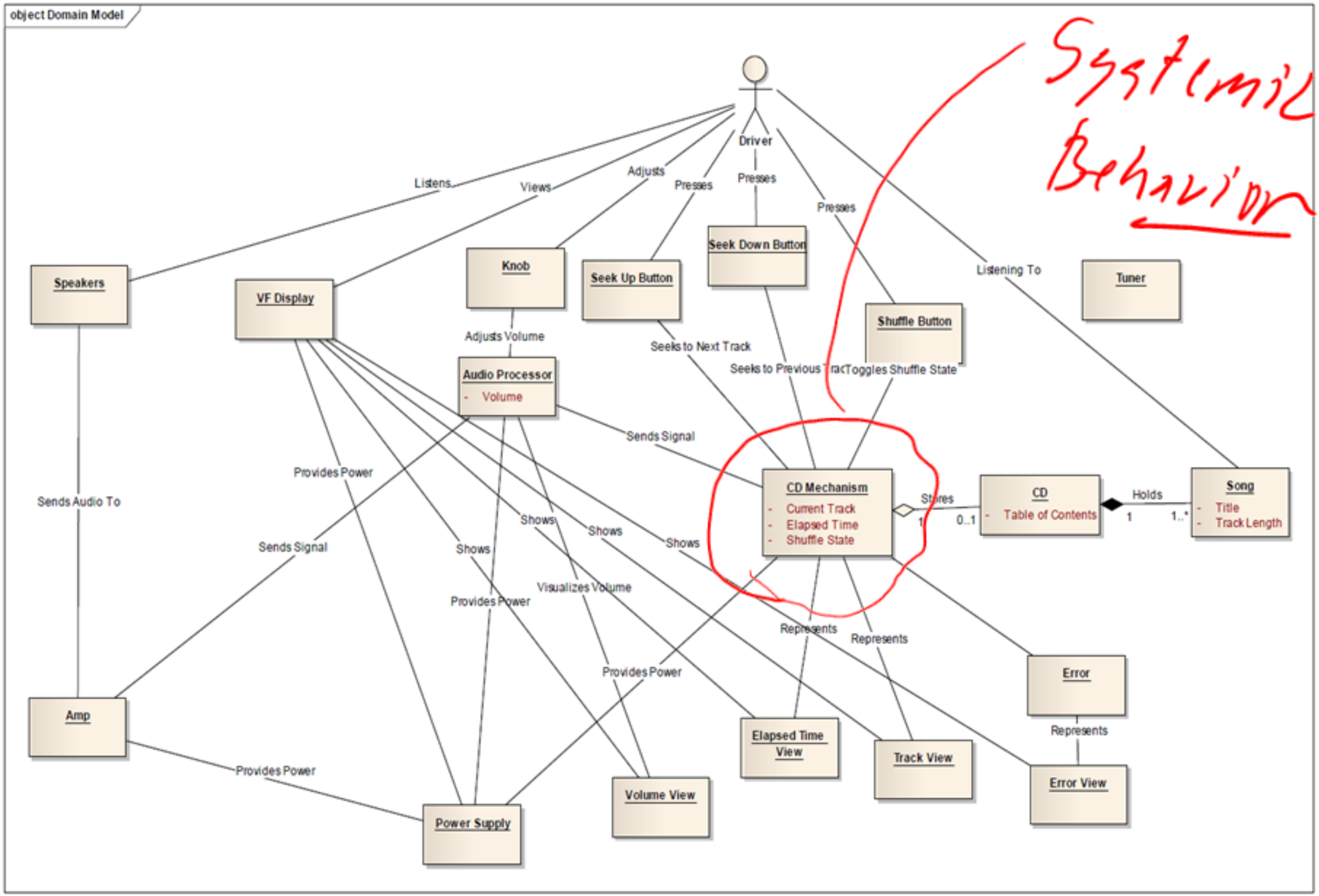
Defining Object Behavior

Objectives

- Explain the relationship between simple, state, and continuous behaviors.
- Define entry actions, exit actions, and activities.
- Explain the transaction syntax for transitions.
- Define guard condition.
- Define object state behavior using a UML state charts
- Represent concurrent state machine behavior using Harel state machines.

No lab

tomorrow?


object Domain Model
Driver
Listens
Views
Adjusts
Presses
Presses
Presses
Speakers
VF Display
Knob
Seek Up Button
Seek Down Button
Shuffle Button
Listening To
Tuner
Adjusts Volume
Audio Processor
- Volume
Seeks to Next Track
Seeks to Previous Trac
Toggles Shuffle State
Sends Signal
Provides Power
Sends Audio To
Sends Signal
Shows
Shows
Shows
Shows
Provides Power
Visualizes Volume
CD Mechanism
- Current Track
- Elapsed Time
- Shuffle State
Stores
1
0..1
CD
- Table of Contents
Holds
1
1..*
Song
- Title
- Track Length
Represents
Represents
Error
Represents
Provides Power
Amp
Provides Power
Power Supply
Volume View
Elapsed Time View
Track View
Error View

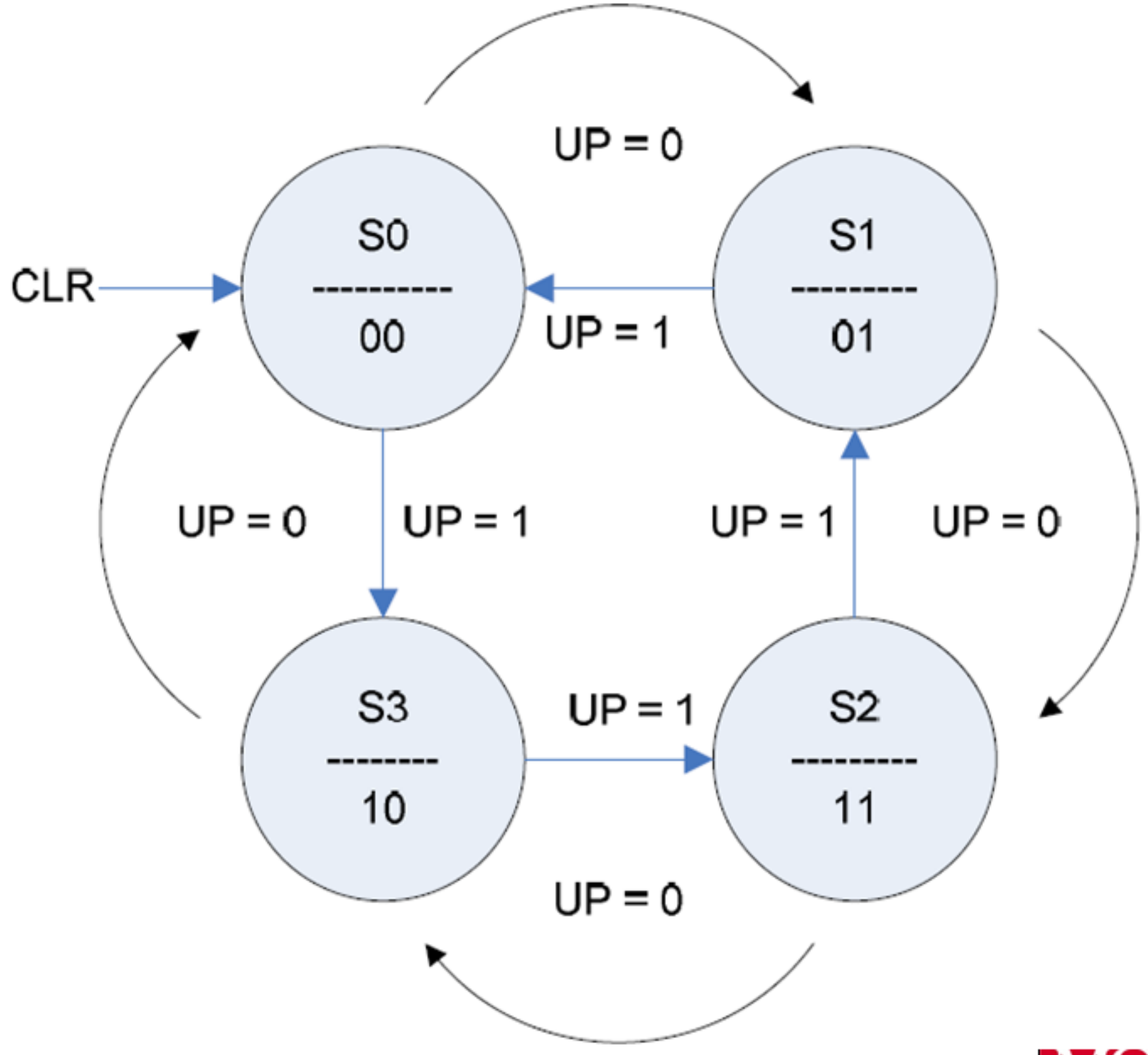
Homework

- For Wednesday in Lab:
- Pg 472, Exercise 1
- Pg 472 Exercise 3
- Pg 472 Exercise 4
- Pg 472, Exercise 6
- Pg 472, Exercise 9
- Pg 472, Exercise 10

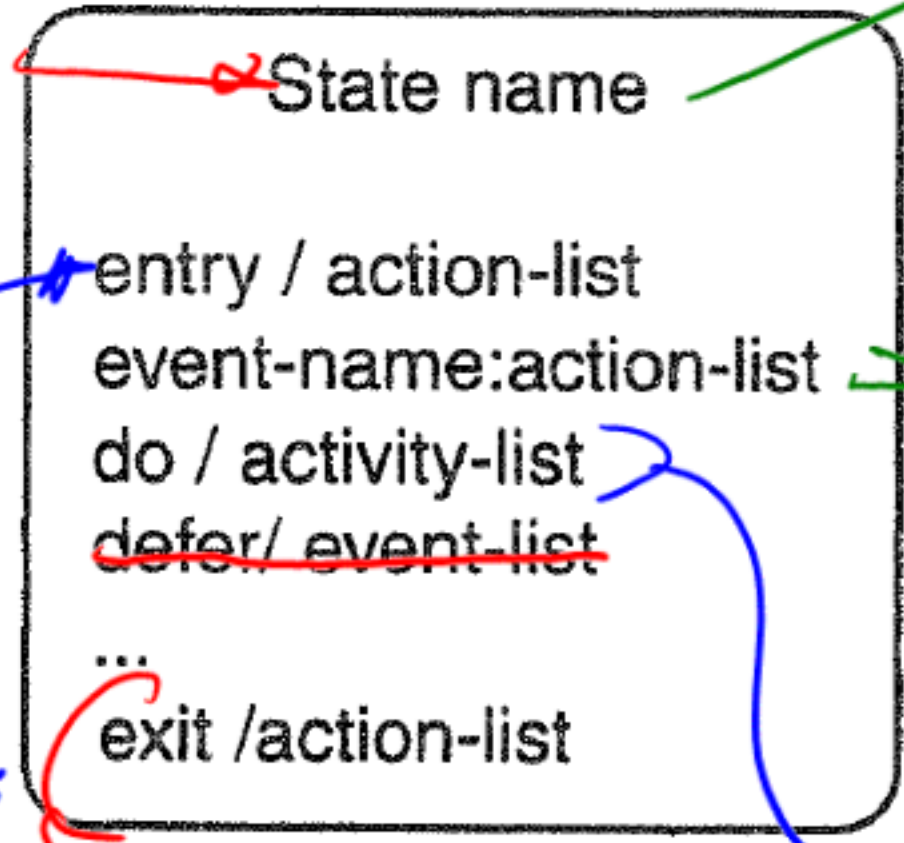
No Homework

- Simple Behavior
  - No memory of previous services
  - Has no concept of history
  - Examples: Mathematical functions, sort operations, etc.
  - Typically use an activity diagram to represent them
- State Behavior
  - To be discussed shortly
- Continuous Behavior
  - The current output depends on the previous history in a "smooth way"
  - PID controller, Fuzzy logic, Neural Networks
  - UML does not provide a mechanism for expressing this kind of behavior

State Machines
Quartus Style



State Chart

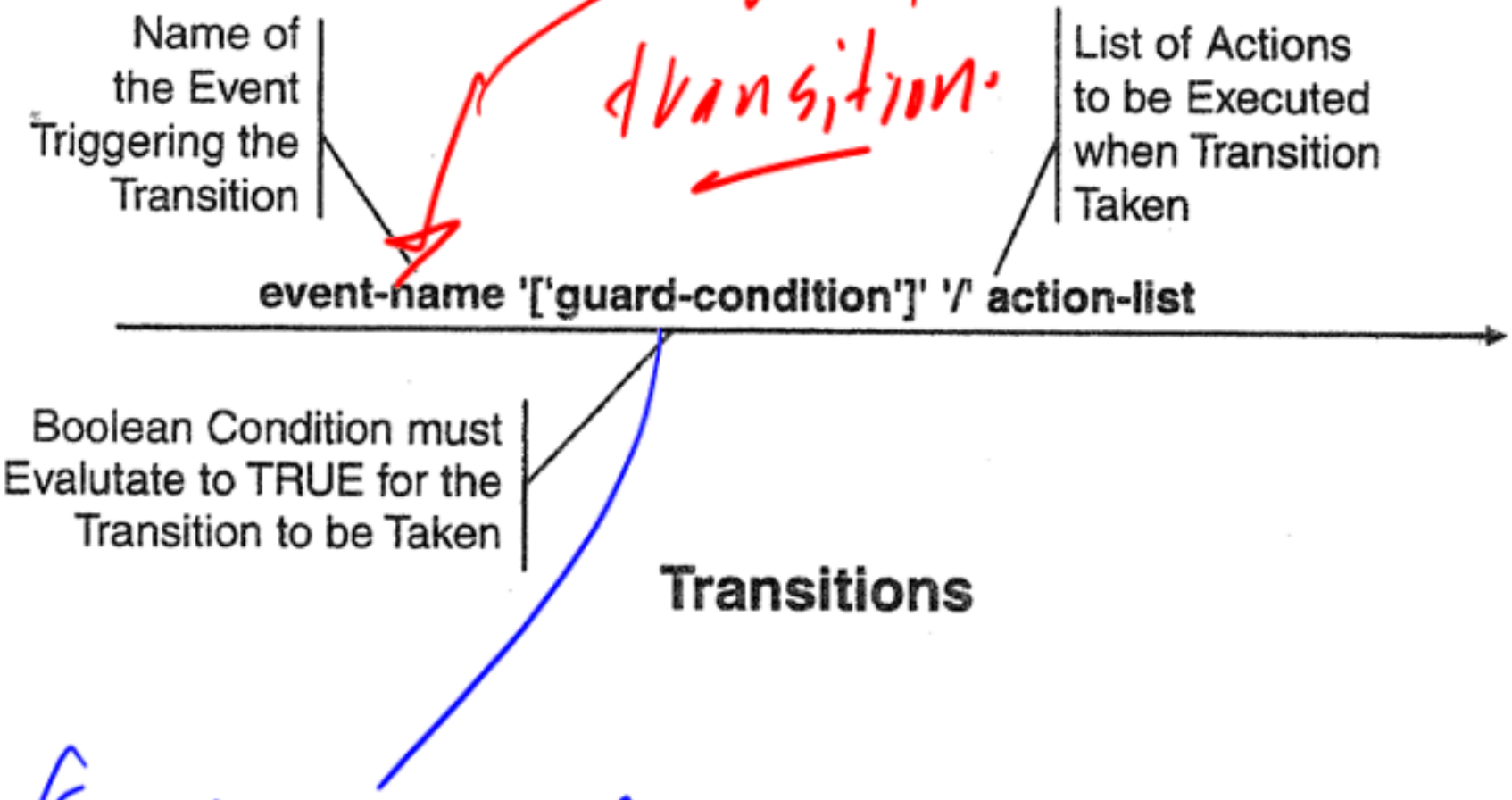


State Icon

State Transitions

Going from one state to another.

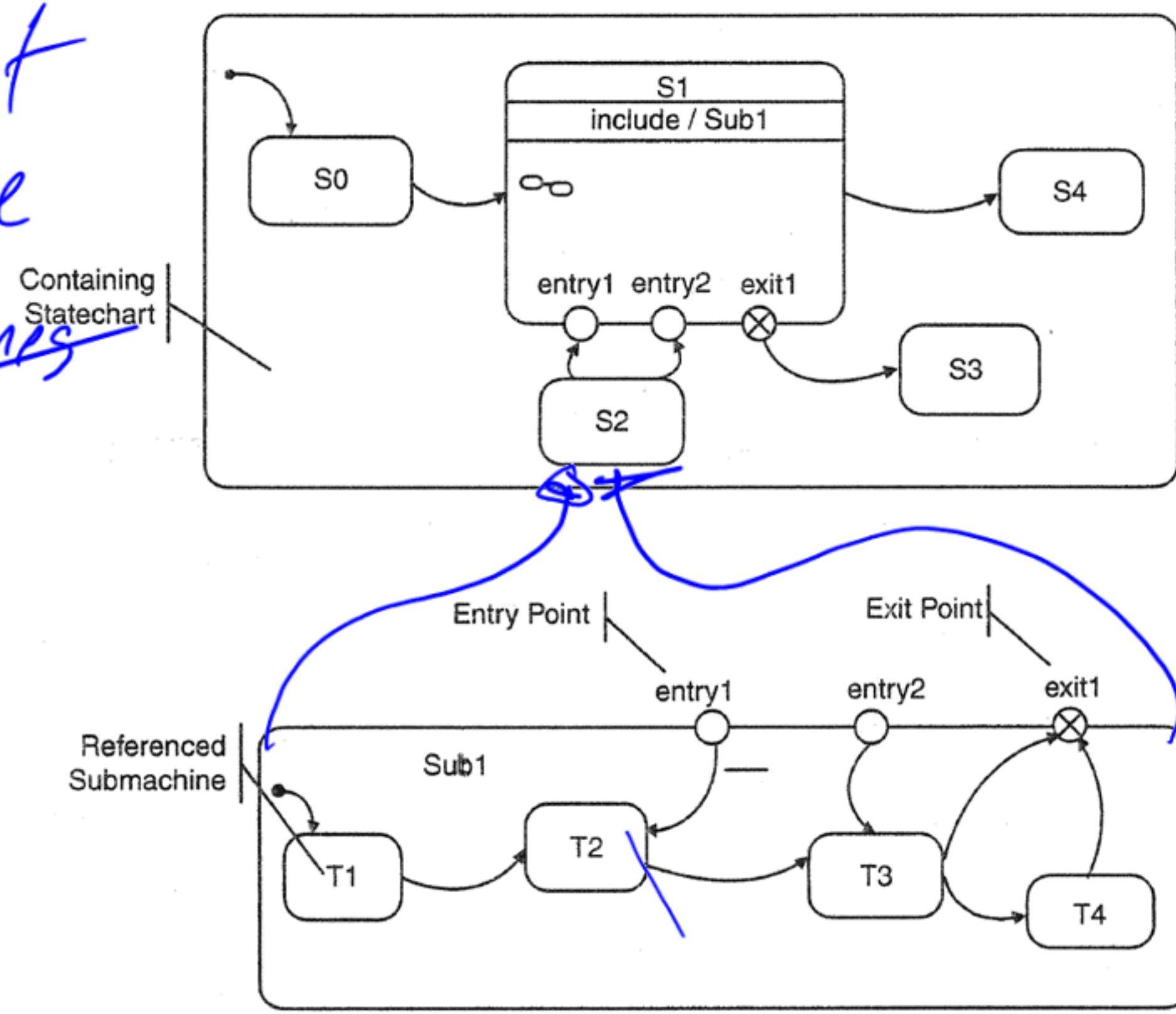
What causes the transition.



(Extra condition, if.)

State Charts

Nest State Machines



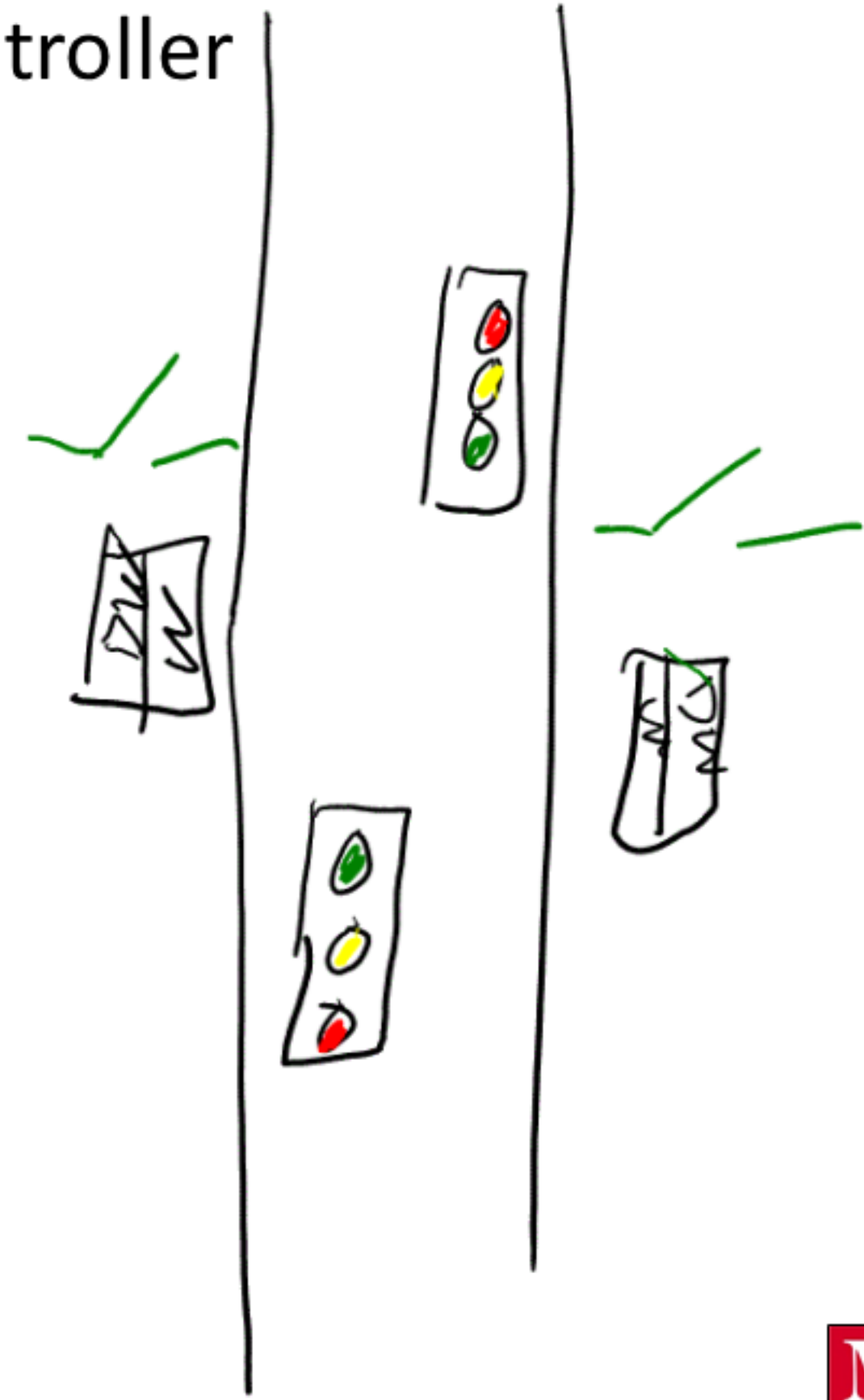
Our Example

- Crosswalk Controller

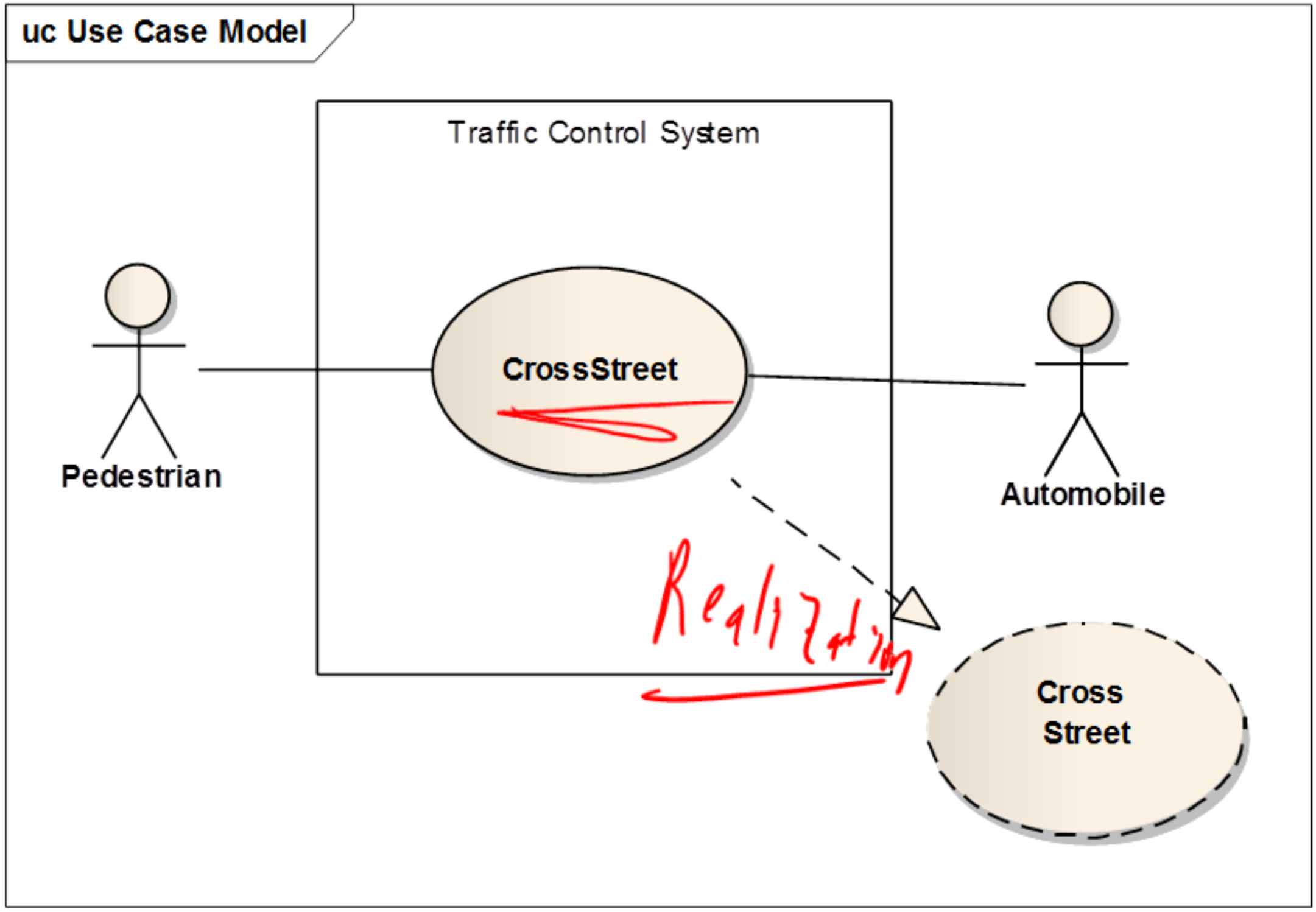
push button

Lights

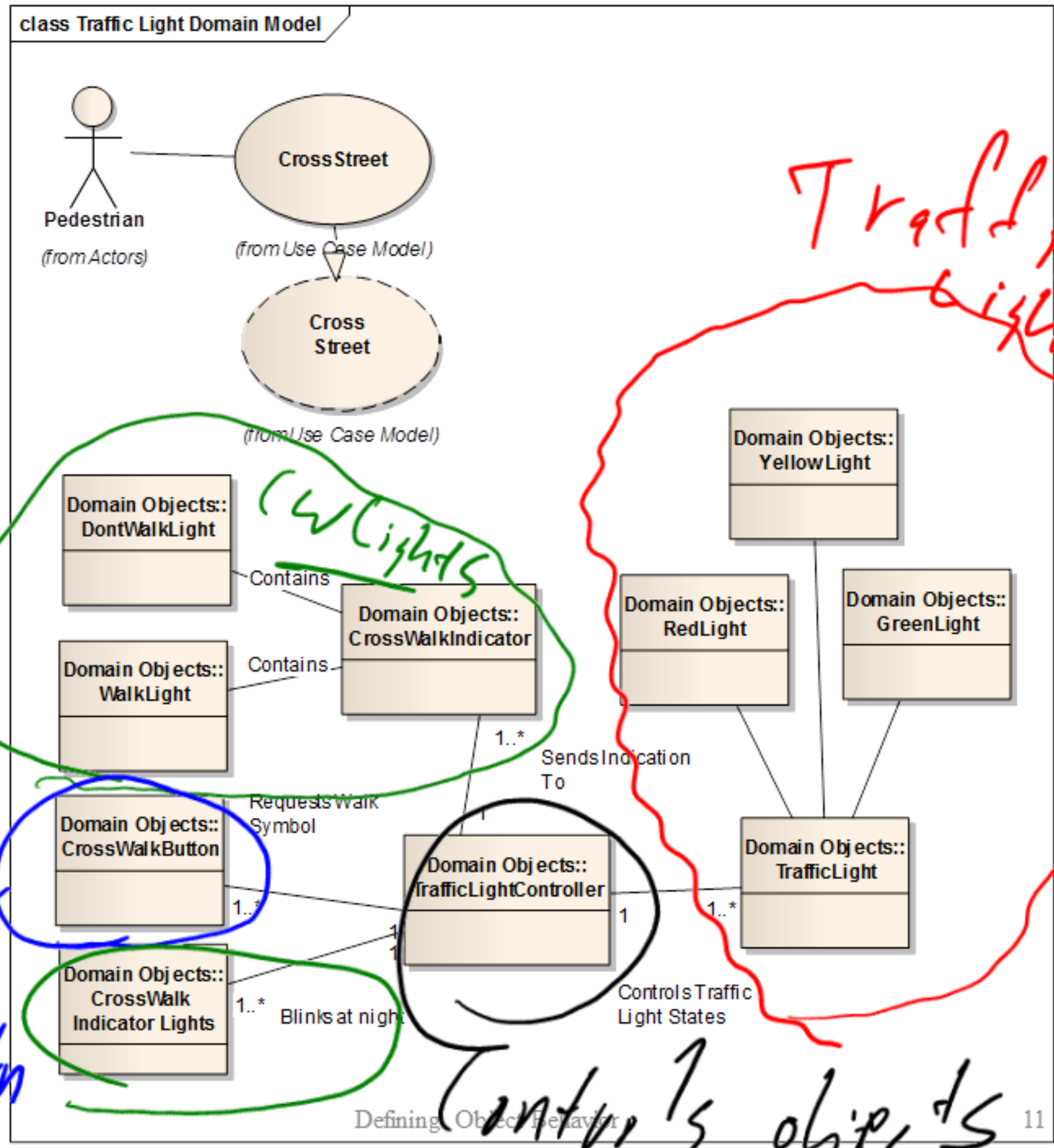
CW signs ...



Example Use Case Model



Crosswalk Controller Domain Model



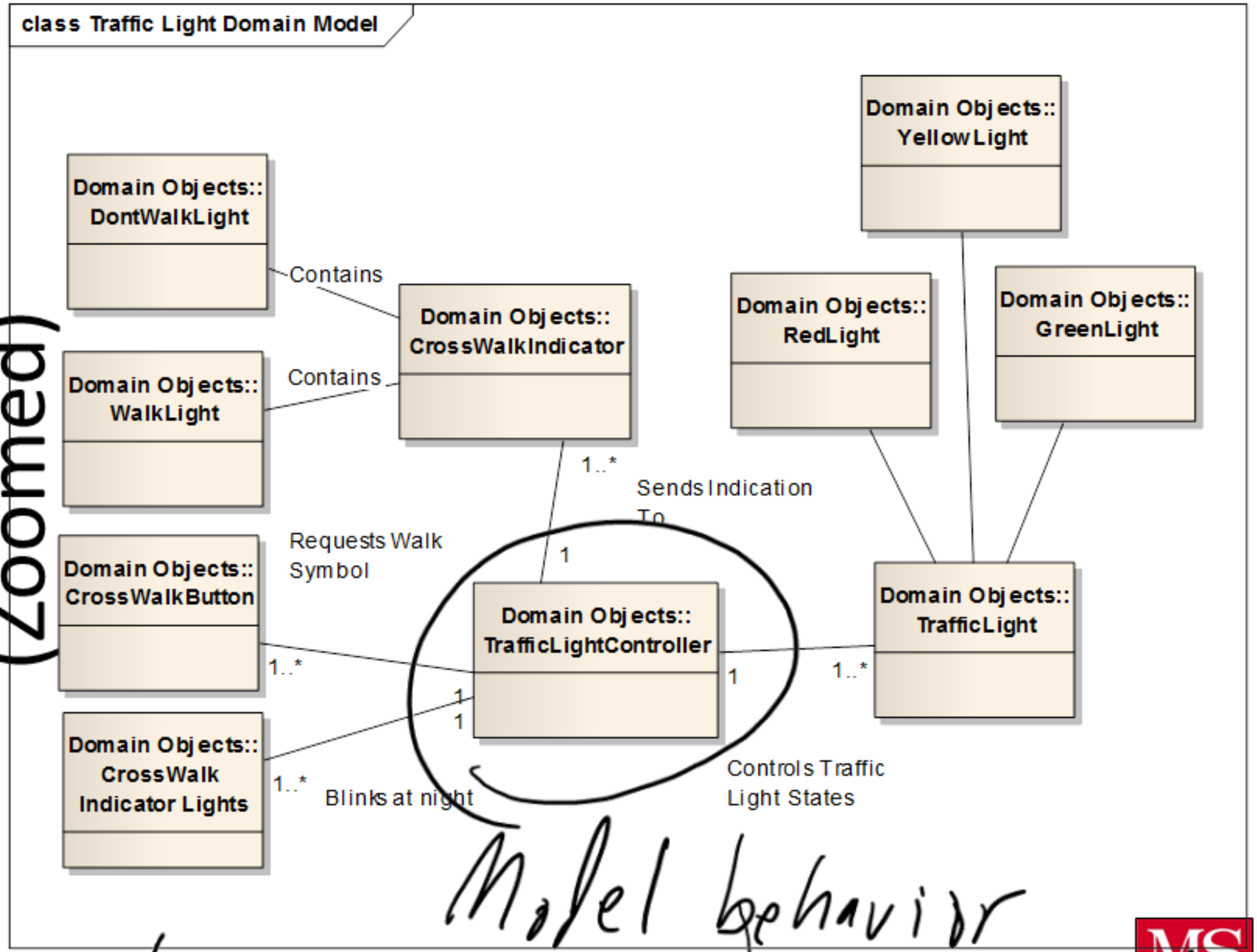
Scenario

- Preconditions
  - Traffic light system is not blinking yellow
- Detailed Scenario
  - User has a desire to cross the street
  - User presses the cross street request button
  - Traffic light changes from green to yellow to red.
  - Crosswalk sign changes to walk

Use Case

- Pedestrian crosses street
- Crosswalk sign begins blinking don't walk
- Crosswalk sign changes to don't walk
- Traffic light changes to green

Cross Walk Domain Model (Zoomed)



Model behavior of this class as a state machine.

stm Design Model

(from CrossWalkController)

DW_FLASH_TIMER_EXPIRED

DontWalk

+ exit / darkenDontWalkLight
+ entry / illuminateDontWalkLight
+ entry / SAFE_FOR_TRAFFIC_TO_DRIVE

Idle

(from CrossWalkController)

CROSS_BUTTON_PRESSED

CrossRequested

+ entry / STOP_TRAFFIC_REQUEST

(from CrossWalkController)

(from CrossWalkController)

Flashing Dont Walk

+ entry / startDontWalkFlashTimer

(from CrossWalkController)

DW_ON

+ entry / illuminateDontWalkLight
+ entry / startFlashTimer
+ exit / darkenDontWalkLight

(from CrossWalkController)

FLASH_TIMER_EXPIRED

FLASH_TIMER_EXPIRED

DW_OFF

+ entry / startFlashTimer

(from CrossWalkController)

SAFE_TO_CROSS

CROSSING_TIMER_EXPIRED

Walk

+ entry / startCrossingTimer
+ exit / turnOffWalkLight
+ entry / turnOnWalkLight

(from CrossWalkController)

State

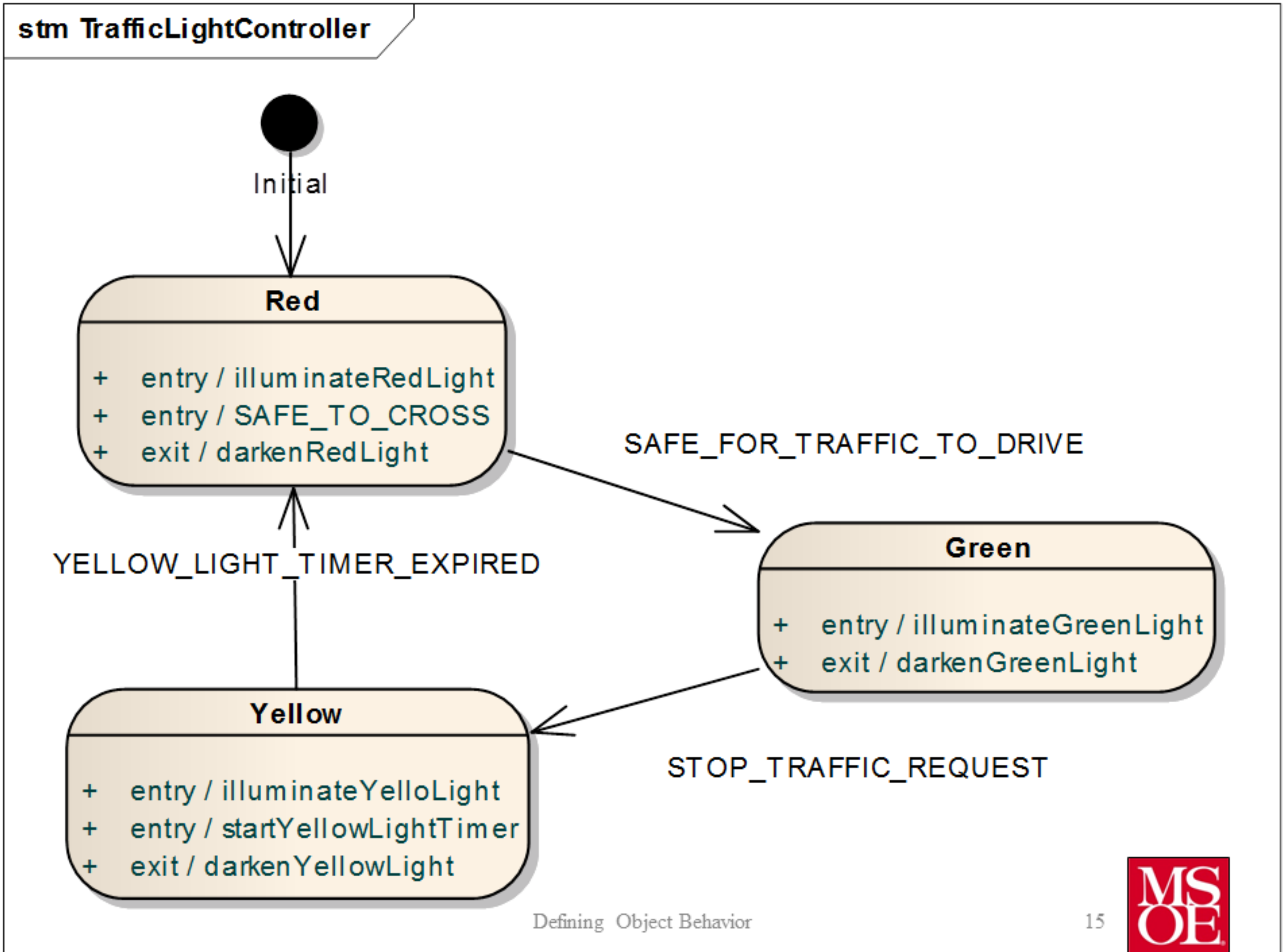
Entry/

Exit

Actions

Detailed.

(from CrossWalkController)


stm TrafficLightController
Initial
Red
+ entry / illuminateRedLight
+ entry / SAFE_TO_CROSS
+ exit / darkenRedLight
SAFE_FOR_TRAFFIC_TO_DRIVE
Green
+ entry / illuminateGreenLight
+ exit / darkenGreenLight
YELLOW_LIGHT_TIMER_EXPIRED
Yellow
+ entry / illuminateYelloLight
+ entry / startYellowLightTimer
+ exit / darkenYellowLight
STOP_TRAFFIC_REQUEST

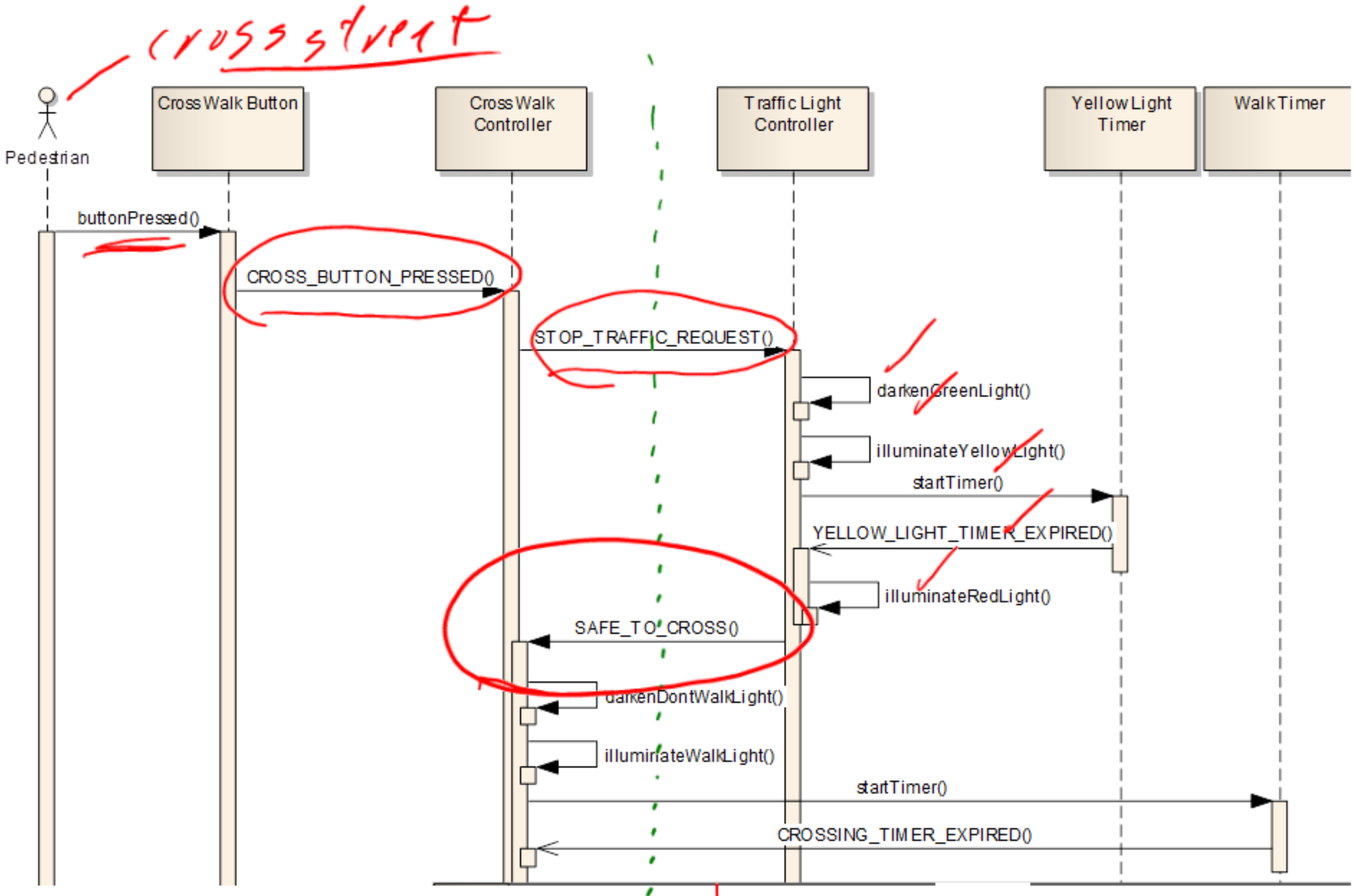
Problems with State Diagrams

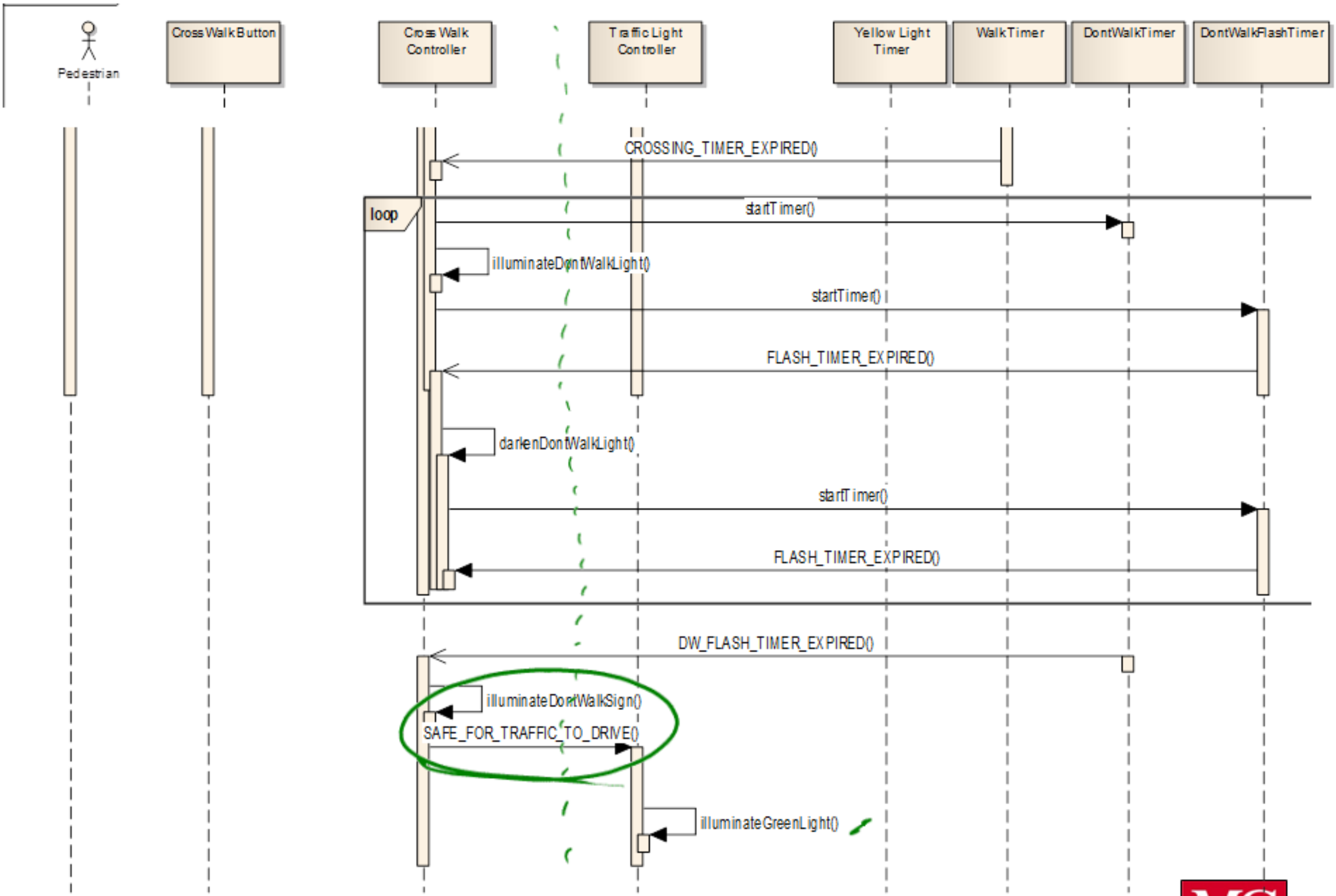
- State diagrams are ineffective at showing how items collaborate
- Incapable of showing typical paths through the states

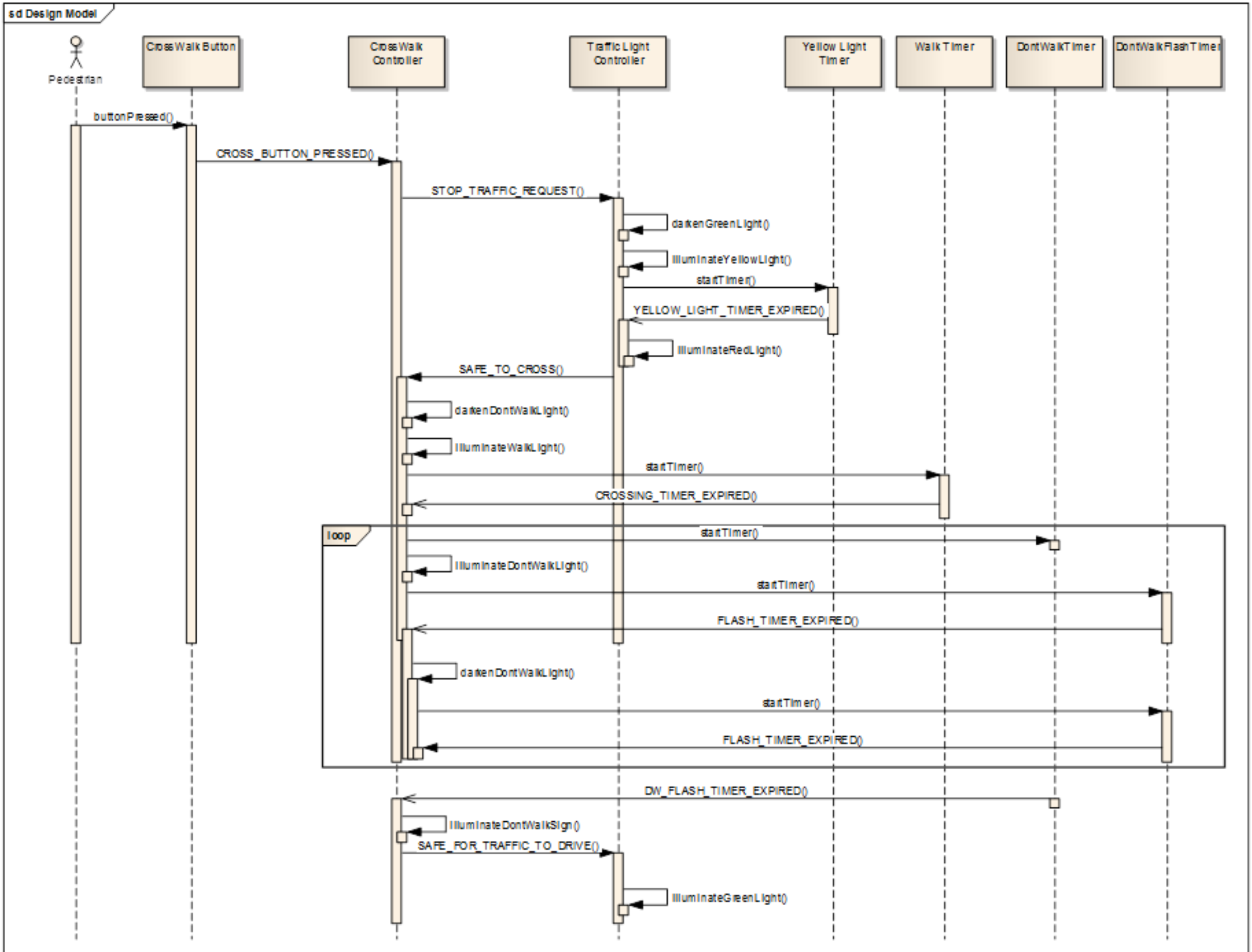
objects

- The above cases are called scenarios
  - Order dependent view of how objects behave

Interaction Diagram


Pedestrian
Cross Walk Button
Cross Walk Controller
Traffic Light Controller
Yellow Light Timer
Walk Timer
buttonPressed()
CROSS_BUTTON_PRESSED()
STOP_TRAFFIC_REQUEST()
darkenGreenLight()
illuminateYellowLight()
startTimer()
YELLOW_LIGHT_TIMER_EXPIRED()
illuminateRedLight()
SAFE_TO_CROSS()
darkenDontWalkLight()
illuminateWalkLight()
startTimer()
CROSSING_TIMER_EXPIRED()


Pedestrian
CrossWalkButton
Cross Walk Controller
Traffic Light Controller
Yellow Light Timer
WalkTimer
DontWalkTimer
DontWalkFlashTimer
CROSSING_TIMER_EXPIRED()
loop
startTimer()
illuminateDontWalkLight()
startTimer()
FLASH_TIMER_EXPIRED()
darkenDontWalkLight()
startTimer()
FLASH_TIMER_EXPIRED()
DW_FLASH_TIMER_EXPIRED()
illuminateDontWalkSign()
SAFE_FOR_TRAFFIC_TO_DRIVE()
illuminateGreenLight()


sd Design Model
Pedestrian
CrossWalk Button
CrossWalk Controller
Traffic Light Controller
Yellow Light Timer
Walk Timer
DontWalkTimer
DontWalkFlashTimer
buttonPressed()
CROSS_BUTTON_PRESSED()
STOP_TRAFFIC_REQUEST()
darkenGreenLight()
illuminateYellowLight()
startTimer()
YELLOW_LIGHT_TIMER_EXPIRED()
illuminateRedLight()
SAFE_TO_CROSS()
darkenDontWalkLight()
illuminateWalkLight()
startTimer()
CROSSING_TIMER_EXPIRED()
loop
startTimer()
illuminateDontWalkLight()
startTimer()
FLASH_TIMER_EXPIRED()
darkenDontWalkLight()
startTimer()
FLASH_TIMER_EXPIRED()
DW_FLASH_TIMER_EXPIRED()
illuminateDontWalkSign()
SAFE_FOR_TRAFFIC_TO_DRIVE()
illuminateGreenLight()

Types of Operations

- Constructor
  - Create an instance of an object
  - "Instantiate" the object (aka new)
- Destructor
  - Destroys an existing instance of an object
- Modifier
  - Changes values within an object
- Selector
  - Read values or request services from an object without modifying them
- Iterator
  - Provide orderly access to a set of objects

Defining Operations

- Provide a set of orthogonal interface operations
- Hide the internal class structure
- All messages into an object must be accepted
  - Each message must have a corresponding acceptor operation
- Mutators and assessors should be provided as is necessary

methods of classes

- Have a nice break!

Homework