Software design patterns

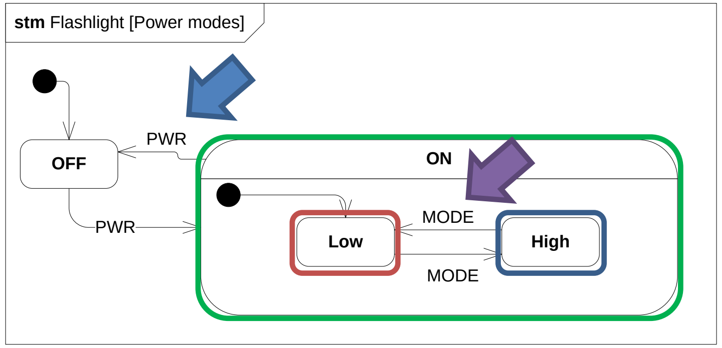
State Machines – Nested and Orthogonal States

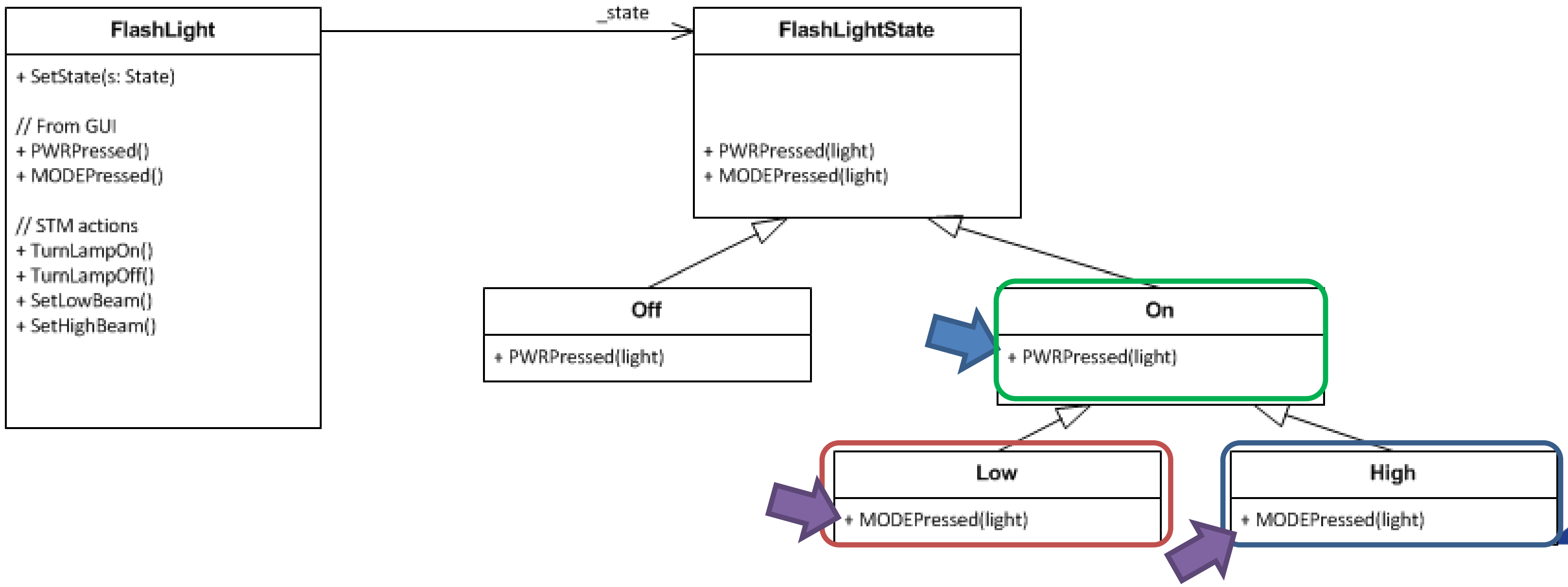
version: 1.0.3

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

GoF State: Nested states

- The GoF State Pattern is especially neat when used for complex (nested, orthogonal) state machines





FlashLight

+ SetState(s: State)

// From GUI
+ PWRPressed()
+ MODEPressed()

// STM actions
+ TurnLampOn()
+ TurnLampOff()
+ SetLowBeam()
+ SetHighBeam()

FlashLightState

+ PWRPressed(light)
+ MODEPressed(light)

Off

+ PWRPressed(light)

On

+ PWRPressed(light)

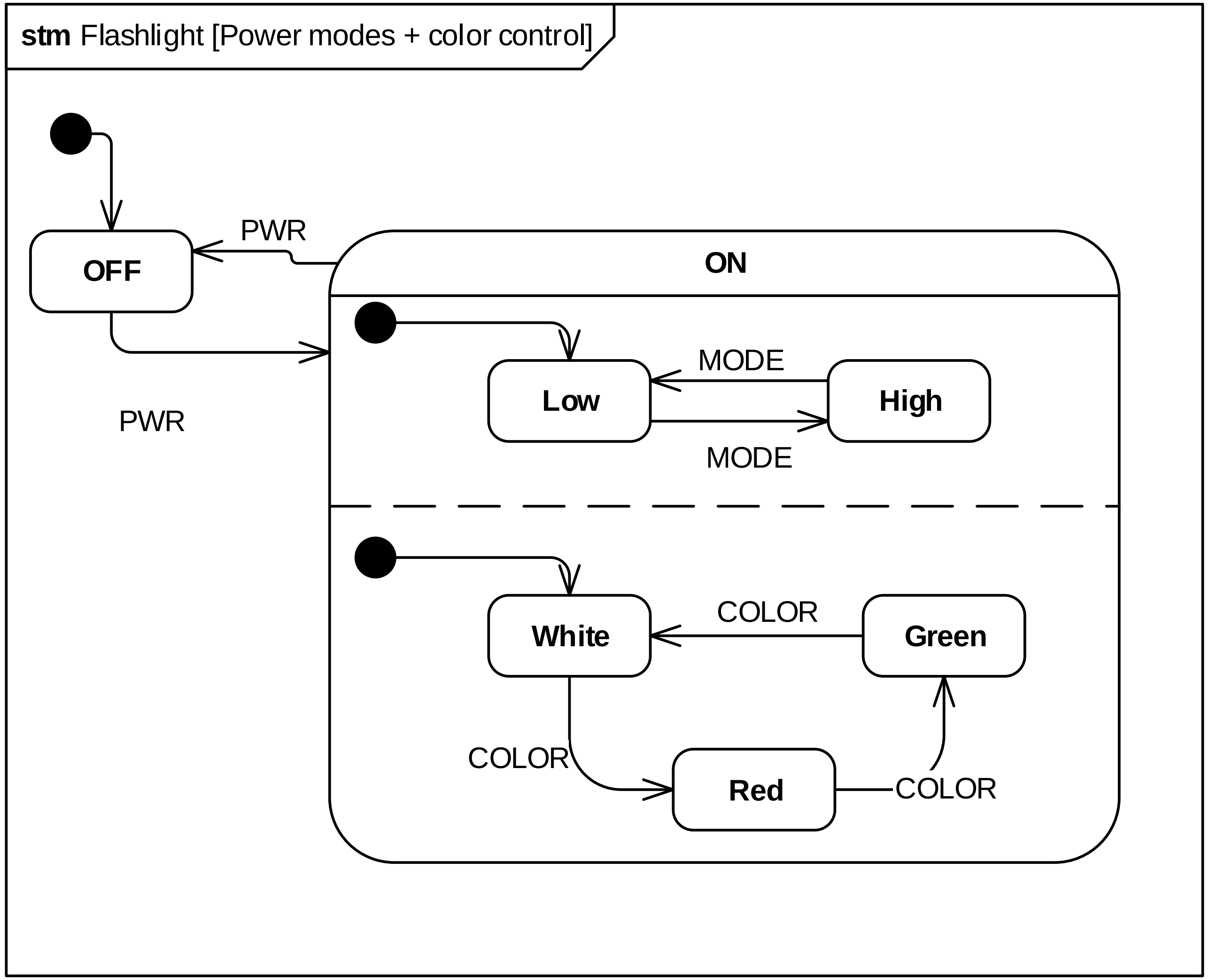
Low

+ MODEPressed(light)

High

+ MODEPressed(light)

GoF State: Orthogonal states

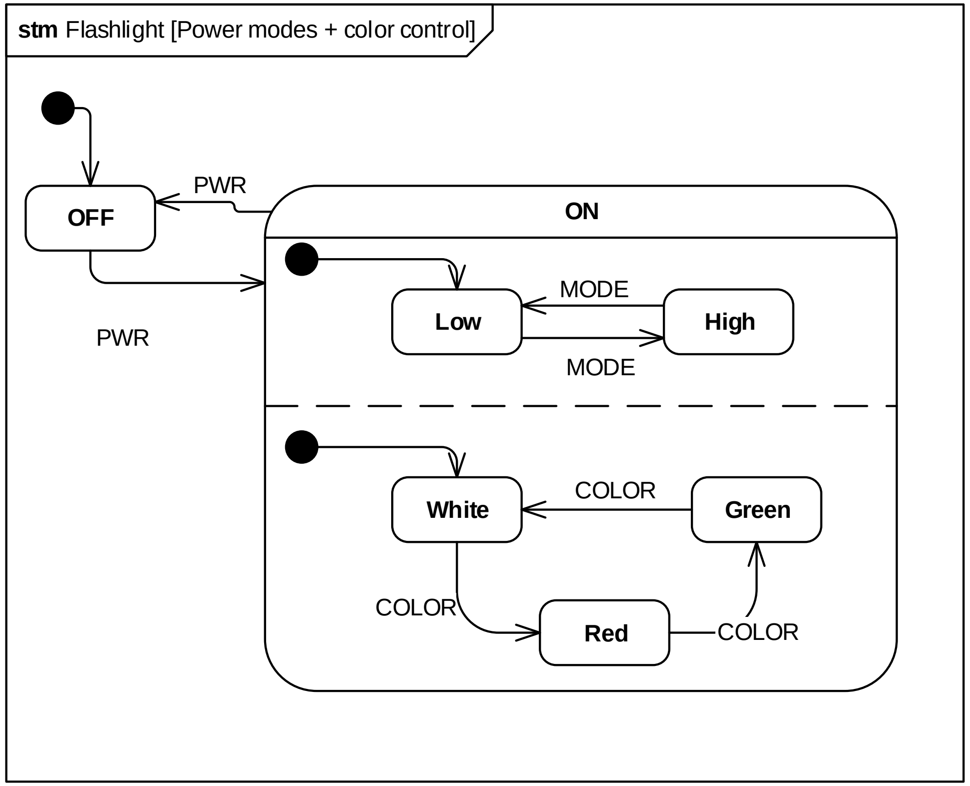


Orthogonal states

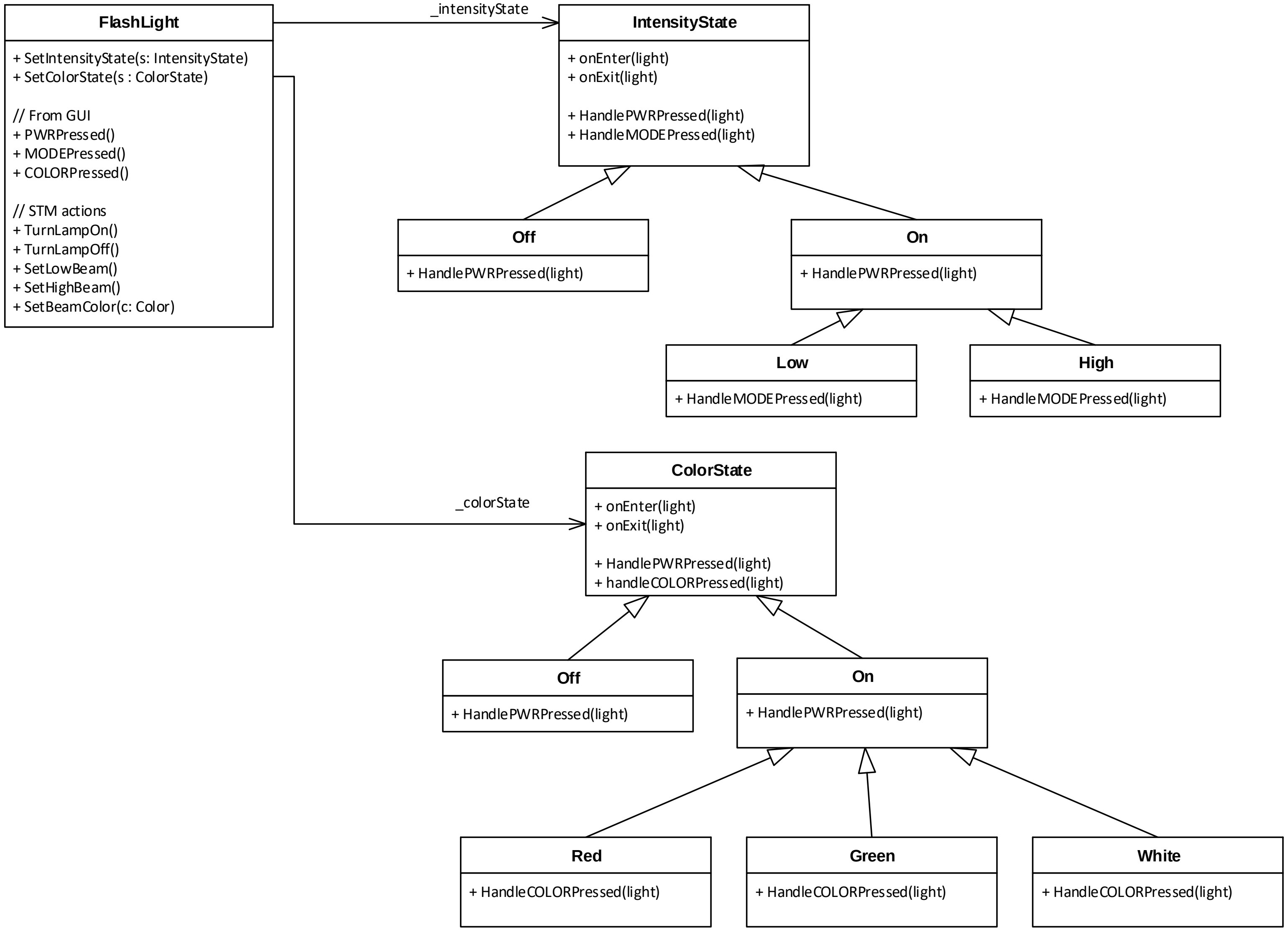
- Two implementation strategies:

  1) Collapse into a single state machine: Low-White, Low-Green, Low-Red, High-White, High-Green, High-Red.

  2) Create two separate state machines and let the context hold a reference to both.



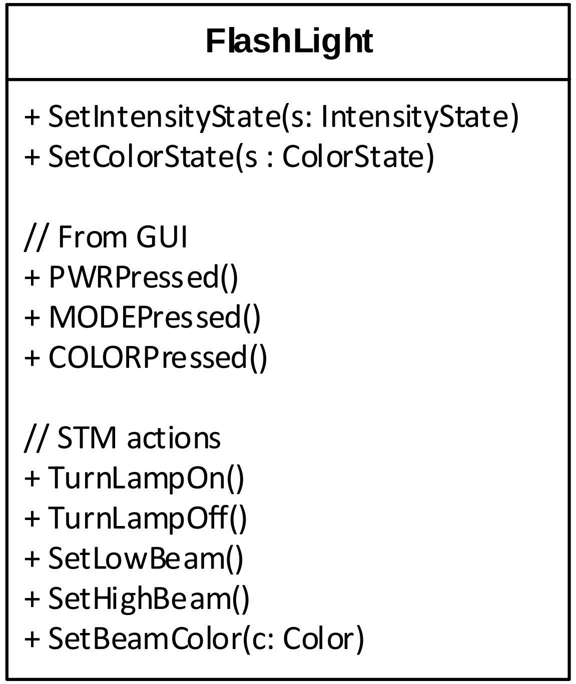
Orthogonal states impl.



FlashLight
+ SetIntensityState(s: IntensityState)
+ SetColorState(s : ColorState)
// From GUI
+ PWRPressed()
+ MODEPressed()
+ COLORPressed()
// STM actions
+ TurnLampOn()
+ TurnLampOff()
+ SetLowBeam()
+ SetHighBeam()
+ SetBeamColor(c: Color)
_intensityState
IntensityState
+ onEnter(light)
+ onExit(light)
+ HandlePWRPressed(light)
+ HandleMODEPressed(light)
Off
+ HandlePWRPressed(light)
On
+ HandlePWRPressed(light)
Low
+ HandleMODEPressed(light)
High
+ HandleMODEPressed(light)
_colorState
ColorState
+ onEnter(light)
+ onExit(light)
+ HandlePWRPressed(light)
+ handleCOLORPressed(light)
Off
+ HandlePWRPressed(light)
On
+ HandlePWRPressed(light)
Red
+ HandleCOLORPressed(light)
Green
+ HandleCOLORPressed(light)
White
+ HandleCOLORPressed(light)

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

The evil client



FlashLight

+ SetIntensityState(s: IntensityState)
+ SetColorState(s : ColorState)

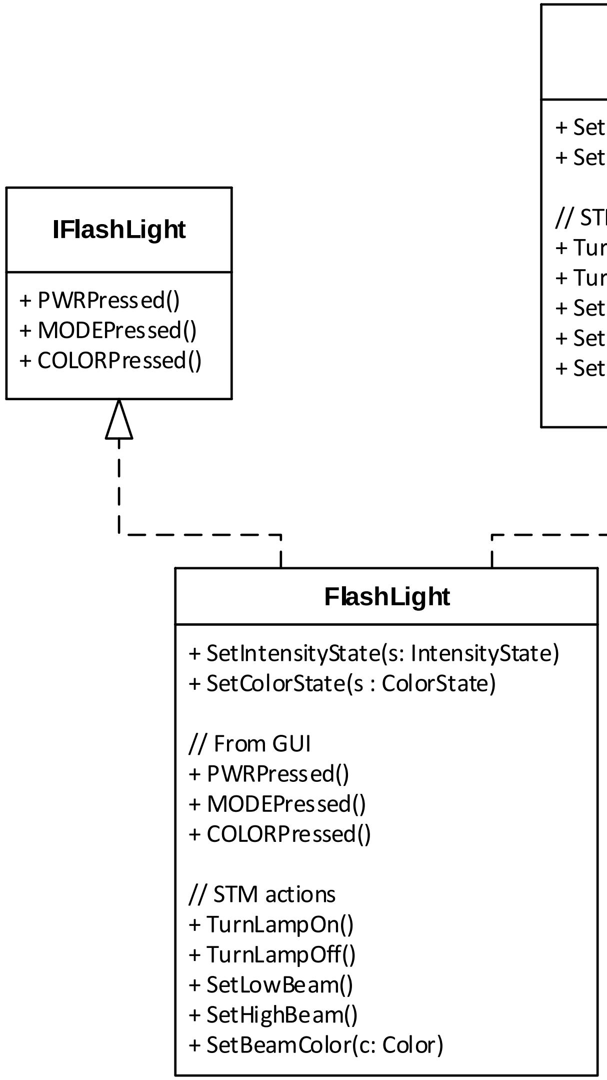
// From GUI
+ PWRPressed()
+ MODEPressed()
+ COLORPressed()

// STM actions
+ TurnLampOn()
+ TurnLampOff()
+ SetLowBeam()
+ SetHighBeam()
+ SetBeamColor(c: Color)

What if a client to the context writes code which sets the state directly?

Or turns on/off the lamp, and thus bypasses the statemachine?

ISP to the rescue



IFlashLight

+ PWRPressed()
+ MODEPressed()
+ COLORPressed()

IFlashLight
Internal

+ SetIntensityState(s: IntensityState)
+ SetColorState(s : ColorState)

// STM actions
+ TurnLampOn()
+ TurnLampOff()
+ SetLowBeam()
+ SetHighBeam()
+ SetBeamColor(c: Color)

FlashLight

+ SetIntensityState(s: IntensityState)
+ SetColorState(s : ColorState)

// From GUI
+ PWRPressed()
+ MODEPressed()
+ COLORPressed()

// STM actions
+ TurnLampOn()
+ TurnLampOff()
+ SetLowBeam()
+ SetHighBeam()
+ SetBeamColor(c: Color)

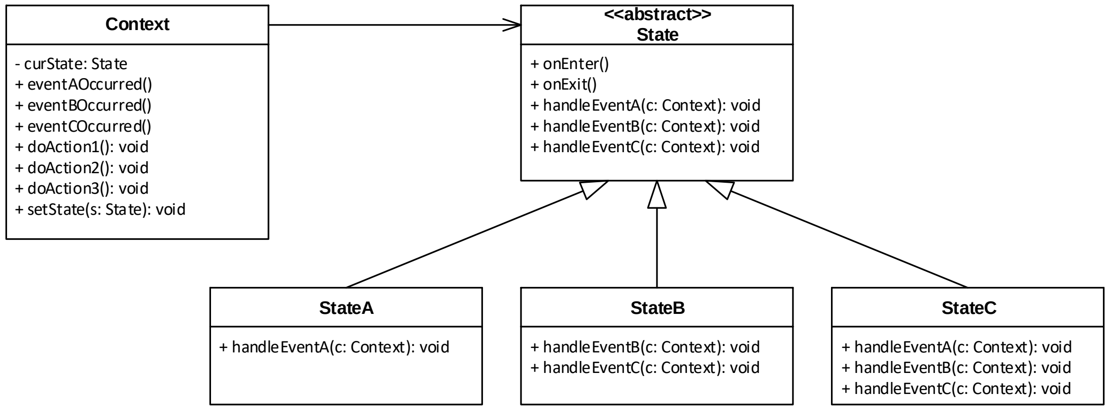
Segregate the interface to the context.

One interface to be used by clients of the context.

Another interface to be used by the state machine implementation

State vs. Strategy

The structure of GoF State and GoF Strategy are quite similar.



Context

- curState: State
+ eventAOccurred()
+ eventBOccurred()
+ eventCOccurred()
+ doAction1(): void
+ doAction2(): void
+ doAction3(): void
+ setState(s: State): void

<<abstract>>
State

+ onEnter()
+ onExit()
+ handleEventA(c: Context): void
+ handleEventB(c: Context): void
+ handleEventC(c: Context): void

StateA

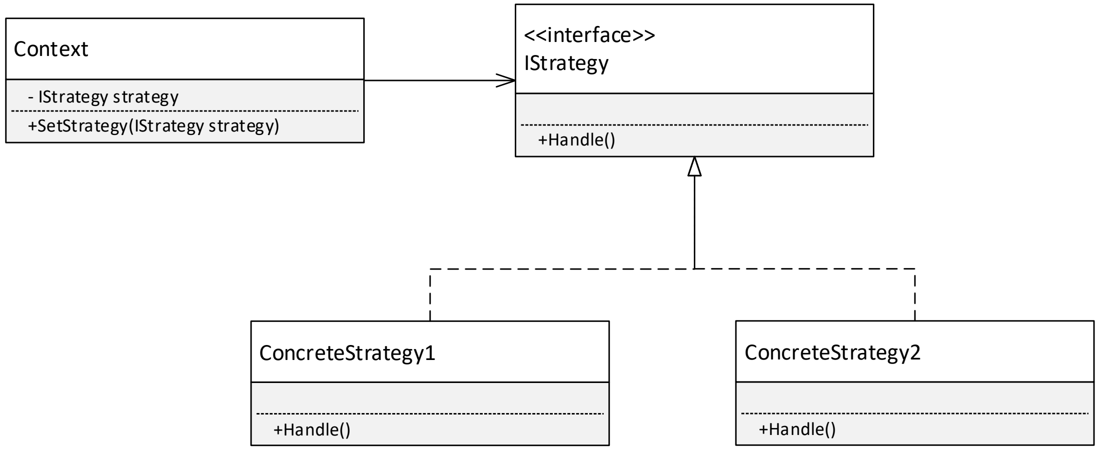
+ handleEventA(c: Context): void

StateB

+ handleEventB(c: Context): void
+ handleEventC(c: Context): void

StateC

+ handleEventA(c: Context): void
+ handleEventB(c: Context): void
+ handleEventC(c: Context): void



Context

- IStrategy strategy
+SetStrategy(IStrategy strategy)

<<interface>>
IStrategy

+Handle()

ConcreteStrategy1

+Handle()

ConcreteStrategy2

+Handle()

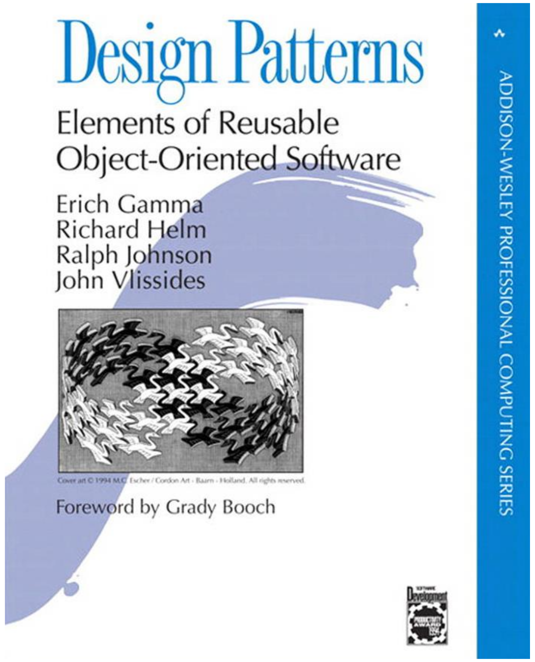
State vs. Strategy

GoF State

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

GoF Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

State vs. Strategy

GoF State

Used to model a system, which has states.

A client is not supposed to modify the state – the transition between states is done by the states.

GoF Strategy

Used to change behavior of a part of a program i.e. change which algorithm to use.

A client is allowed to change the algorithm.

State vs. Strategy

Summary: Each pattern uses a polymorphic call to do something depending on the context.

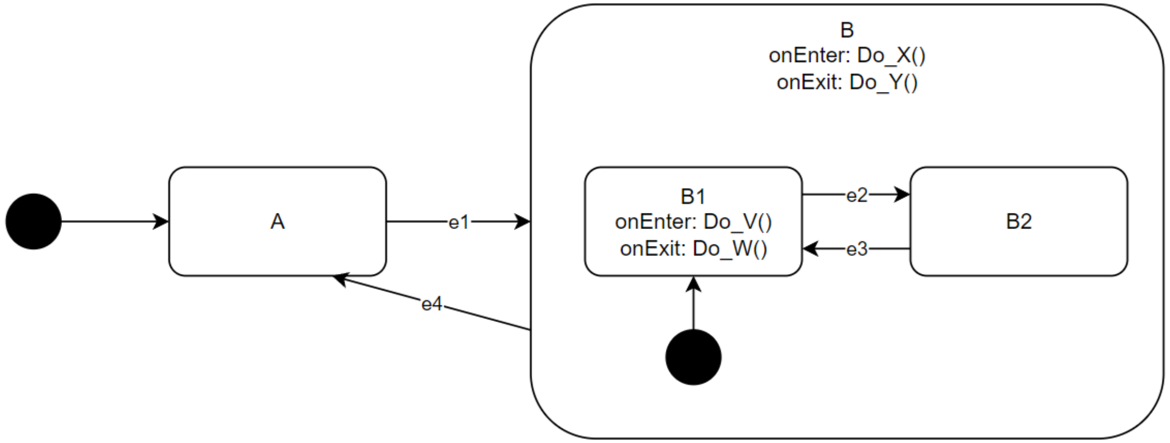
In the State pattern, the polymorphic call **often** causes a change to another *state* – and therefore its behavior.

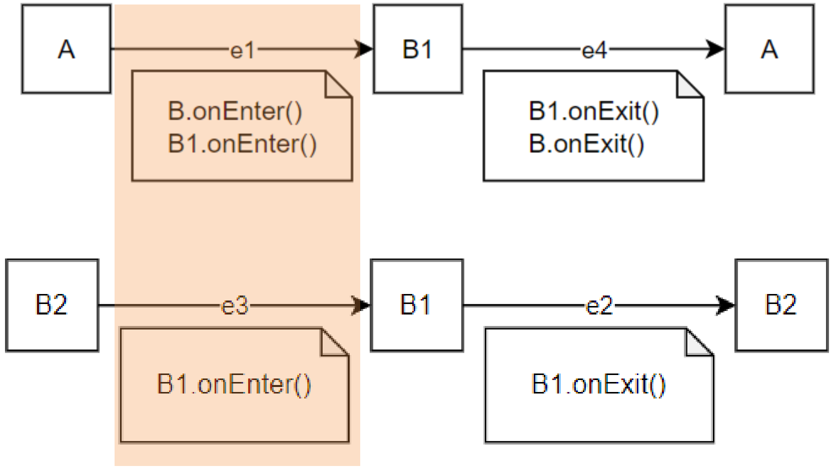
In the Strategy pattern, the polymorphic call does **not** typically change the context's behavior.

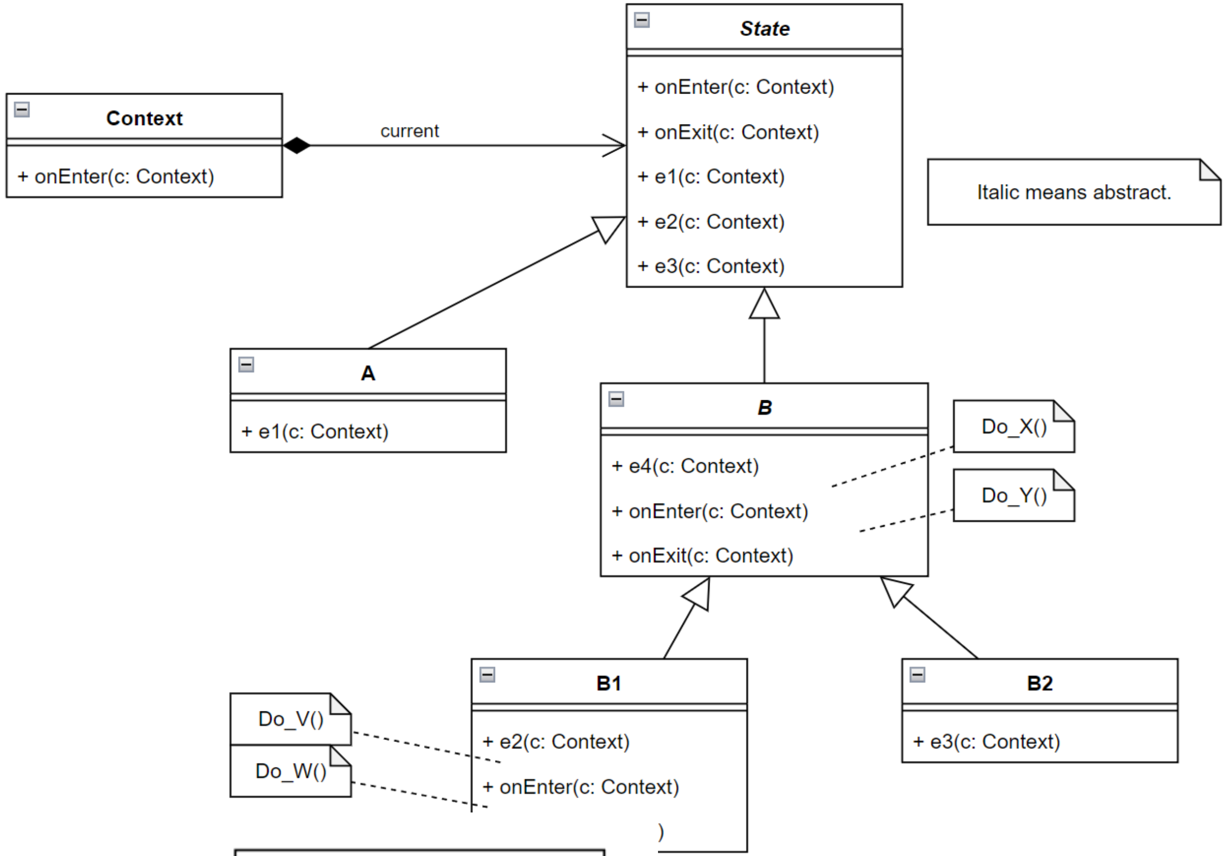
Again, the State pattern's dynamics are determined by its corresponding *finite state machine,* which is essential to correct application of this pattern.

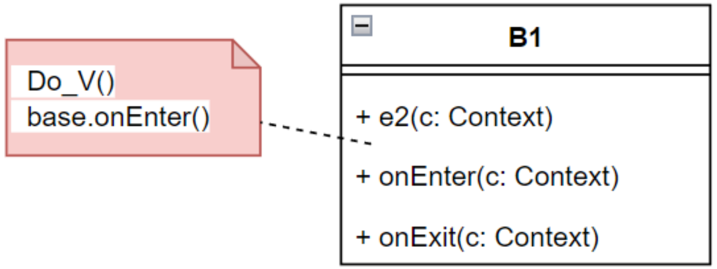
User Fuhrmanator @ https://stackoverflow.com/questions/1658192/what-is-the-difference-between-strategy-design-pattern-and-state-design-pattern

Warning: State Pattern Limitations









Source Notes

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Your turn!

- State patterns exercise 1 (intro), question 5-7
- State pattern exercise 2 (phone)