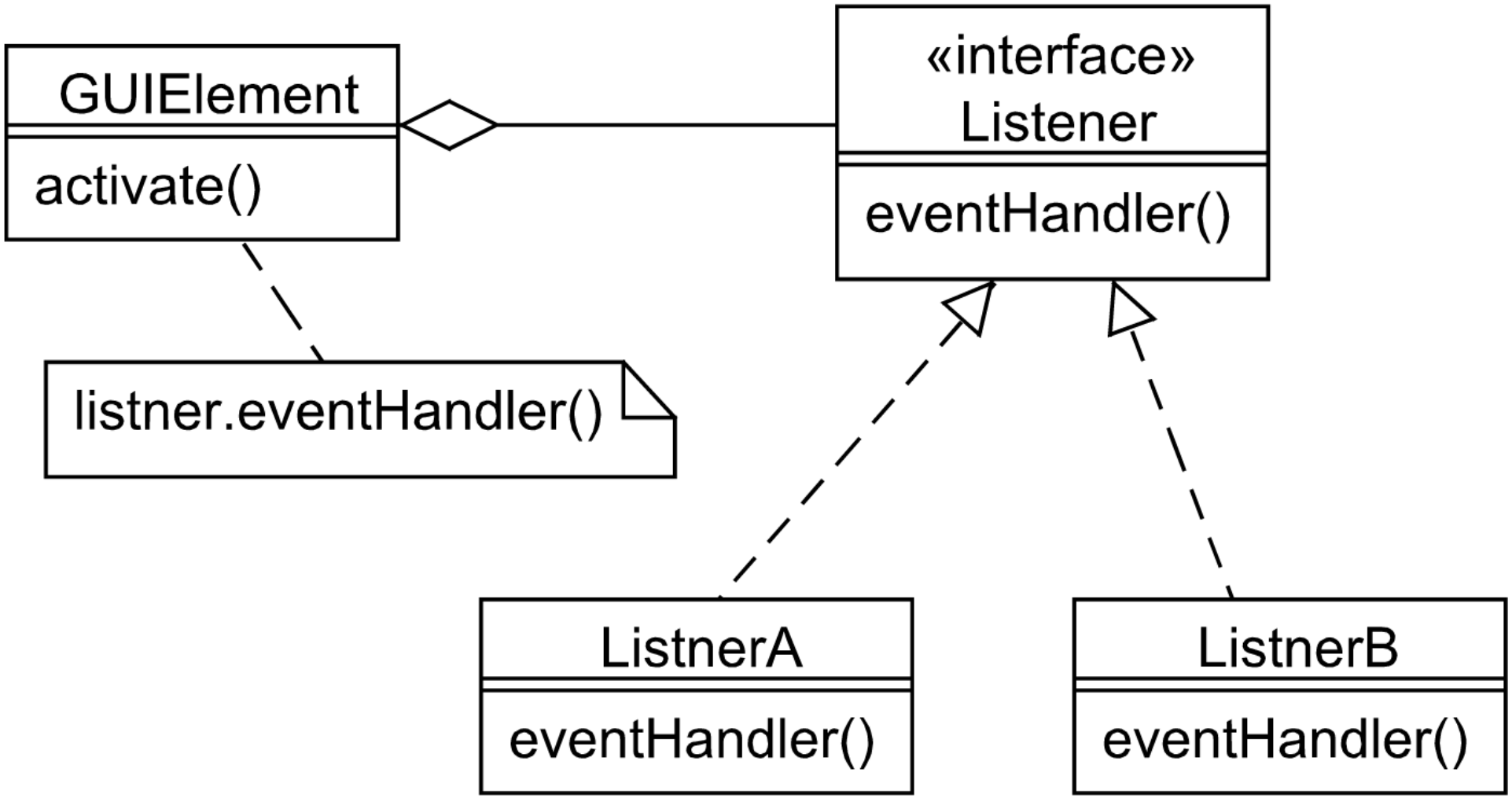


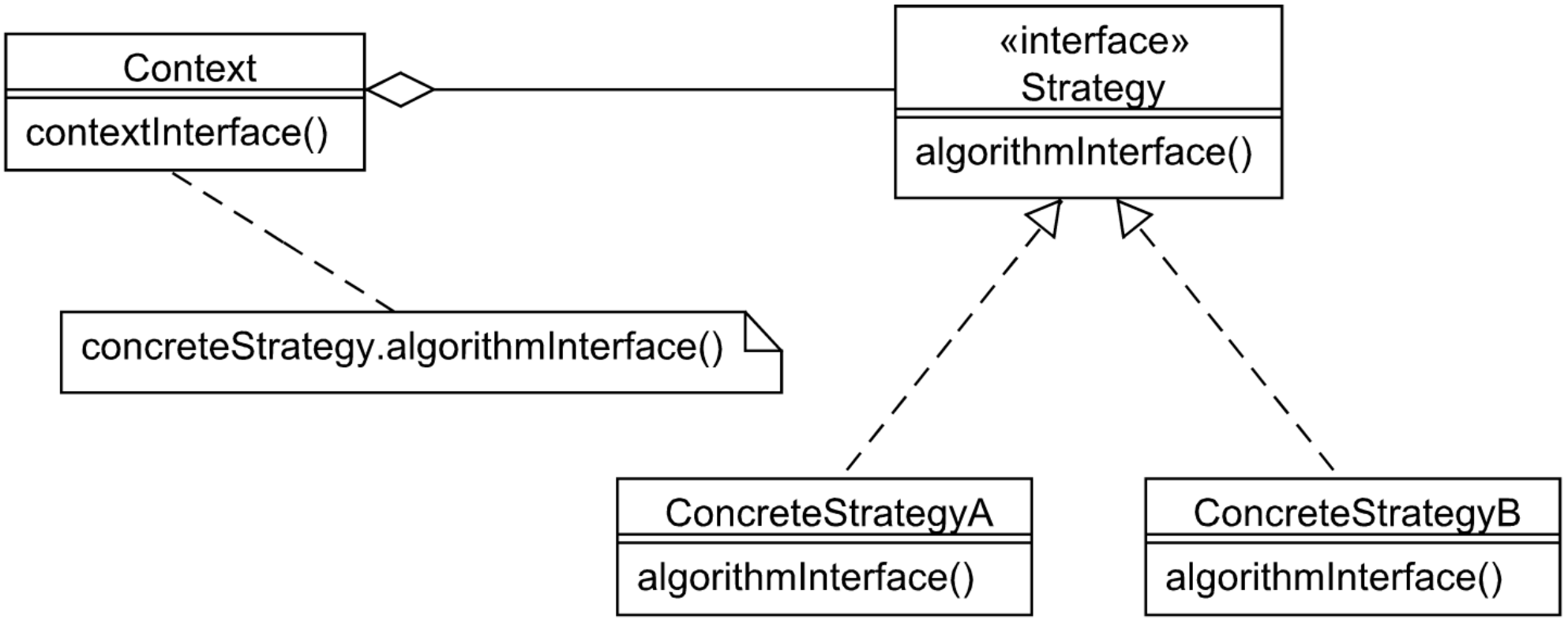

Client
«interface»
Component
operation()
*
Leaf
+operation()
Composite
+operation()
+add(component : Component)
+remove(component : Component)
+getChild(i : Integer)




«interface»
Subject
attach(observer: Observer)
detach(observer: Observer)
notify()
«interface»
Observer
update()
Subject1
state
+getState()
+setState()
Observer1
state
+update()




GUIElement
activate()
«interface»
Listener
eventHandler()
listner.eventHandler()
ListnerA
eventHandler()
ListnerB
eventHandler()


Context
contextInterface()
«interface»
Strategy
algorithmInterface()
concreteStrategy.algorithmInterface()
ConcreteStrategyA
algorithmInterface()
ConcreteStrategyB
algorithmInterface()

MVC ≈ Observer + Strategy + Composite

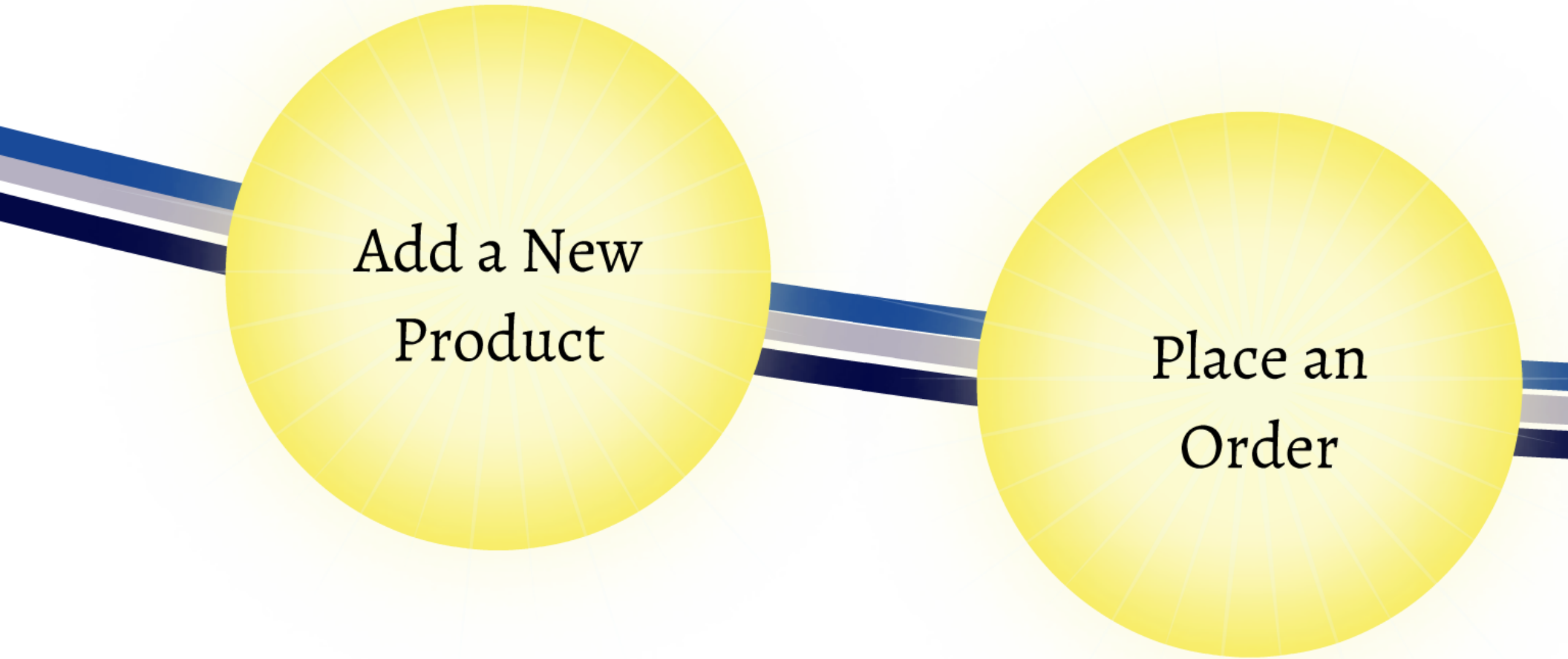
Observer: the Model–View relationship
(Model = Subject, View = Observer)

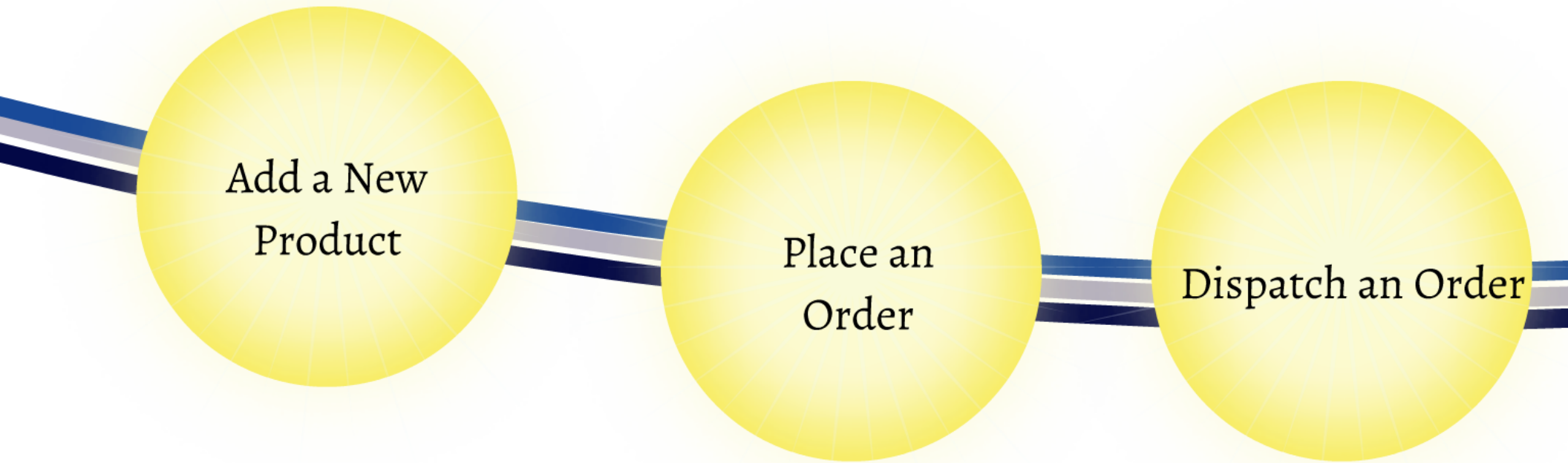
Strategy: the View–Controller relationship

Composite: nested views (View)

Unified Process class stereotypes:
a preparation for MVC,
but controller != control

Some decisions about the structure are generic (given by the realization domain)


Add a New Product
Place an Order


Add a New Product
Place an Order
Dispatch an Order


Add a New Product
Place an Order
Dispatch an Order
Adapt the Restock Plan
Architecture

Use cases

Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Conway's law

The software system structure copies the communication paths among the people who are developing this system

A part of the software system structure can be derived from use cases

A more stable part of the software system structure comes from the application domain

Software architecture

Some decisions about the structure are generic (given by the realization domain)

The software system structure copies the communication paths among the people who are developing this system

fiit.sk/~vranic/dot