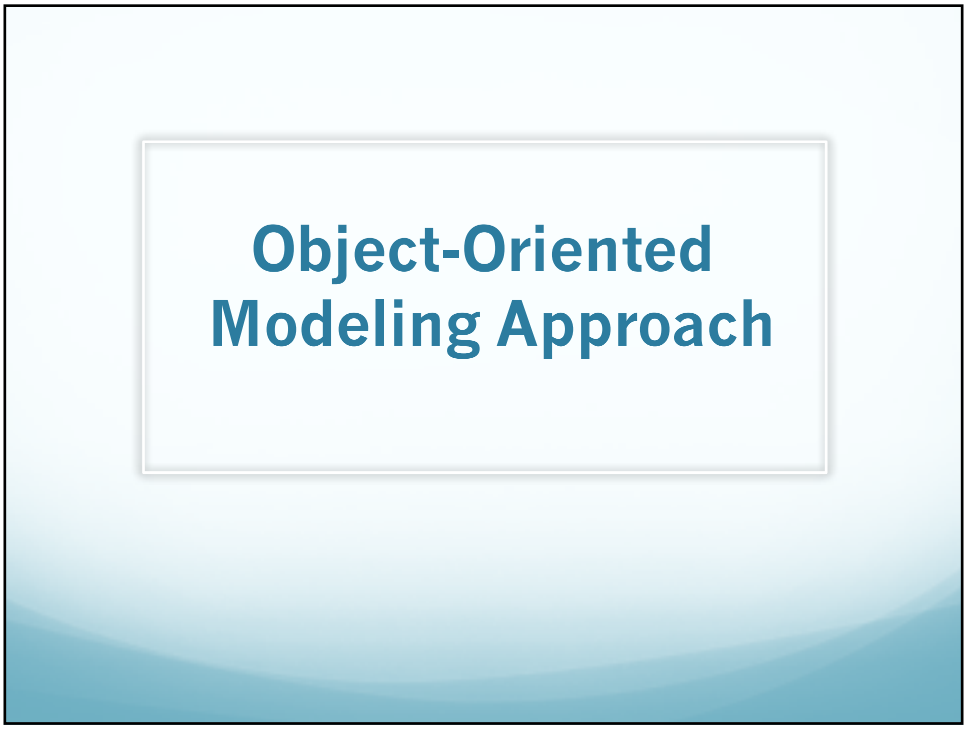
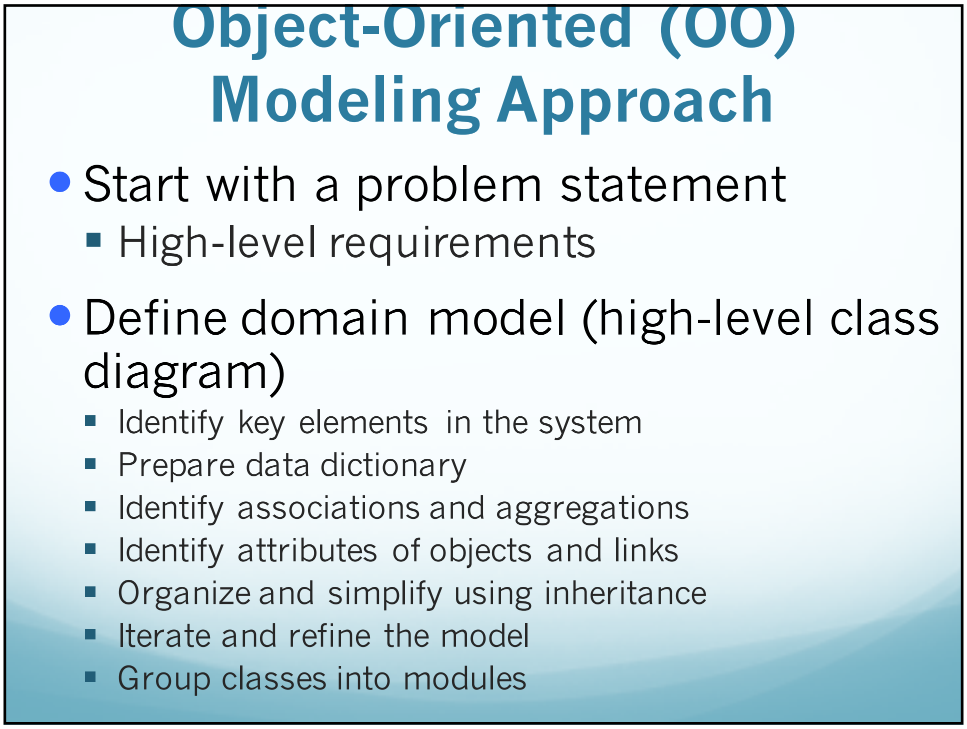


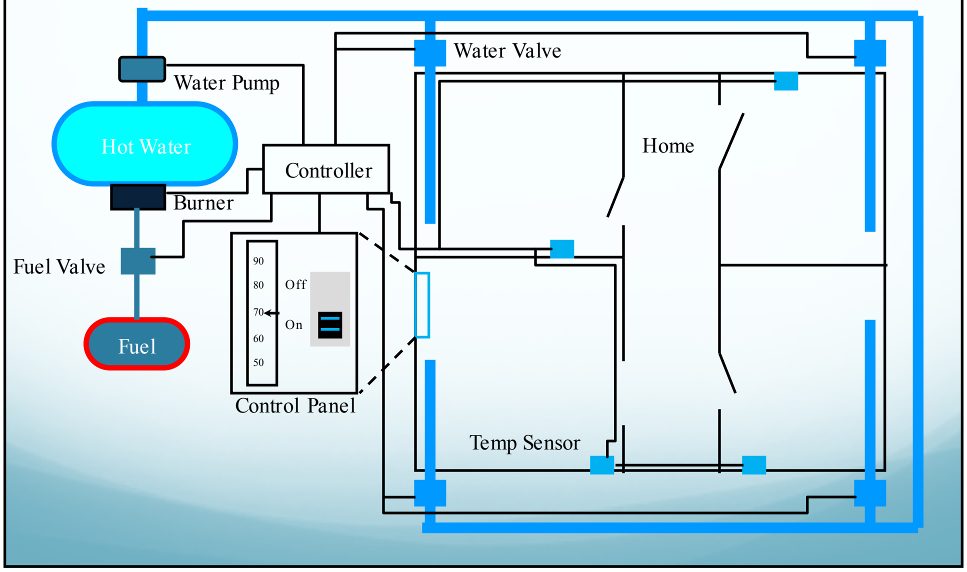
Object-Oriented Modeling Approach

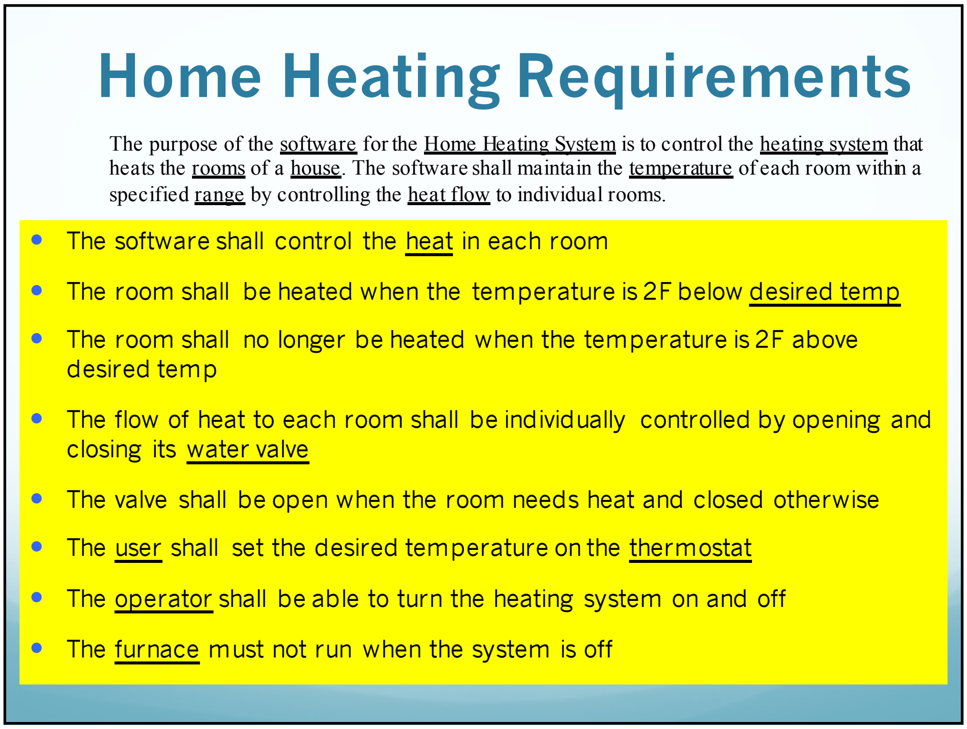


Object-Oriented (OO) Modeling Approach

- Start with a problem statement
  - High-level requirements
- Define domain model (high-level class diagram)
  - Identify key elements in the system
  - Prepare data dictionary
  - Identify associations and aggregations
  - Identify attributes of objects and links
  - Organize and simplify using inheritance
  - Iterate and refine the model
  - Group classes into modules

The Home Heating System

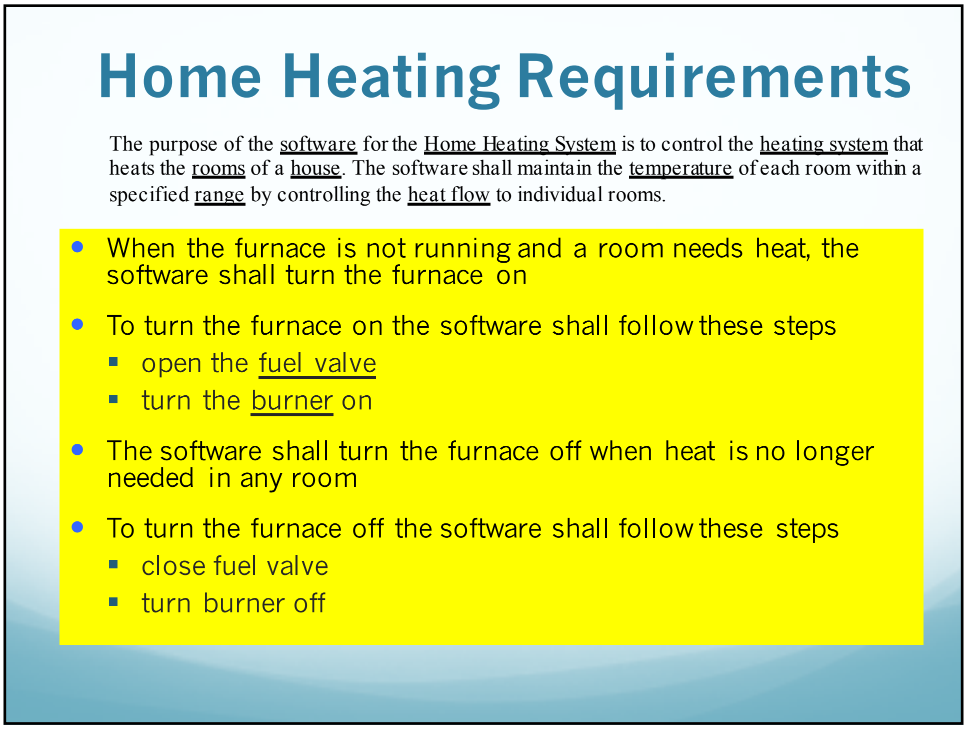




Home Heating Requirements

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

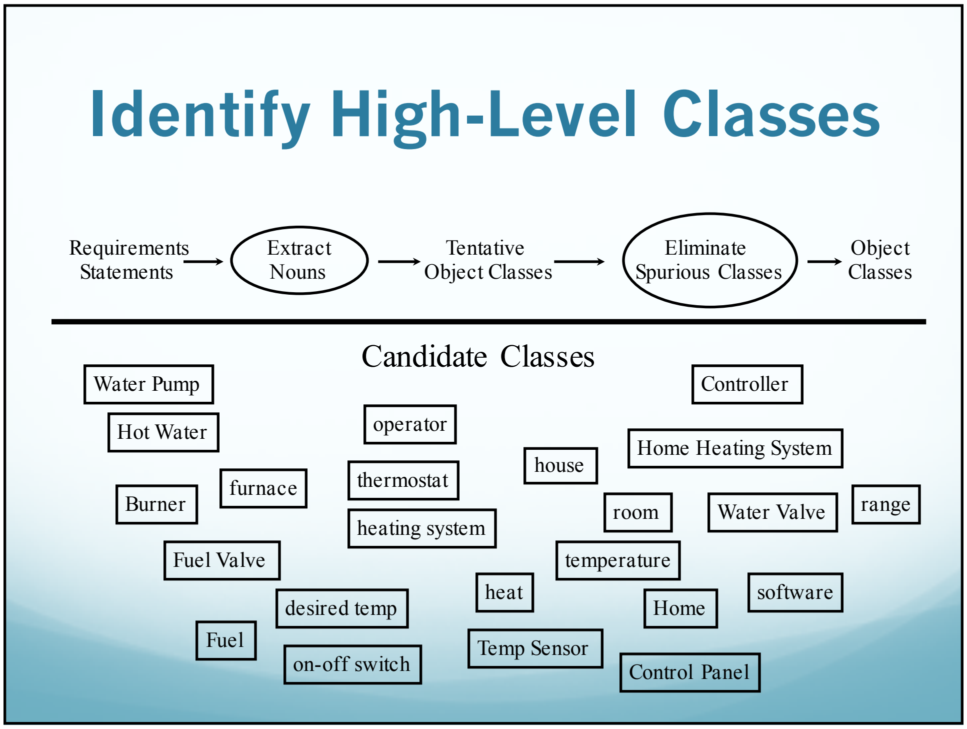
- The software shall control the heat in each room
- The room shall be heated when the temperature is 2F below desired temp
- The room shall no longer be heated when the temperature is 2F above desired temp
- The flow of heat to each room shall be individually controlled by opening and closing its water valve
- The valve shall be open when the room needs heat and closed otherwise
- The user shall set the desired temperature on the thermostat
- The operator shall be able to turn the heating system on and off
- The furnace must not run when the system is off



Home Heating Requirements

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

- When the furnace is not running and a room needs heat, the software shall turn the furnace on
- To turn the furnace on the software shall follow these steps
  - open the fuel valve
  - turn the burner on
- The software shall turn the furnace off when heat is no longer needed in any room
- To turn the furnace off the software shall follow these steps
  - close fuel valve
  - turn burner off



Identify High-Level Classes

Requirements Statements → Extract Nouns → Tentative Object Classes → Eliminate Spurious Classes → Object Classes

Candidate Classes

Water Pump

Controller

Hot Water

operator

house

Home Heating System

furnace

thermostat

Burner

room

Water Valve

range

heating system

Fuel Valve

temperature

heat

software

desired temp

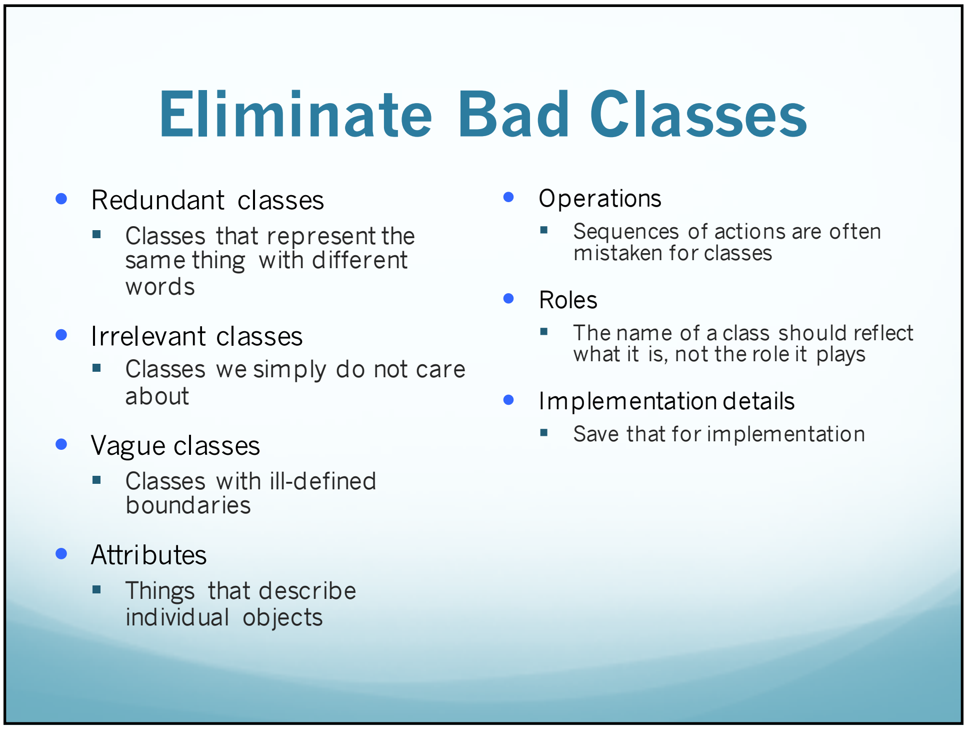
Home

Fuel

on-off switch

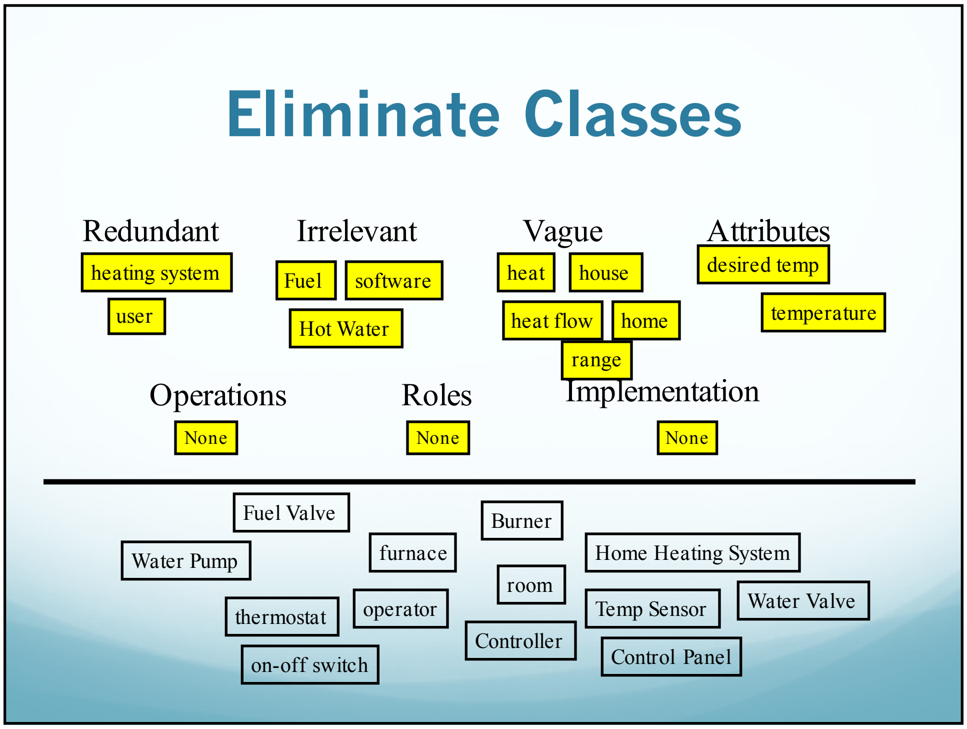
Temp Sensor

Control Panel



Eliminate Bad Classes

- Redundant classes
  - Classes that represent the same thing with different words
- Irrelevant classes
  - Classes we simply do not care about
- Vague classes
  - Classes with ill-defined boundaries
- Attributes
  - Things that describe individual objects
- Operations
  - Sequences of actions are often mistaken for classes
- Roles
  - The name of a class should reflect what it is, not the role it plays
- Implementation details
  - Save that for implementation



Eliminate Classes

Redundant

heating system

user

Irrelevant

Fuel

software

Hot Water

Vague

heat

house

heat flow

home

range

Attributes

desired temp

temperature

Operations

None

Roles

None

Implementation

None

Fuel Valve

Burner

Water Pump

furnace

Home Heating System

room

thermostat

operator

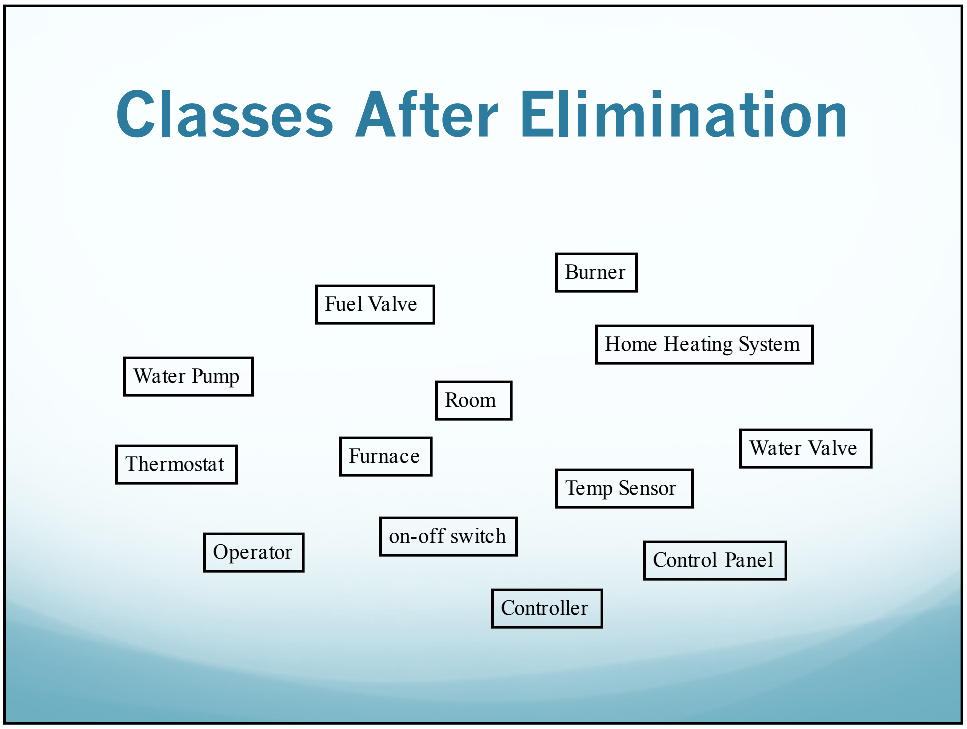
Temp Sensor

Water Valve

Controller

on-off switch

Control Panel



Classes After Elimination

Burner

Fuel Valve

Home Heating System

Water Pump

Room

Water Valve

Thermostat

Furnace

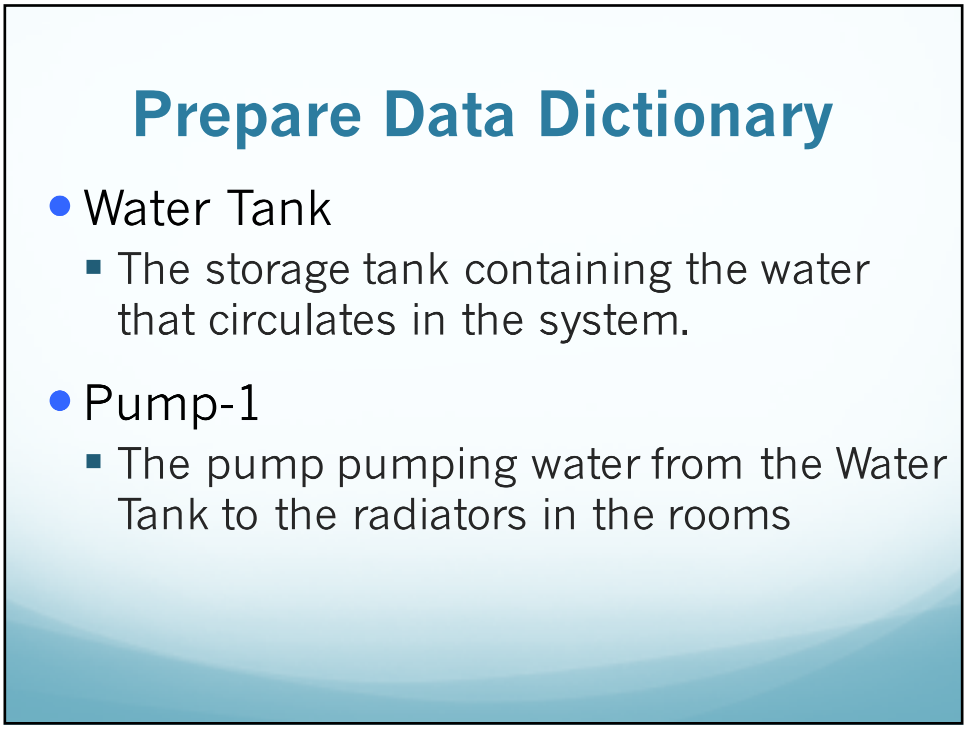
Temp Sensor

on-off switch

Operator

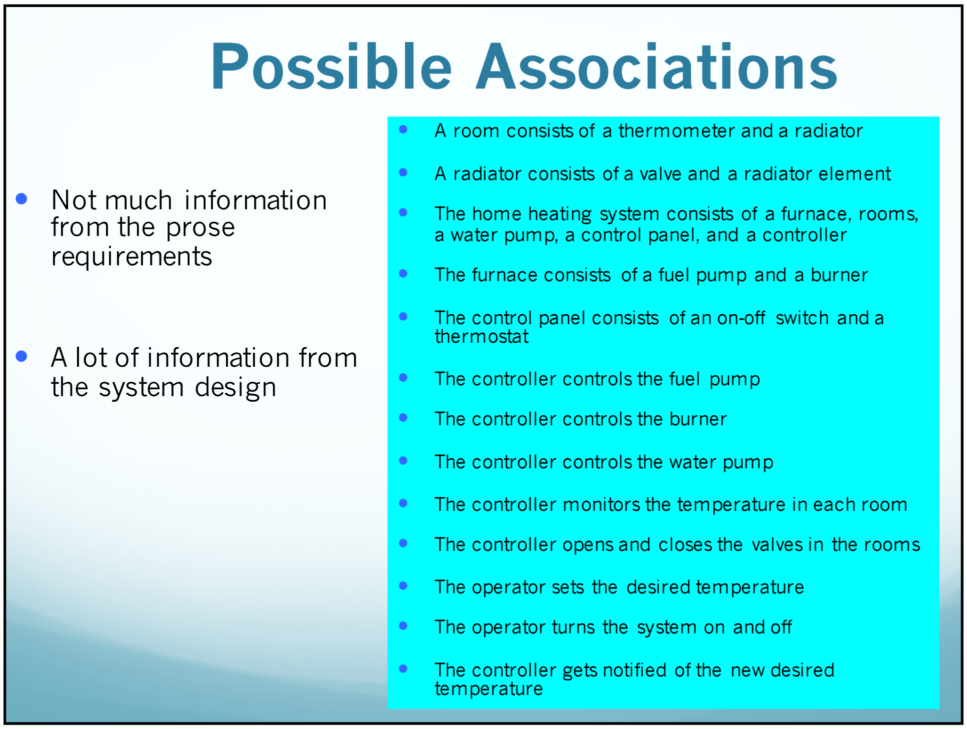
Control Panel

Controller



Prepare Data Dictionary

- Water Tank
  - The storage tank containing the water that circulates in the system.
- Pump-1
  - The pump pumping water from the Water Tank to the radiators in the rooms

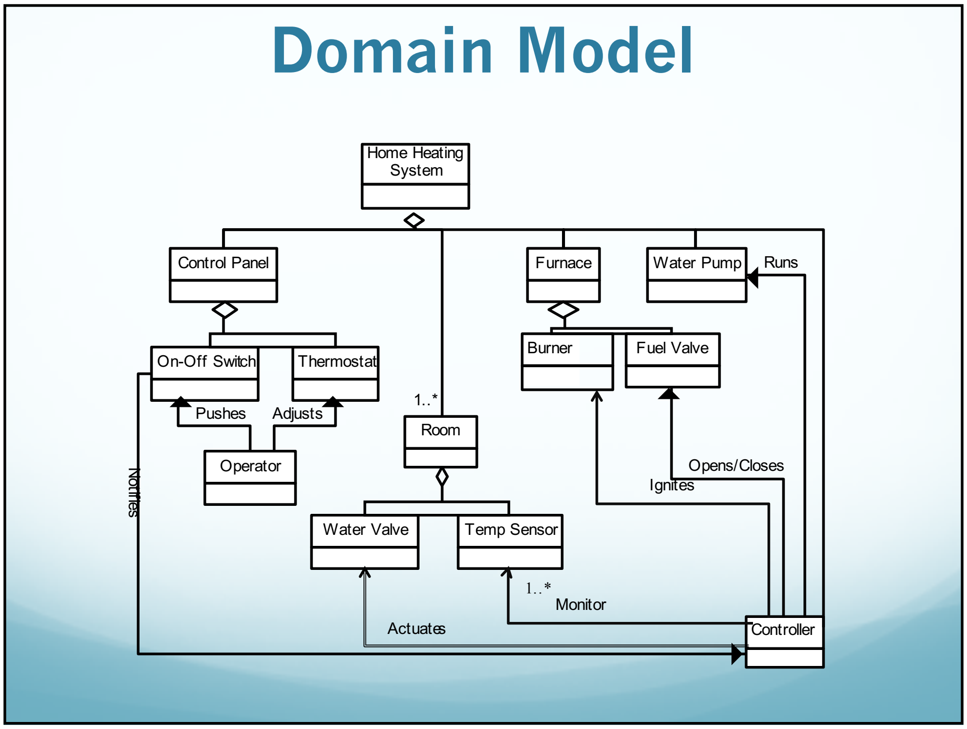


Possible Associations

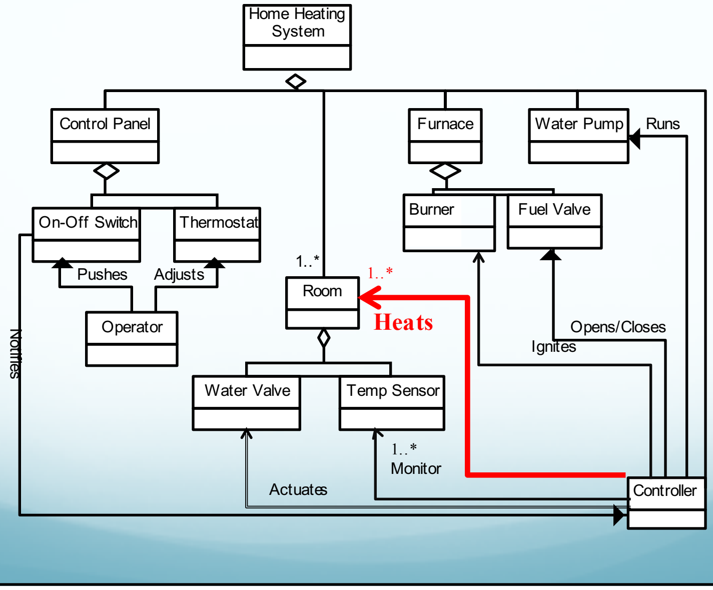
- Not much information from the prose requirements
- A lot of information from the system design

- A room consists of a thermometer and a radiator
- A radiator consists of a valve and a radiator element
- The home heating system consists of a furnace, rooms, a water pump, a control panel, and a controller
- The furnace consists of a fuel pump and a burner
- The control panel consists of an on-off switch and a thermostat
- The controller controls the fuel pump
- The controller controls the burner
- The controller controls the water pump
- The controller monitors the temperature in each room
- The controller opens and closes the valves in the rooms
- The operator sets the desired temperature
- The operator turns the system on and off
- The controller gets notified of the new desired temperature

Domain Model



Domain Model



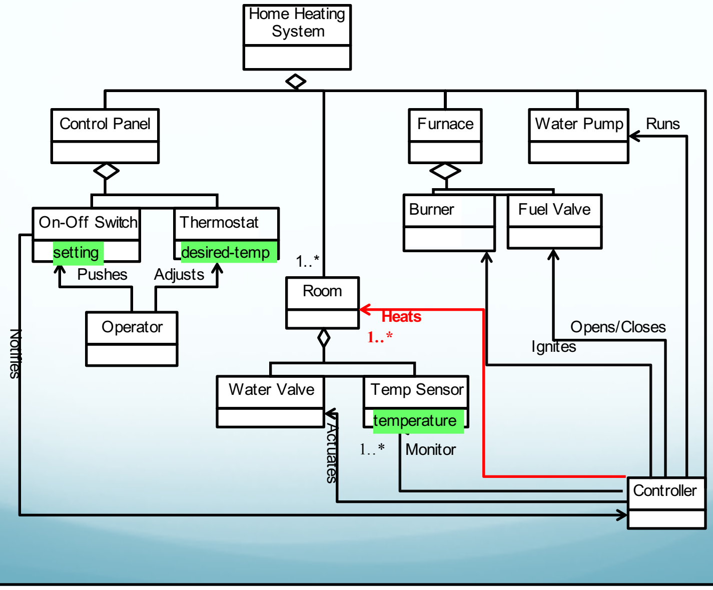
Attributes

Thermostat
desired-temp

On-Off switch
setting

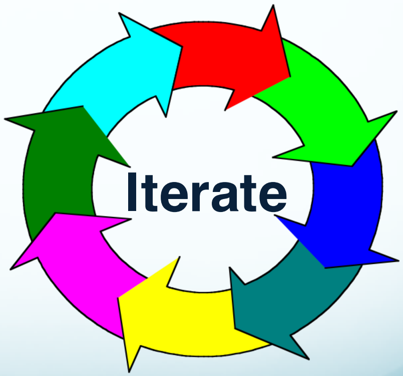
Temp Sensor
temperature

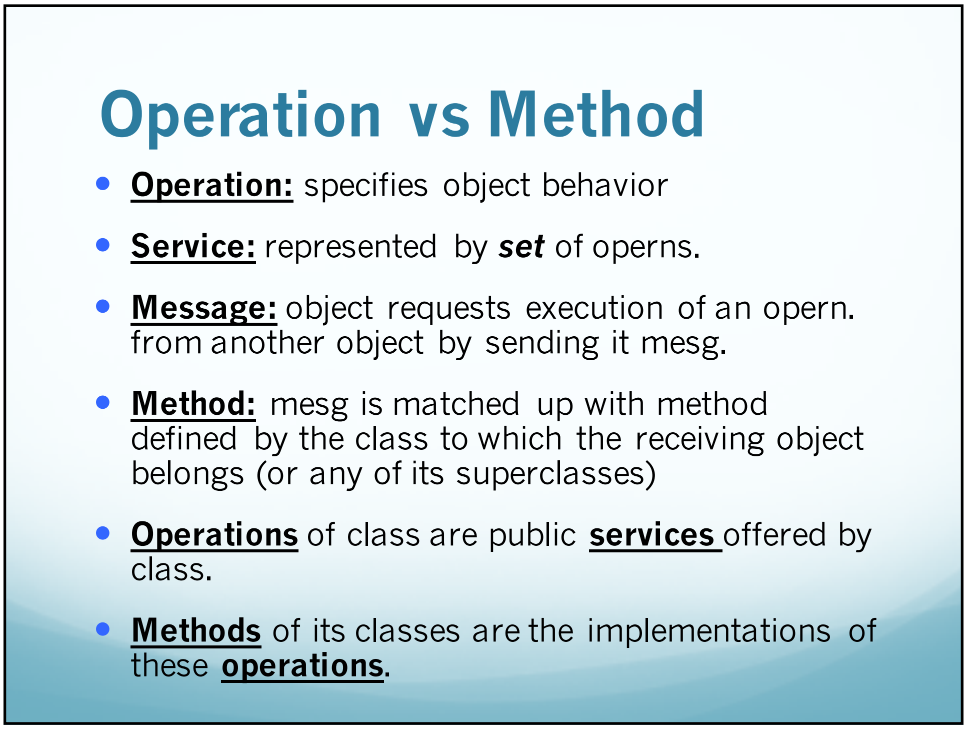
Final Domain Model



Iterate the Model

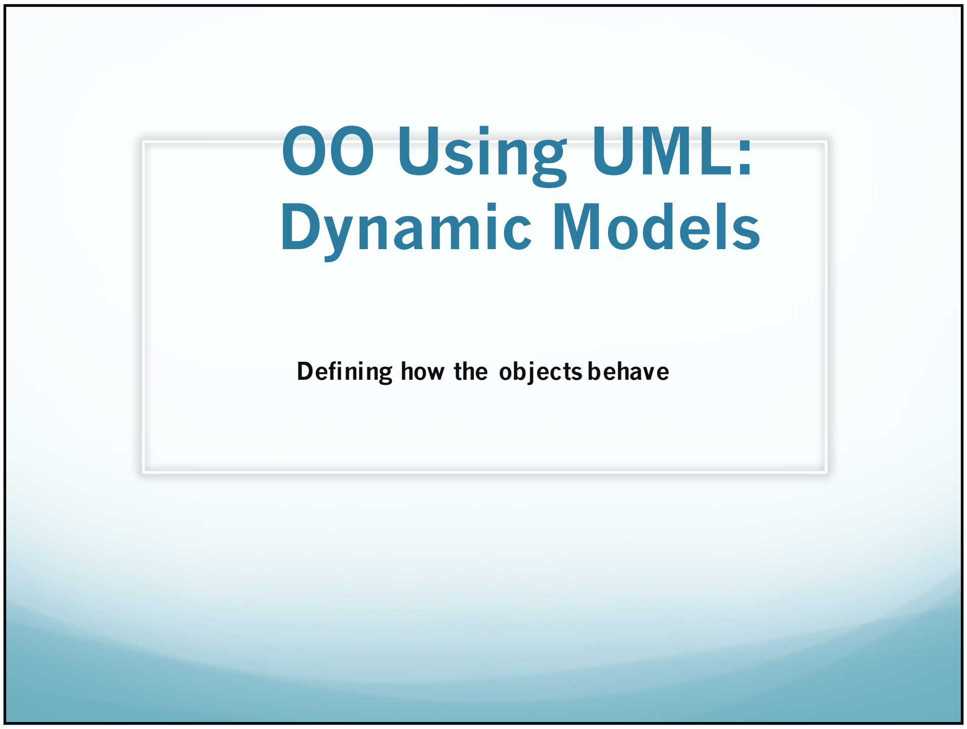
- Keep on doing this until you, your customer, and your engineers are happy with the model





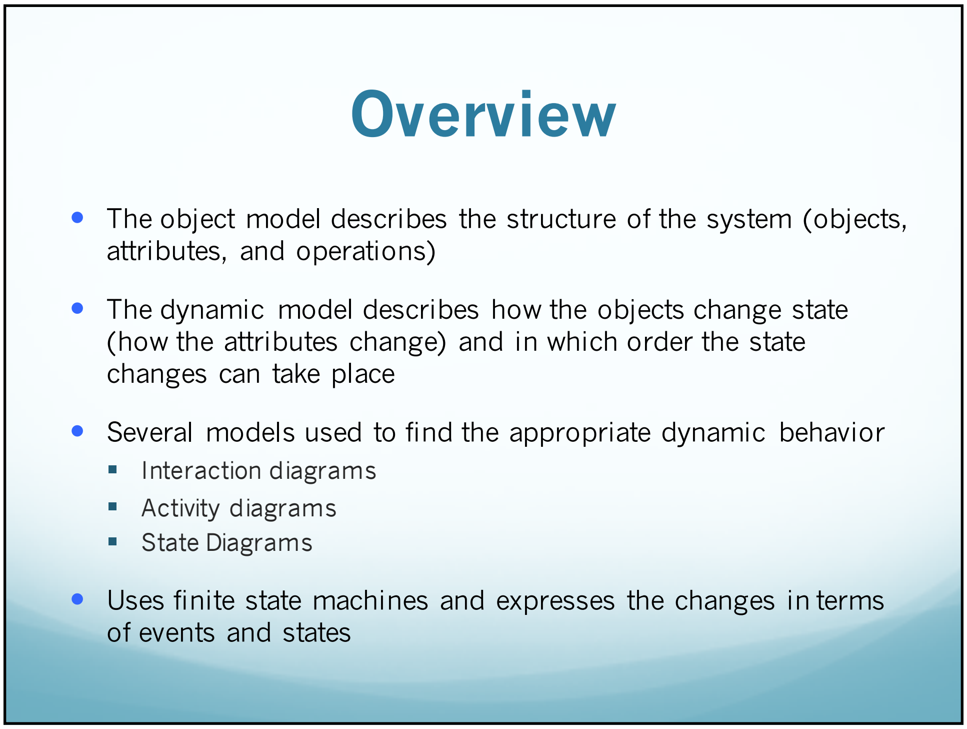
Operation vs Method

- **Operation:** specifies object behavior
- **Service:** represented by ***set*** of operns.
- **Message:** object requests execution of an opern. from another object by sending it mesg.
- **Method:** mesg is matched up with method defined by the class to which the receiving object belongs (or any of its superclasses)
- **Operations** of class are public **services** offered by class.
- **Methods** of its classes are the implementations of these **operations**.



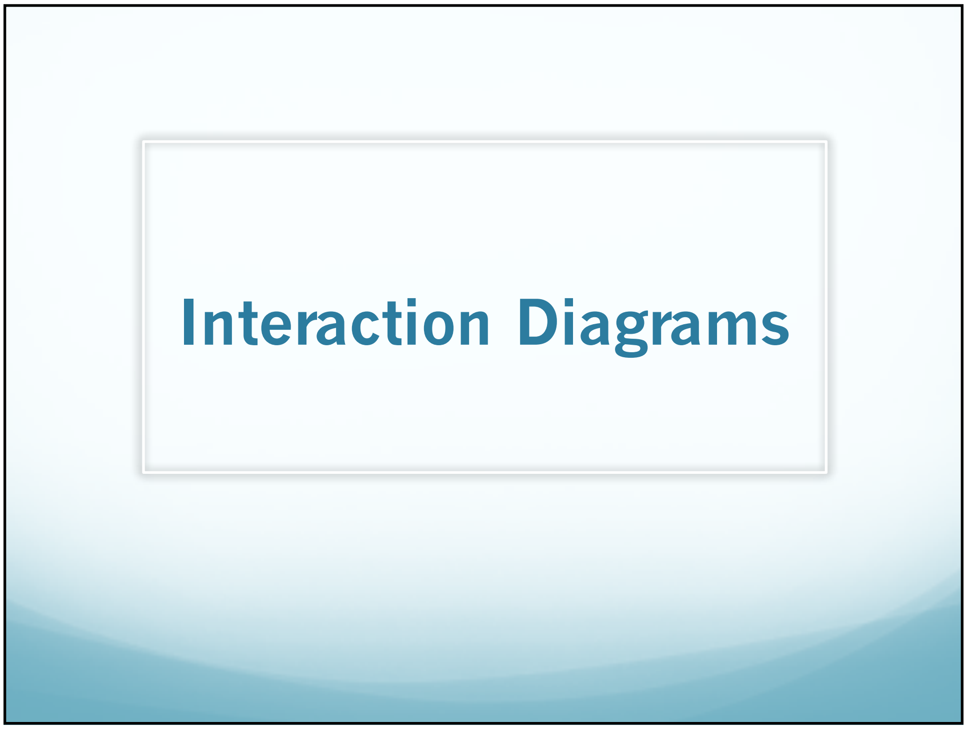
OO Using UML: Dynamic Models

Defining how the objects behave

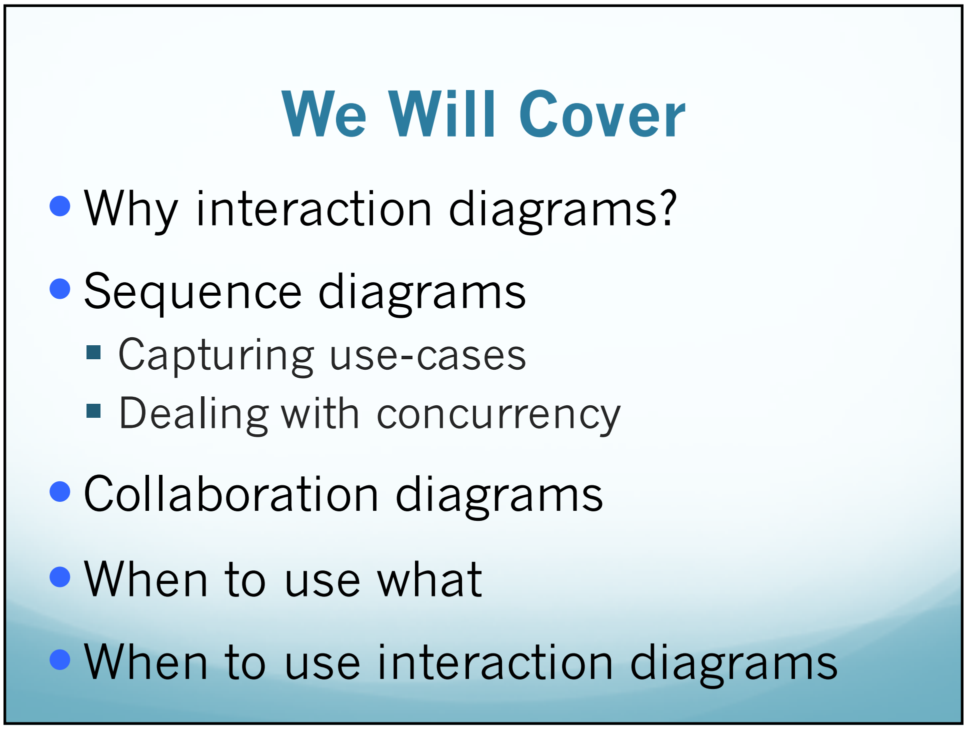


Overview

- The object model describes the structure of the system (objects, attributes, and operations)
- The dynamic model describes how the objects change state (how the attributes change) and in which order the state changes can take place
- Several models used to find the appropriate dynamic behavior
  - Interaction diagrams
  - Activity diagrams
  - State Diagrams
- Uses finite state machines and expresses the changes in terms of events and states

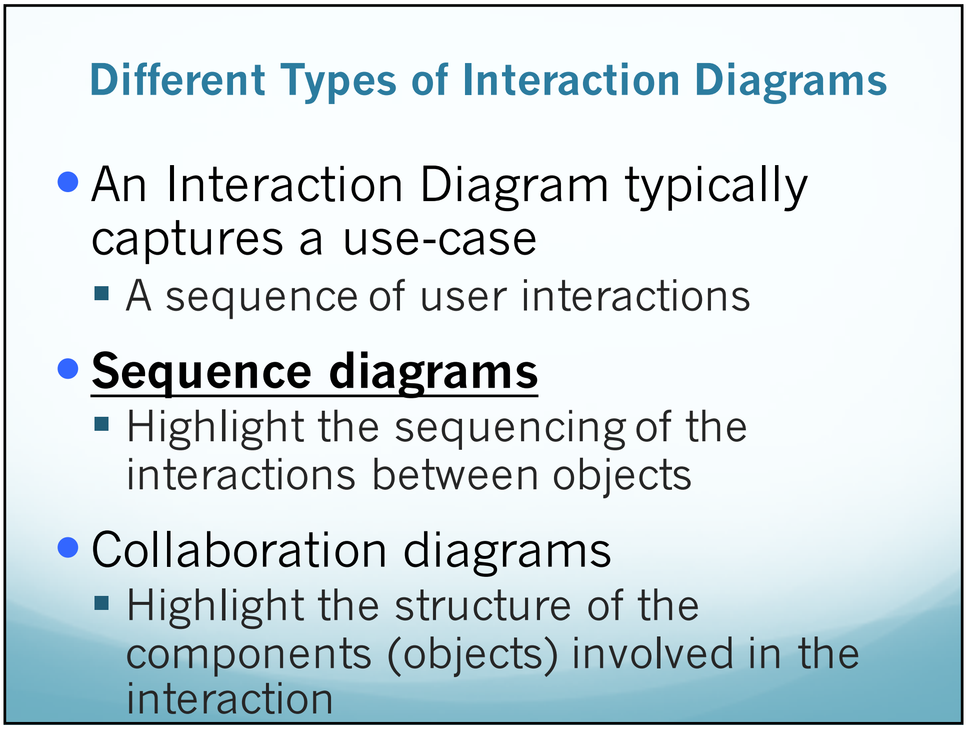


Interaction Diagrams



We Will Cover

- Why interaction diagrams?
- Sequence diagrams
  - Capturing use-cases
  - Dealing with concurrency
- Collaboration diagrams
- When to use what
- When to use interaction diagrams



Different Types of Interaction Diagrams

- An Interaction Diagram typically captures a use-case
  - A sequence of user interactions
- **Sequence diagrams**
  - Highlight the sequencing of the interactions between objects
- Collaboration diagrams
  - Highlight the structure of the components (objects) involved in the interaction

Home Heating Use-Case

Use case: **Power Up**

Actors: Home Owner (initiator)

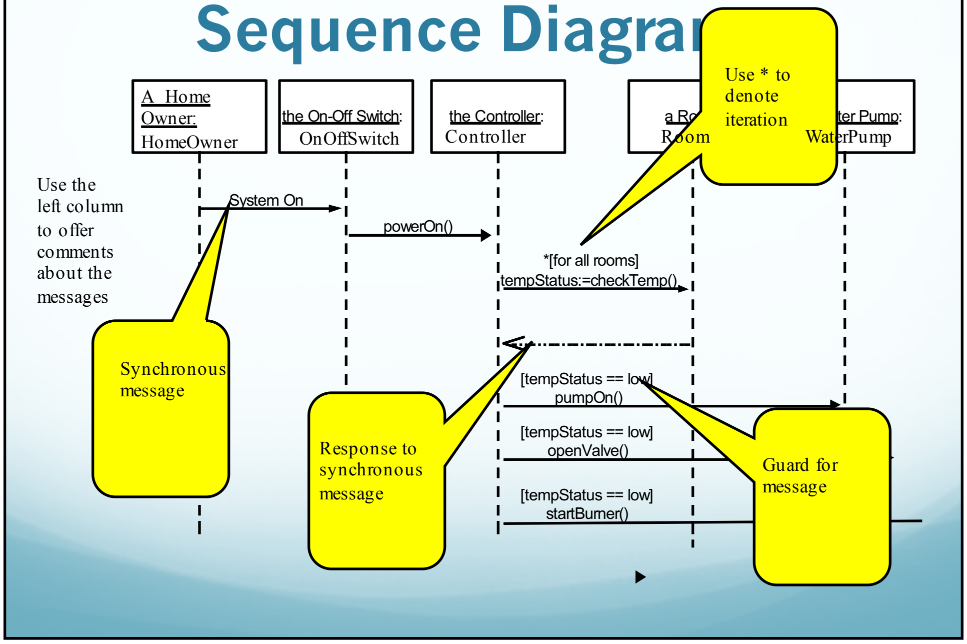
Type: Primary and essential

Description: The Home Owner turns the power on. Each room is temperature checked. If a room is below the the desired temperature the valve for the room is opened, the water pump started, the fuel valve opened, and the burner ignited.
If the temperature in all rooms is above the desired temperature, no actions are taken.

Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: None

Sequence Diagra



A Home Owner: HomeOwner

the On-Off Switch: OnOffSwitch

the Controller: Controller

a Ro Room

ter Pump: WaterPump

Use the left column to offer comments about the messages

System On

powerOn()

*[for all rooms] tempStatus:=checkTemp()

[tempStatus == low] pumpOn()

[tempStatus == low] openValve()

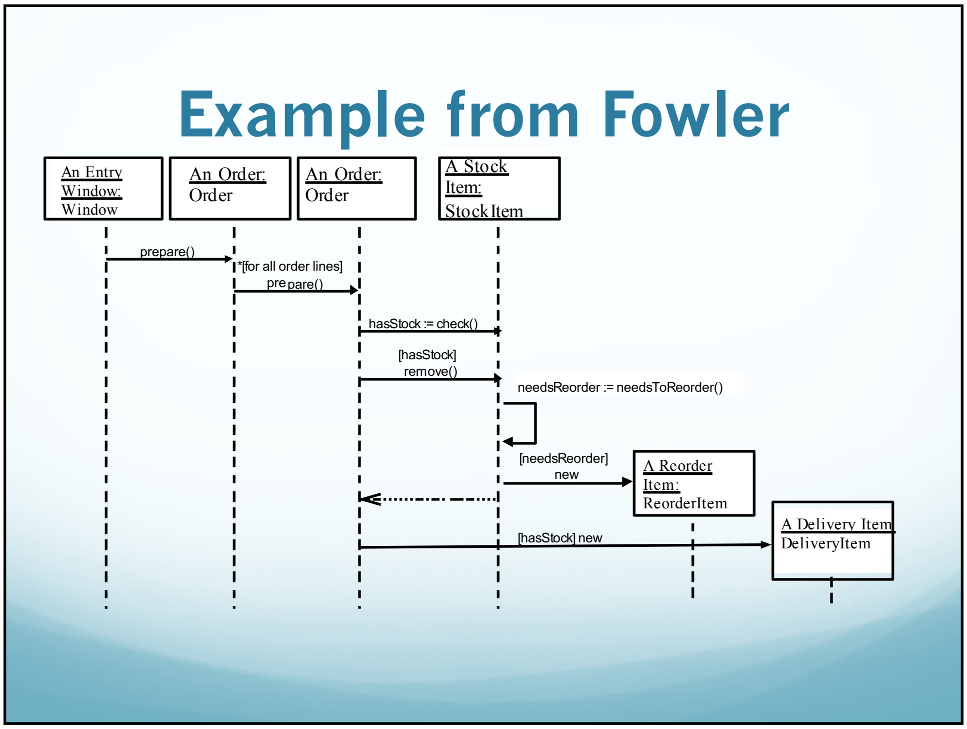
[tempStatus == low] startBurner()

Synchronous message

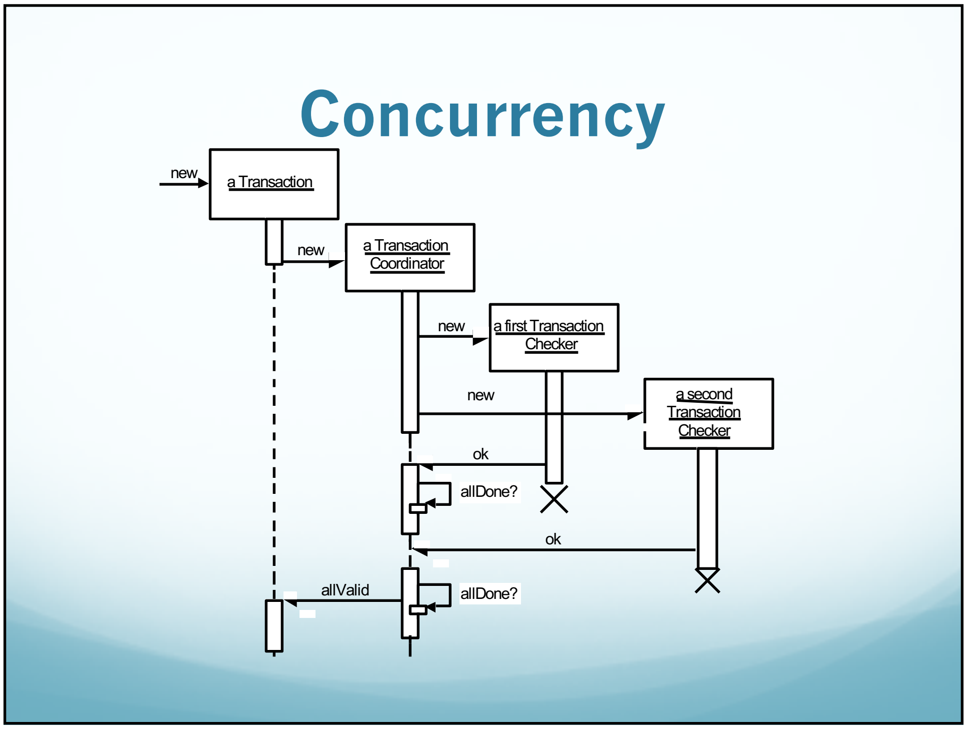
Response to synchronous message

Use * to denote iteration

Guard for message

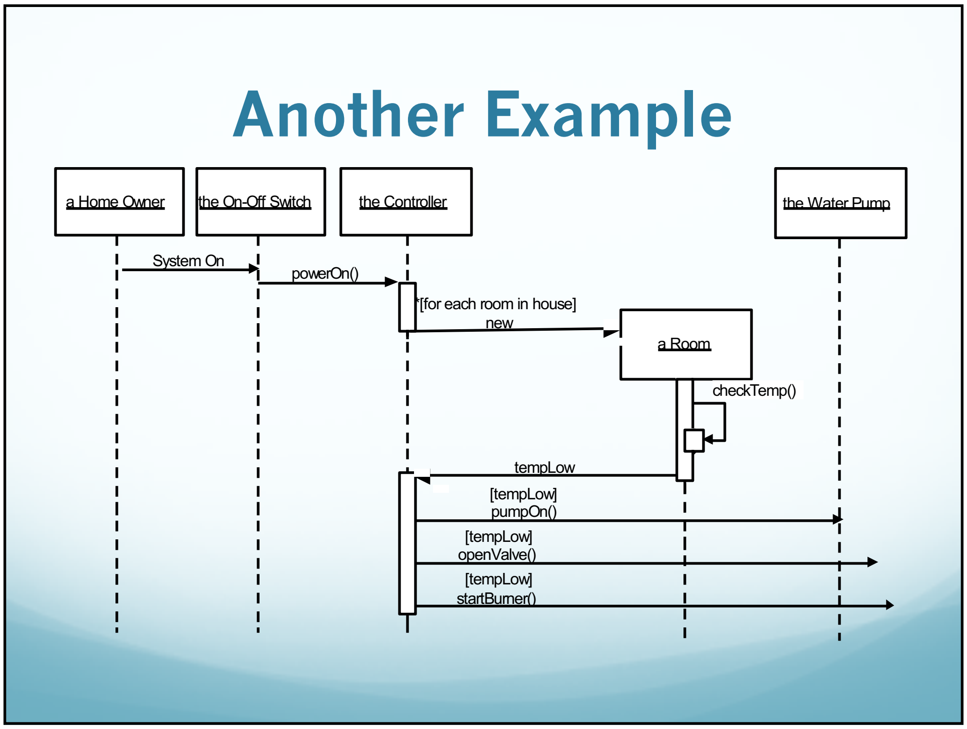


Example from Fowler



Concurrency

new
a Transaction
new
a Transaction Coordinator
new
a first Transaction Checker
new
a second Transaction Checker
ok
allDone?
ok
allValid
allDone?



Another Example

a Home Owner

the On-Off Switch

the Controller

the Water Pump

System On

powerOn()

*[for each room in house]
new

a Room

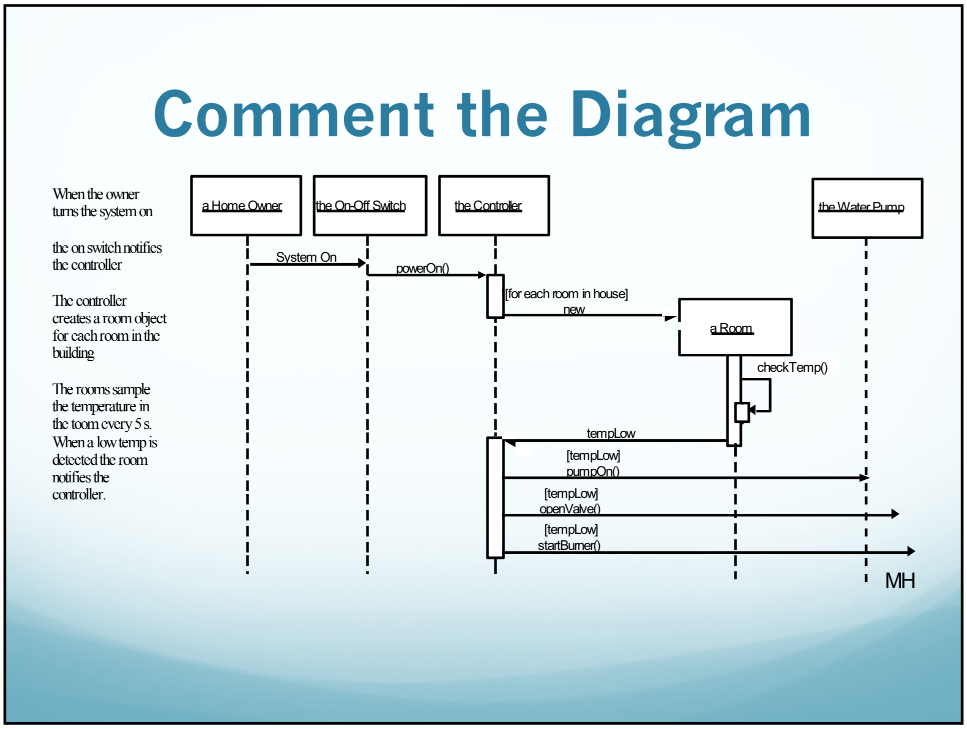
checkTemp()

tempLow

[tempLow]
pumpOn()

[tempLow]
openValve()

[tempLow]
startBurner()



Comment the Diagram

When the owner turns the system on

the on switch notifies the controller

The controller creates a room object for each room in the building

The rooms sample the temperature in the toom every 5 s. When a low temp is detected the room notifies the controller.

a Home Owner

the On-Off Switch

the Controller

the Water Pump

System On

powerOn()

[for each room in house]
new

a Room

checkTemp()

tempLow

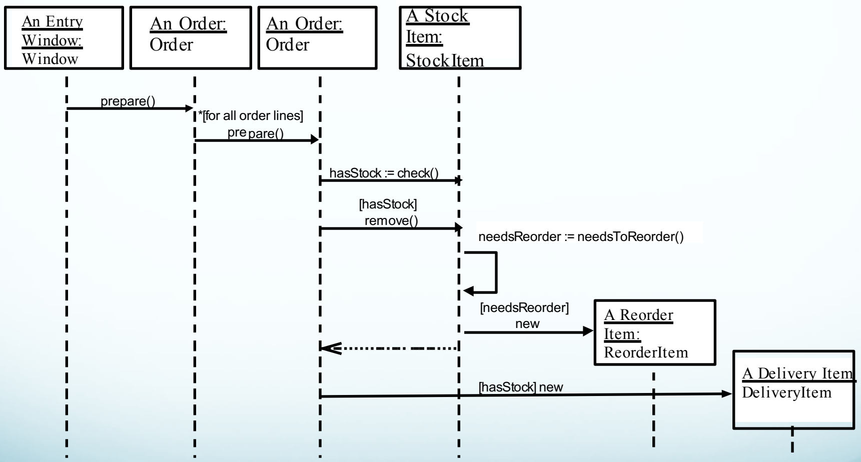
[tempLow]
pumpOn()

[tempLow]
openValve()

[tempLow]
startBurner()

MH

Example from Fowler



An Entry Window: Window

An Order: Order

An Order: Order

A Stock Item: StockItem

prepare()

*[for all order lines] prepare()

hasStock := check()

[hasStock] remove()

needsReorder := needsToReorder()

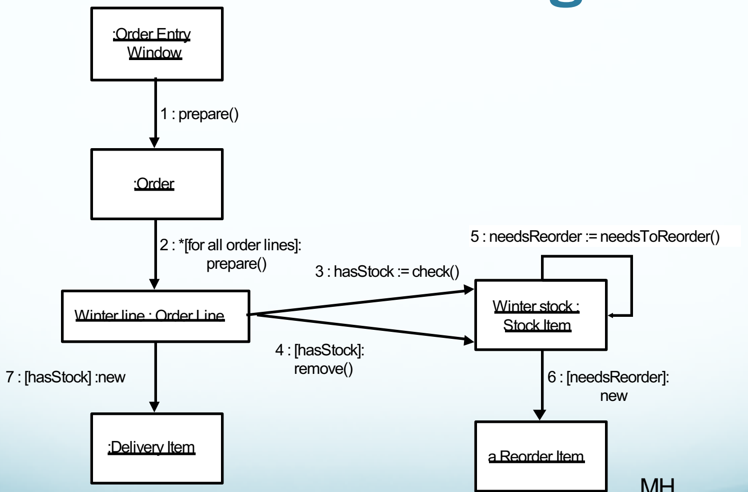
[needsReorder] new

A Reorder Item: ReorderItem

[hasStock] new

A Delivery Item DeliveryItem

Collaboration Diagrams



:Order Entry Window

1 : prepare()

:Order

2 : *[for all order lines]: prepare()

Winter line : Order Line

3 : hasStock := check()

4 : [hasStock]: remove()

5 : needsReorder := needsToReorder()

Winter stock : Stock Item

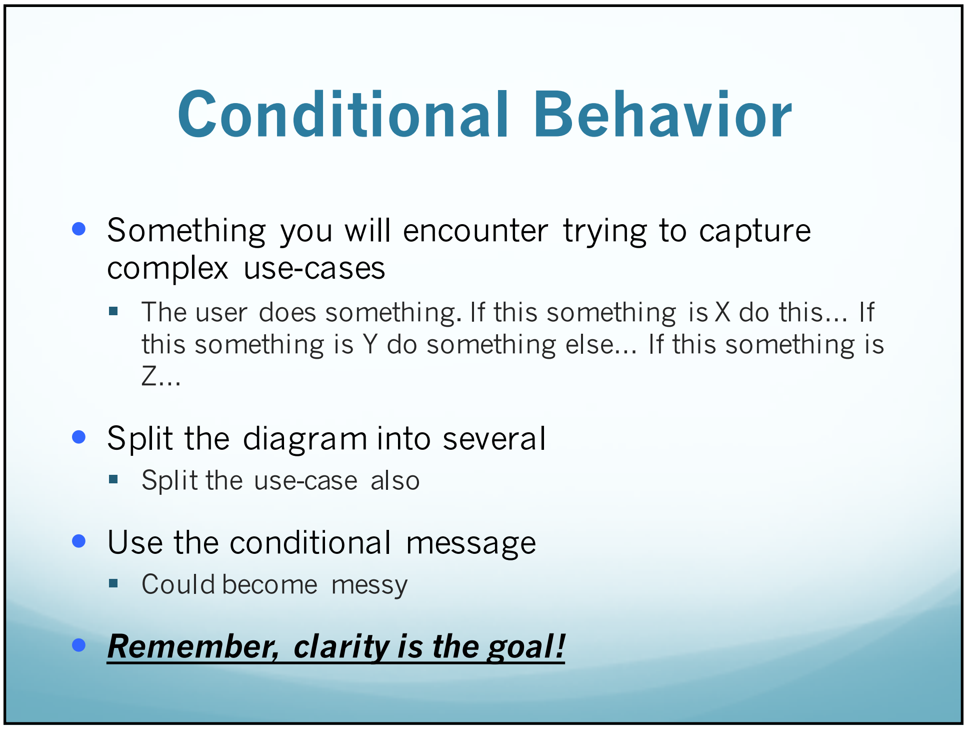
6 : [needsReorder]: new

a Reorder Item

7 : [hasStock] :new

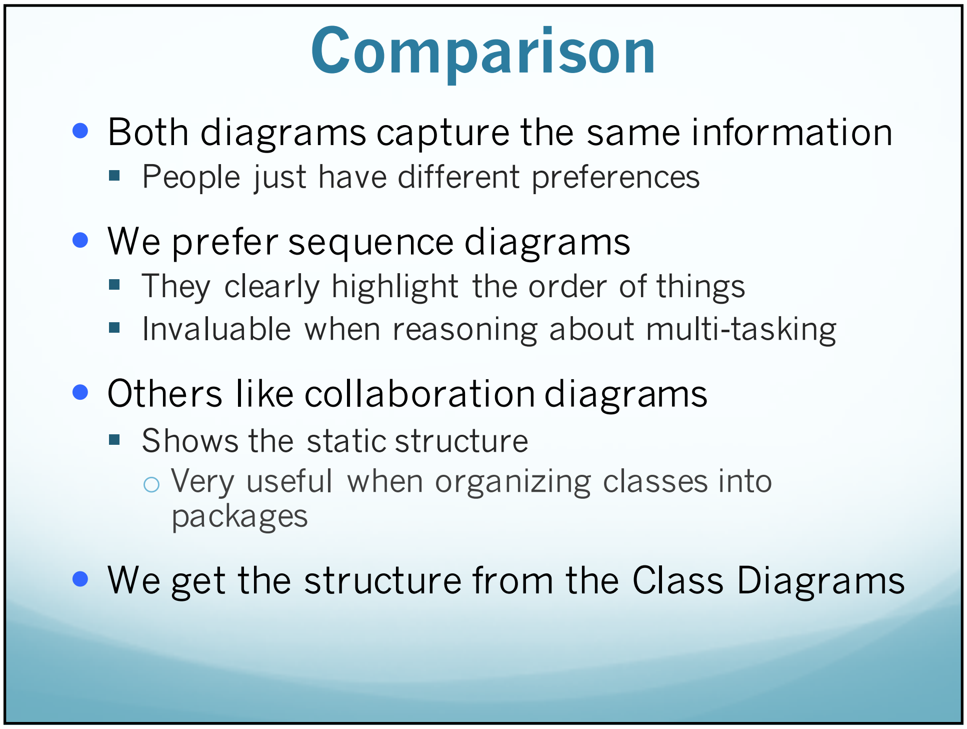
:Delivery Item

MH



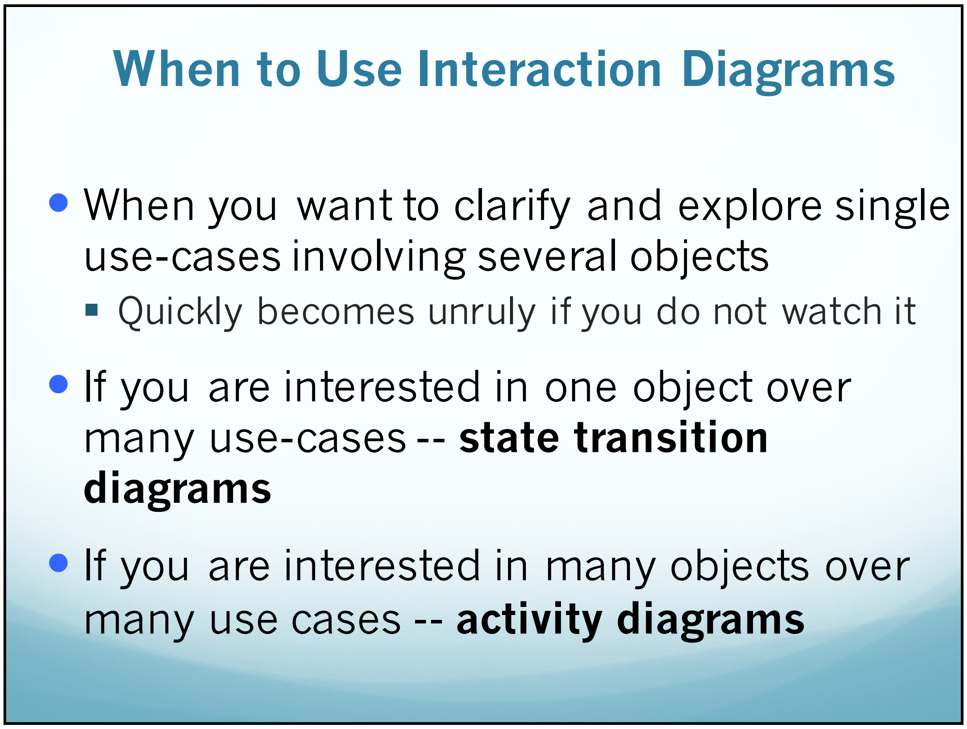
Conditional Behavior

- Something you will encounter trying to capture complex use-cases
  - The user does something. If this something is X do this... If this something is Y do something else... If this something is Z...
- Split the diagram into several
  - Split the use-case also
- Use the conditional message
  - Could become messy
- ***<u>Remember, clarity is the goal!</u>***



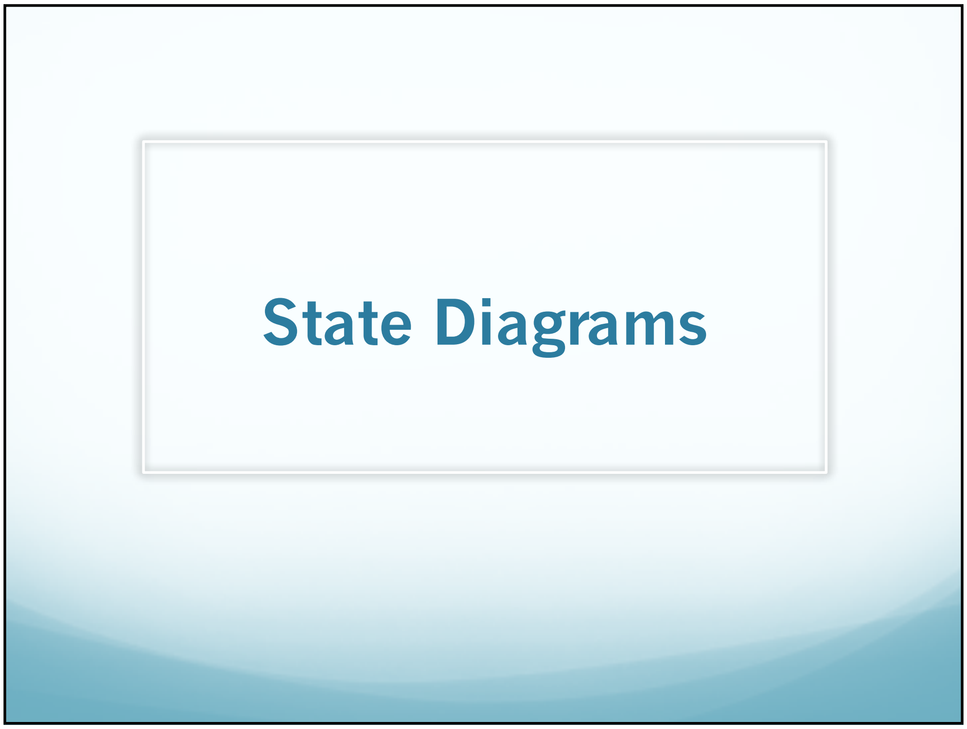
Comparison

- Both diagrams capture the same information
  - People just have different preferences
- We prefer sequence diagrams
  - They clearly highlight the order of things
  - Invaluable when reasoning about multi-tasking
- Others like collaboration diagrams
  - Shows the static structure
    - Very useful when organizing classes into packages
- We get the structure from the Class Diagrams

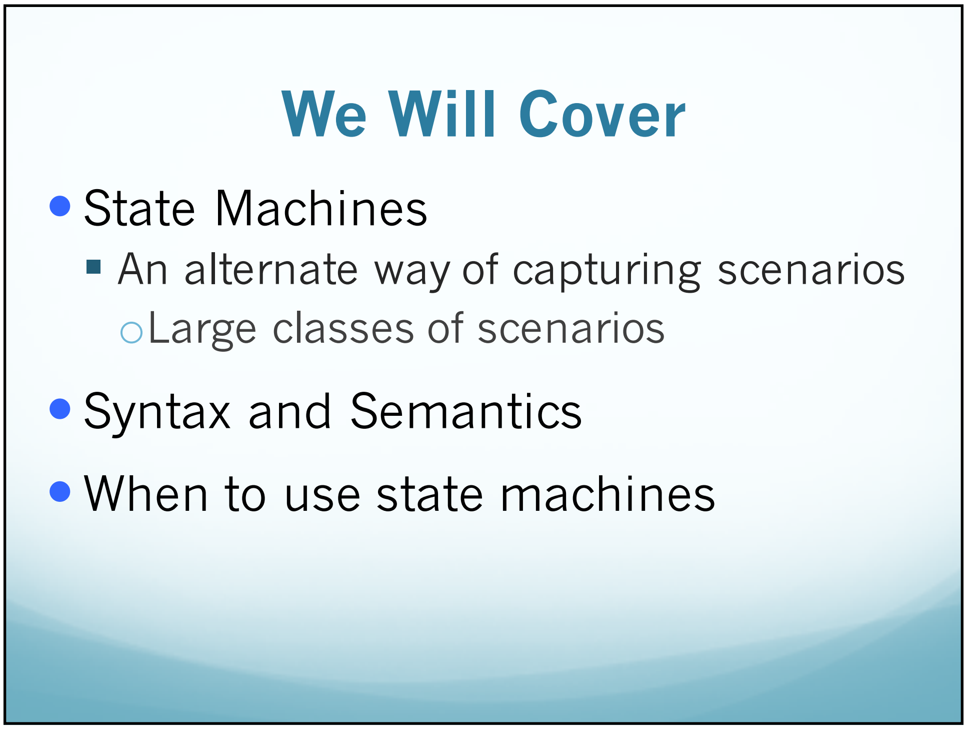


When to Use Interaction Diagrams

- When you want to clarify and explore single use-cases involving several objects
  - Quickly becomes unruly if you do not watch it
- If you are interested in one object over many use-cases -- **state transition diagrams**
- If you are interested in many objects over many use cases -- **activity diagrams**

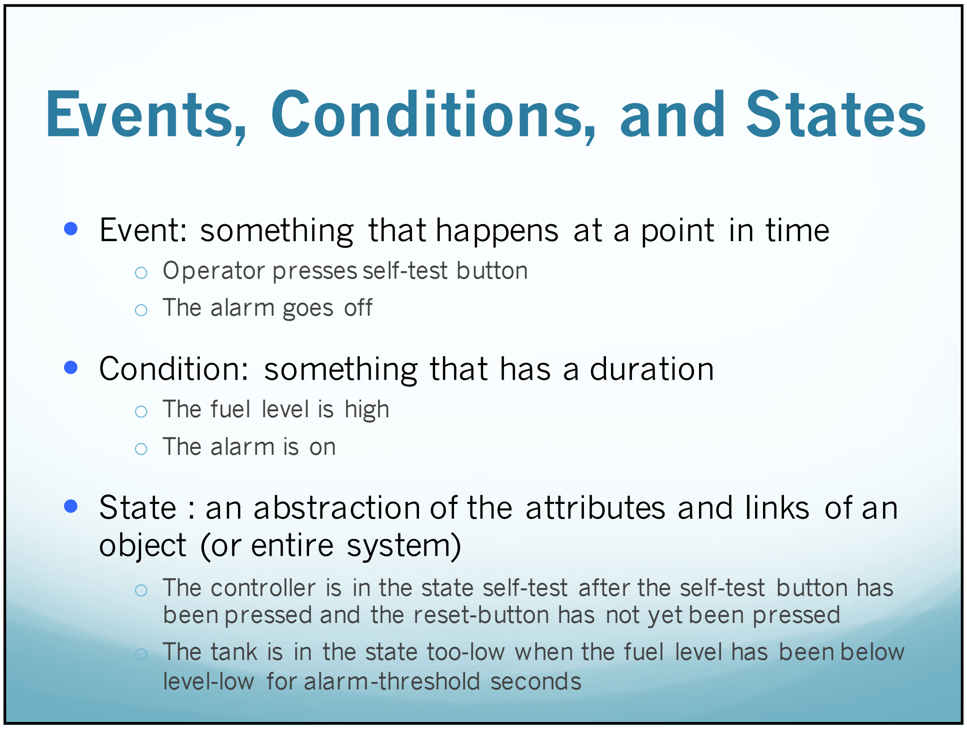


State Diagrams



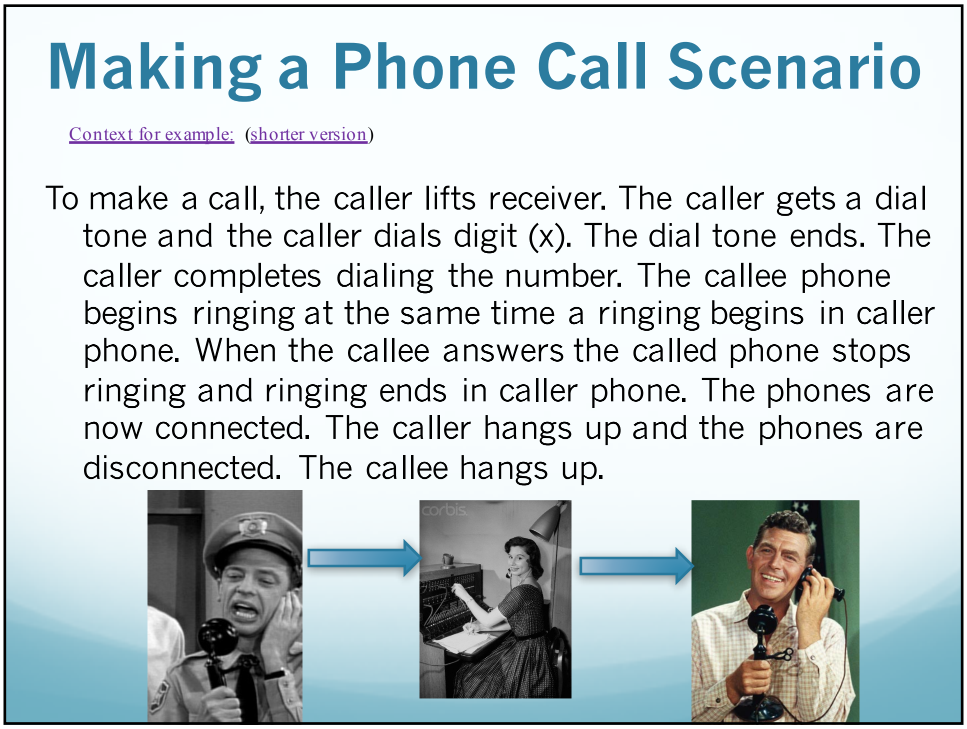
We Will Cover

- State Machines
  - An alternate way of capturing scenarios
    - Large classes of scenarios
- Syntax and Semantics
- When to use state machines



Events, Conditions, and States

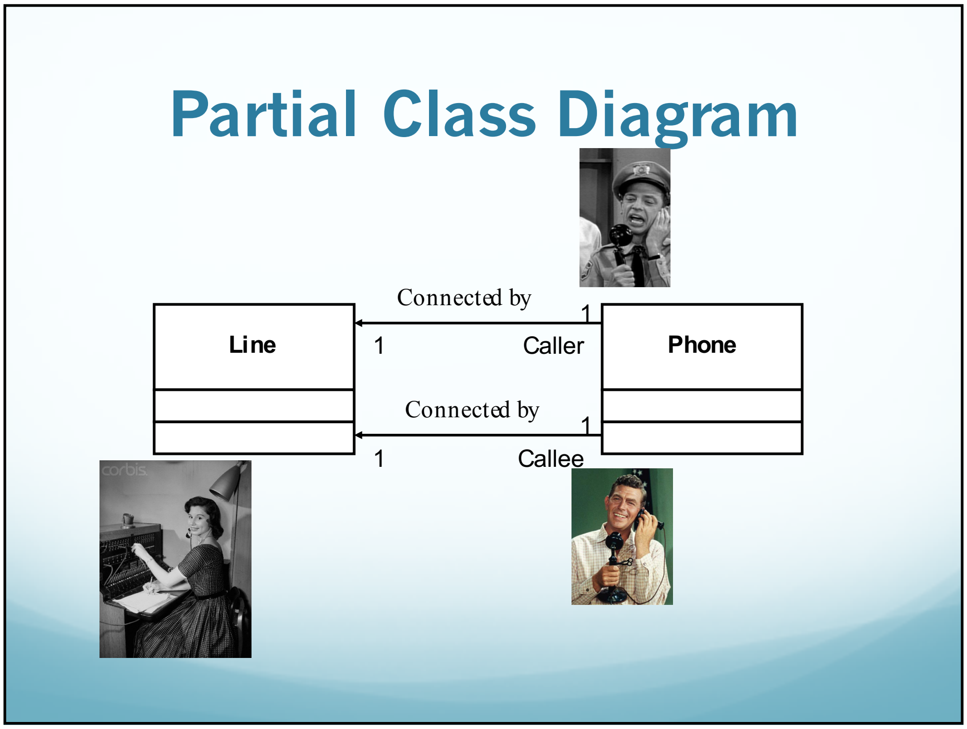
- Event: something that happens at a point in time
  - Operator presses self-test button
  - The alarm goes off
- Condition: something that has a duration
  - The fuel level is high
  - The alarm is on
- State : an abstraction of the attributes and links of an object (or entire system)
  - The controller is in the state self-test after the self-test button has been pressed and the reset-button has not yet been pressed
  - The tank is in the state too-low when the fuel level has been below level-low for alarm-threshold seconds



Making a Phone Call Scenario

Context for example: (shorter version)

To make a call, the caller lifts receiver. The caller gets a dial tone and the caller dials digit (x). The dial tone ends. The caller completes dialing the number. The callee phone begins ringing at the same time a ringing begins in caller phone. When the callee answers the called phone stops ringing and ringing ends in caller phone. The phones are now connected. The caller hangs up and the phones are disconnected. The callee hangs up.



Partial Class Diagram

Line

Phone

Connected by

1

1

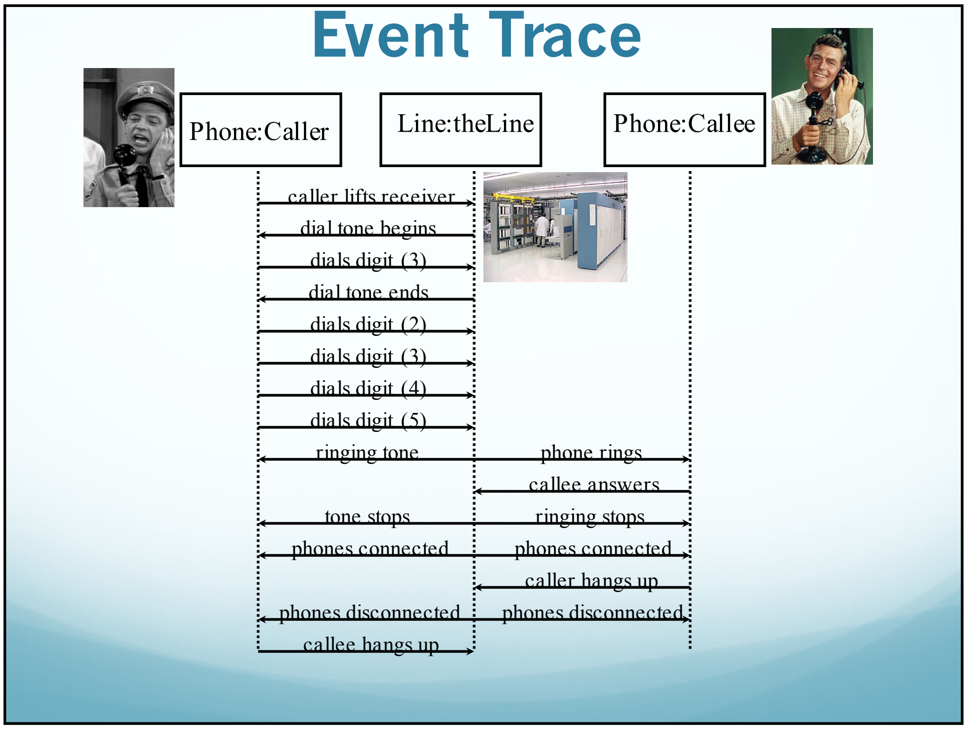
Caller

Connected by

1

1

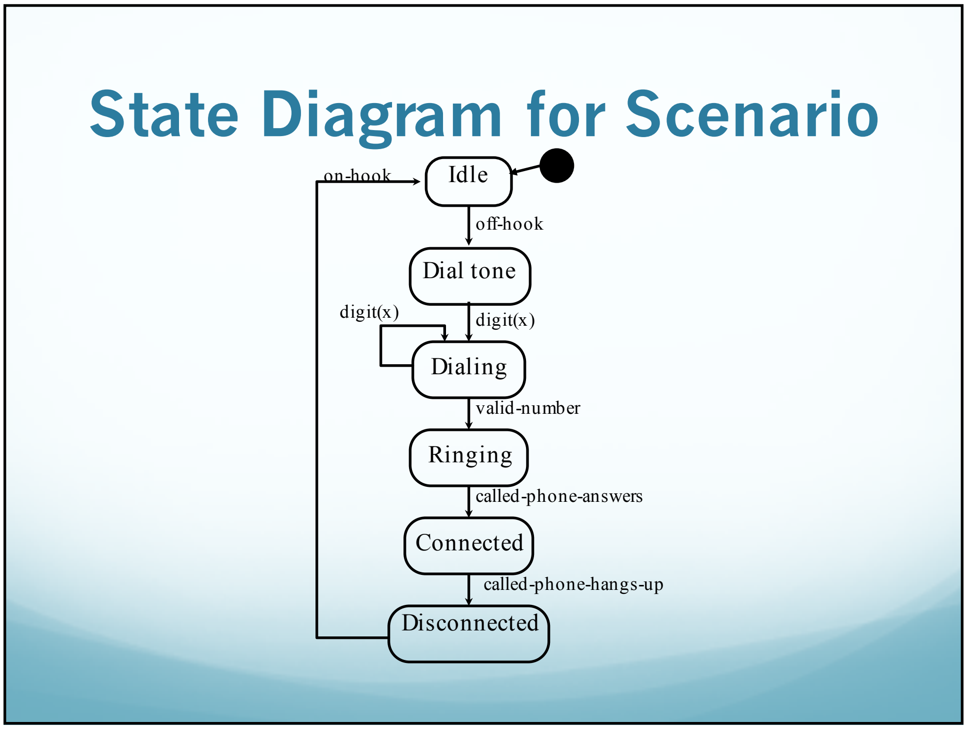
Callee



Event Trace

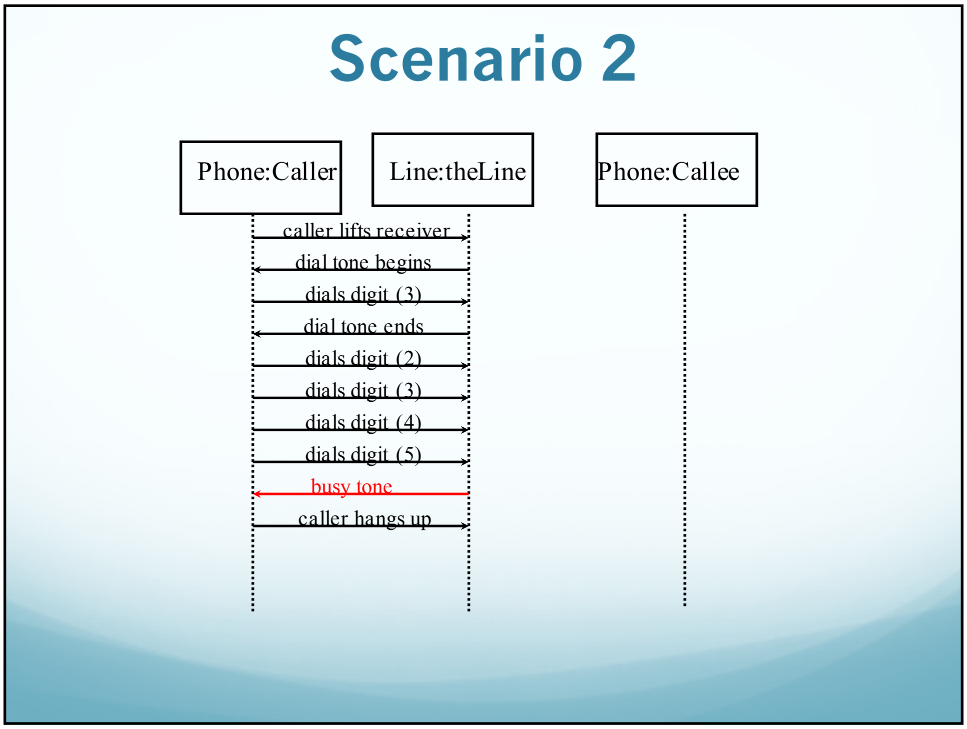
Phone:Caller
Line:theLine
Phone:Callee

caller lifts receiver
dial tone begins
dials digit (3)
dial tone ends
dials digit (2)
dials digit (3)
dials digit (4)
dials digit (5)
ringing tone
phone rings
callee answers
tone stops
ringing stops
phones connected
phones connected
caller hangs up
phones disconnected
phones disconnected
callee hangs up



State Diagram for Scenario

Idle
off-hook
Dial tone
digit(x)
digit(x)
Dialing
valid-number
Ringing
called-phone-answers
Connected
called-phone-hangs-up
Disconnected
on-hook



Scenario 2

Phone:Caller

Line:theLine

Phone:Callee

caller lifts receiver

dial tone begins

dials digit (3)

dial tone ends

dials digit (2)

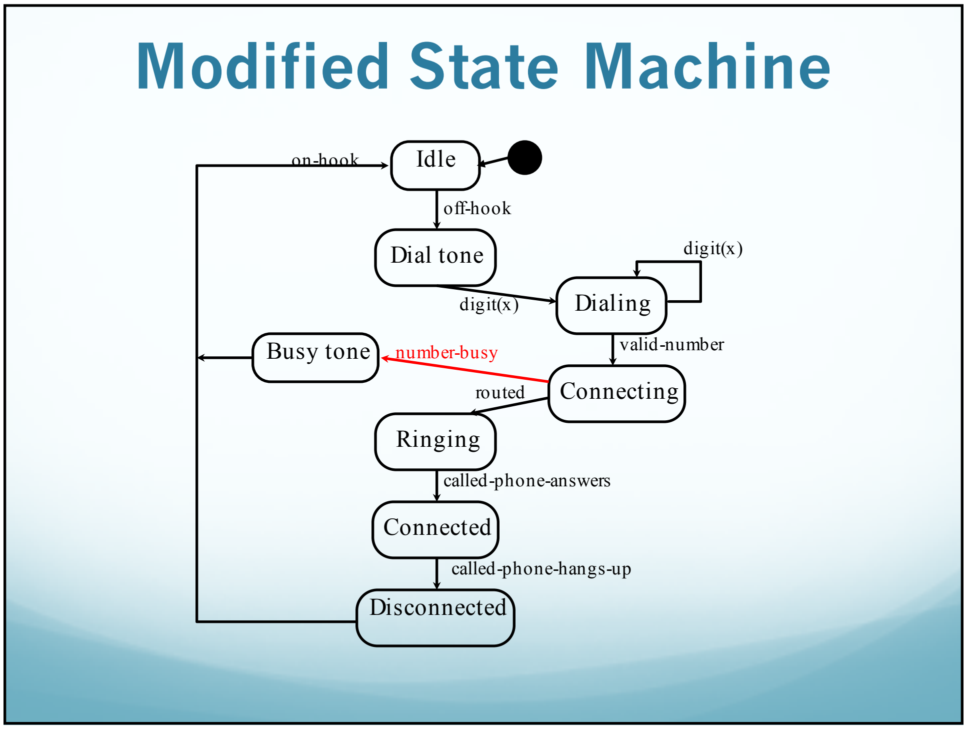
dials digit (3)

dials digit (4)

dials digit (5)

busy tone

caller hangs up



Modified State Machine

Idle

on-hook

off-hook

Dial tone

digit(x)

Dialing

digit(x)

valid-number

Busy tone

number-busy

Connecting

routed

Ringing

called-phone-answers

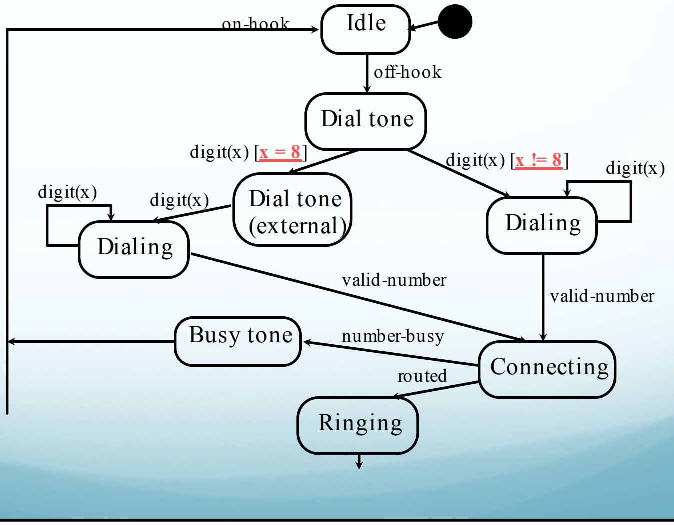
Connected

called-phone-hangs-up

Disconnected

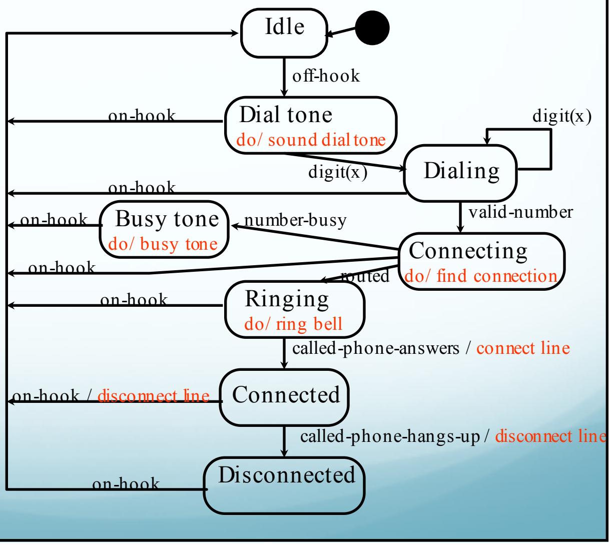
Conditions

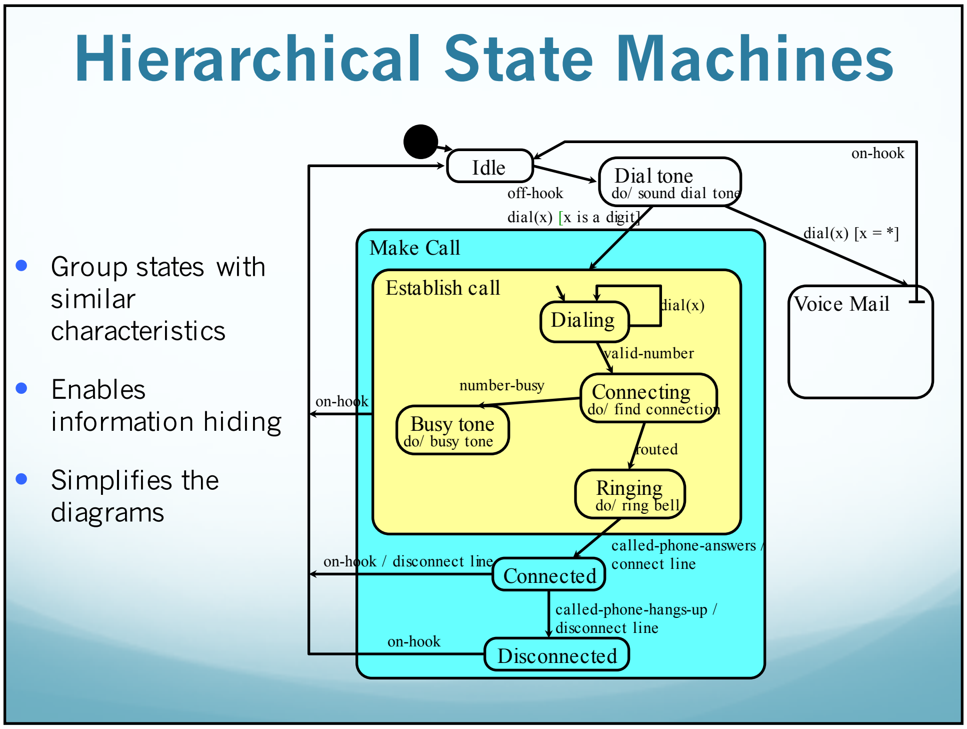
- **Sometimes the state transitions are conditional**



Operations (AKA Actions)

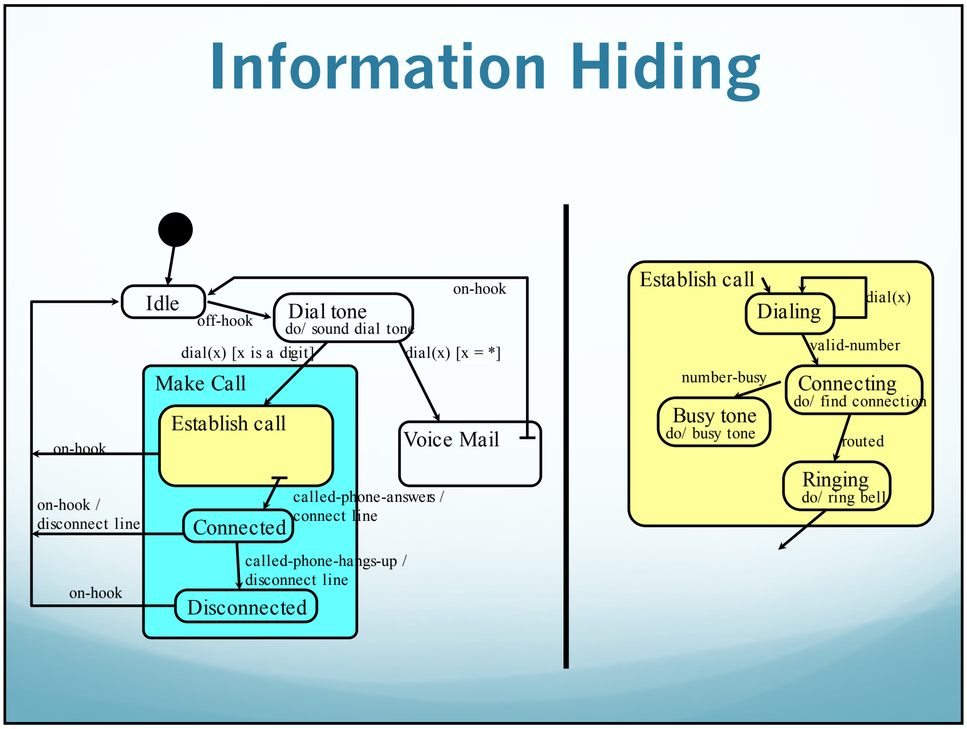
- Actions are performed when a transition is taken or performed while in a state
- Actions are terminated when leaving the state



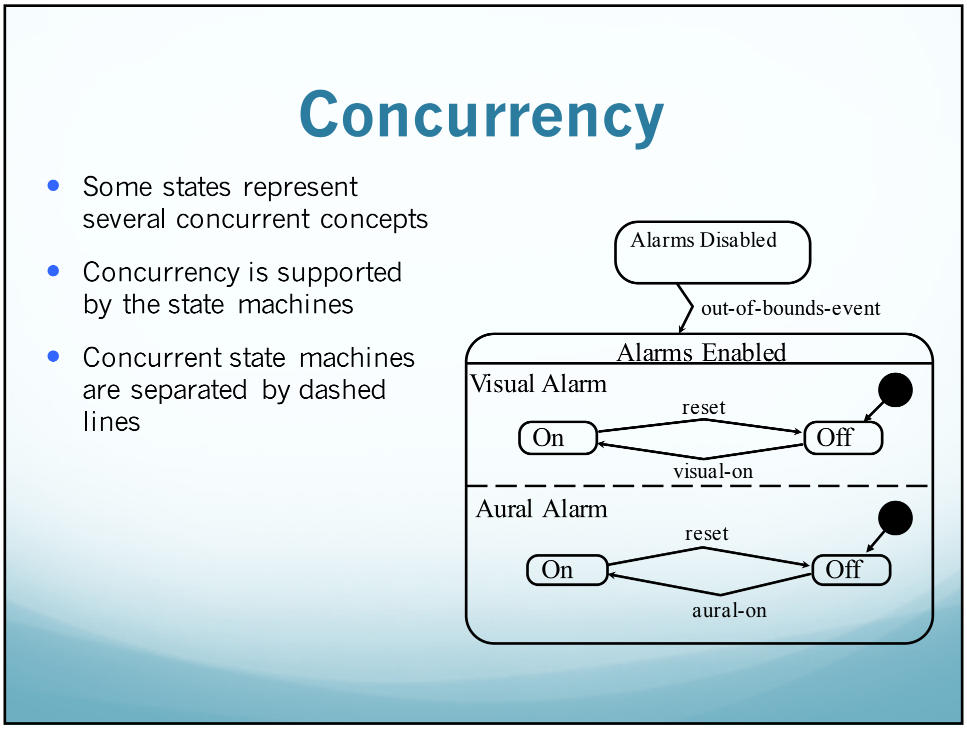


Hierarchical State Machines

- Group states with similar characteristics
- Enables information hiding
- Simplifies the diagrams

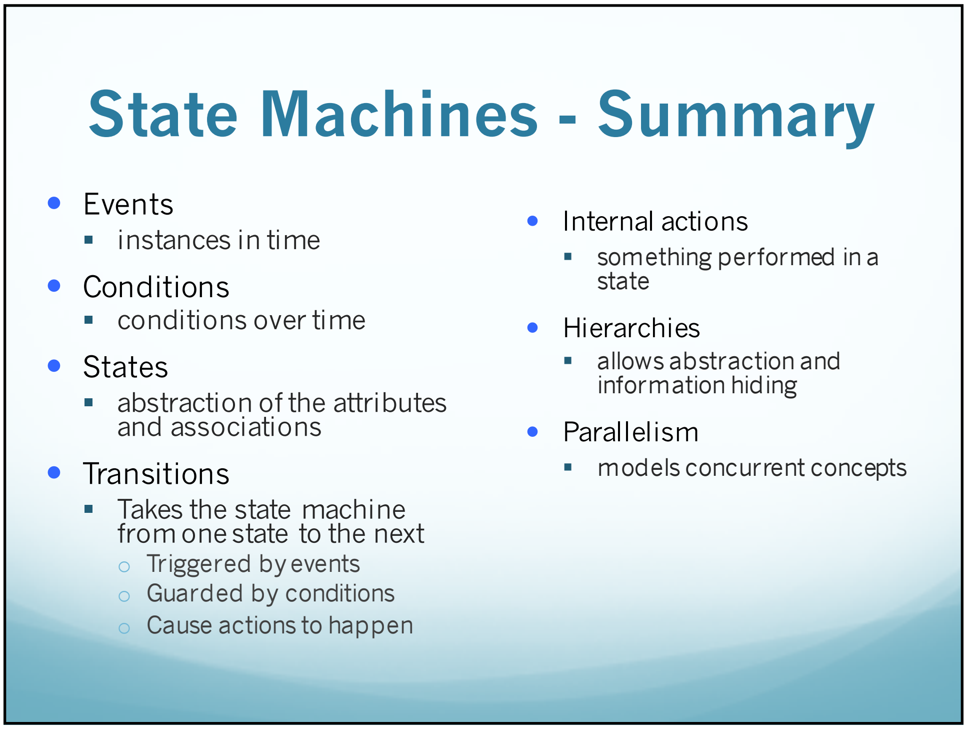


Information Hiding



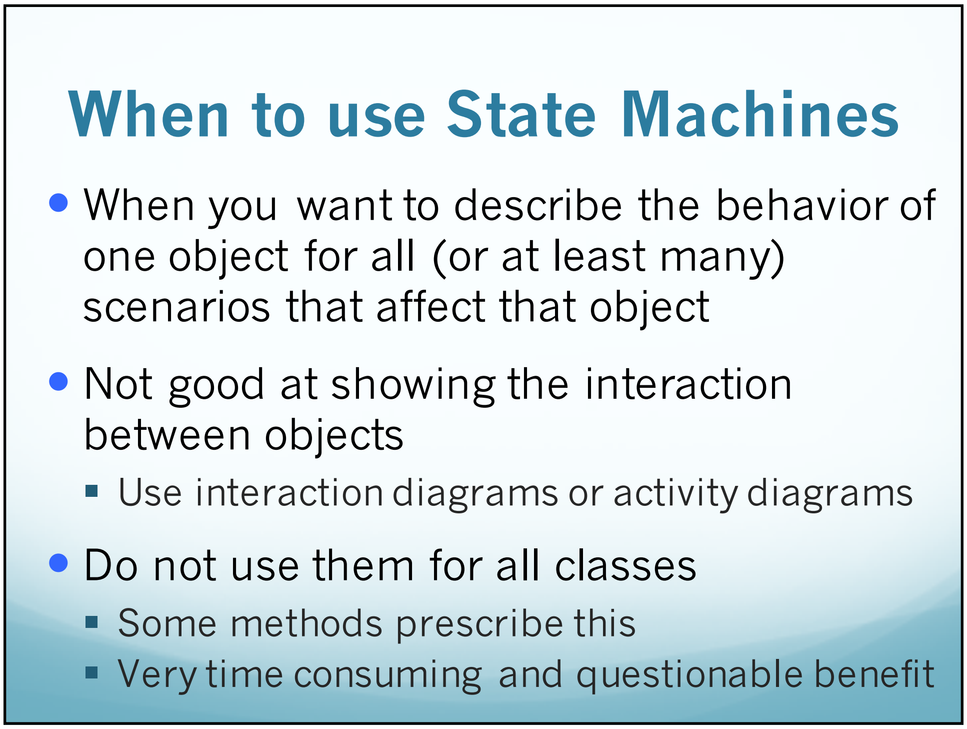
Concurrency

- Some states represent several concurrent concepts
- Concurrency is supported by the state machines
- Concurrent state machines are separated by dashed lines



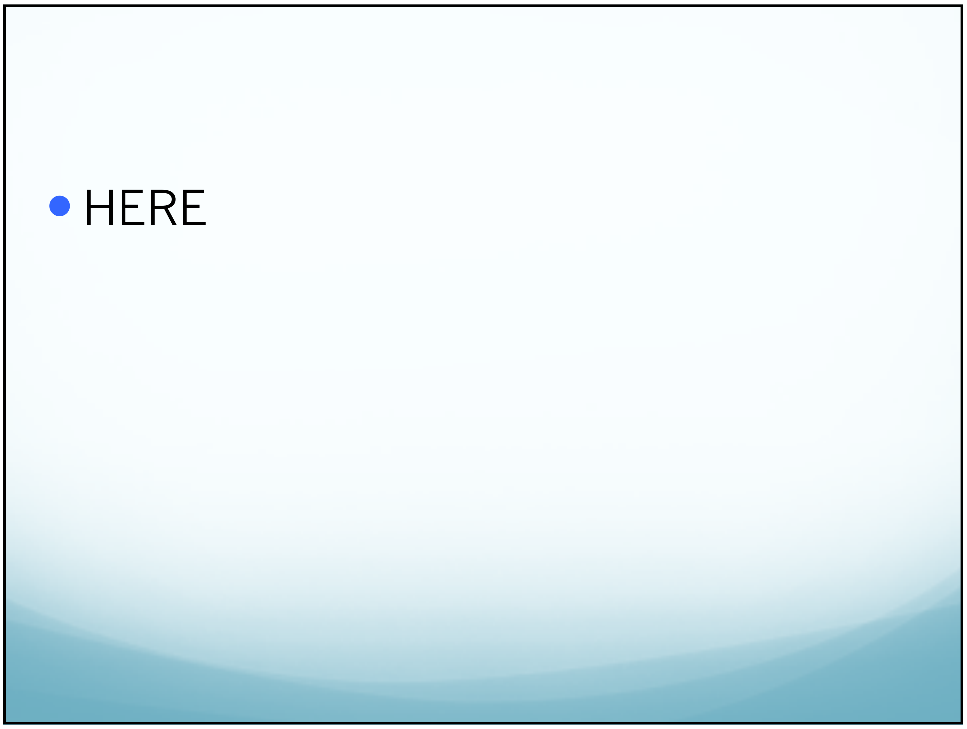
State Machines - Summary

- Events
  - instances in time
- Conditions
  - conditions over time
- States
  - abstraction of the attributes and associations
- Transitions
  - Takes the state machine from one state to the next
    - Triggered by events
    - Guarded by conditions
    - Cause actions to happen
- Internal actions
  - something performed in a state
- Hierarchies
  - allows abstraction and information hiding
- Parallelism
  - models concurrent concepts

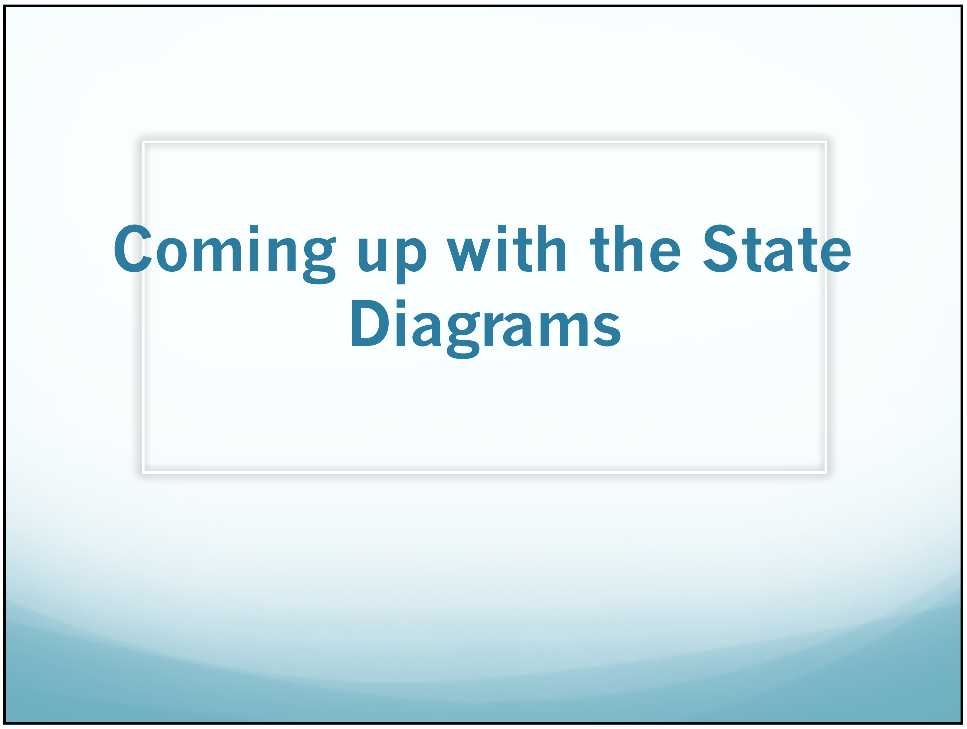


When to use State Machines

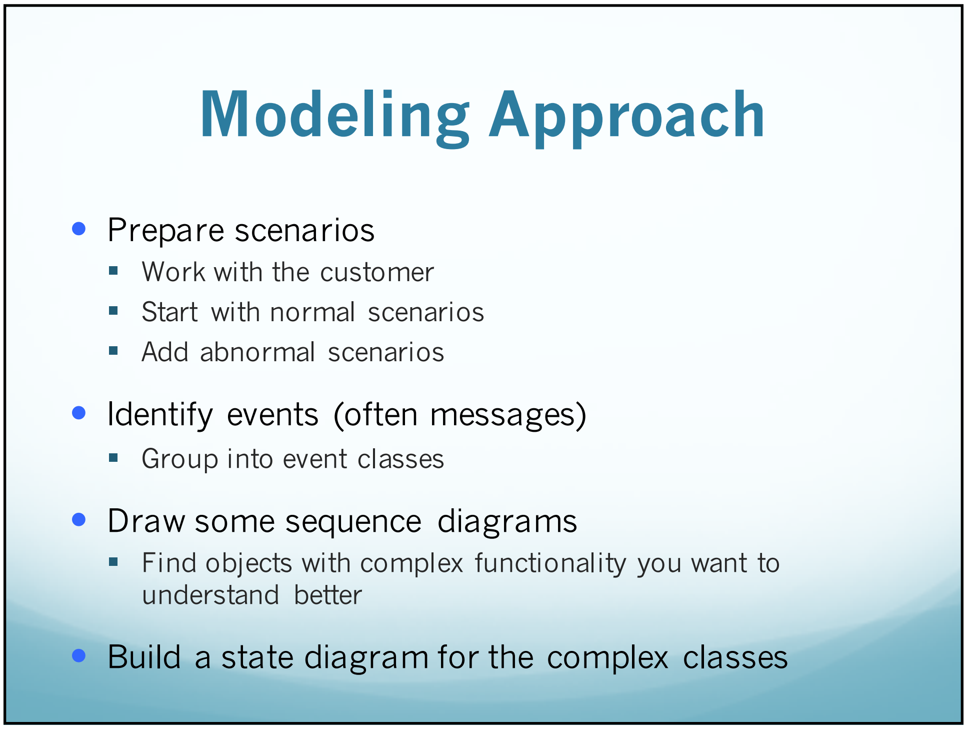
- When you want to describe the behavior of one object for all (or at least many) scenarios that affect that object
- Not good at showing the interaction between objects
  - Use interaction diagrams or activity diagrams
- Do not use them for all classes
  - Some methods prescribe this
  - Very time consuming and questionable benefit



- HERE



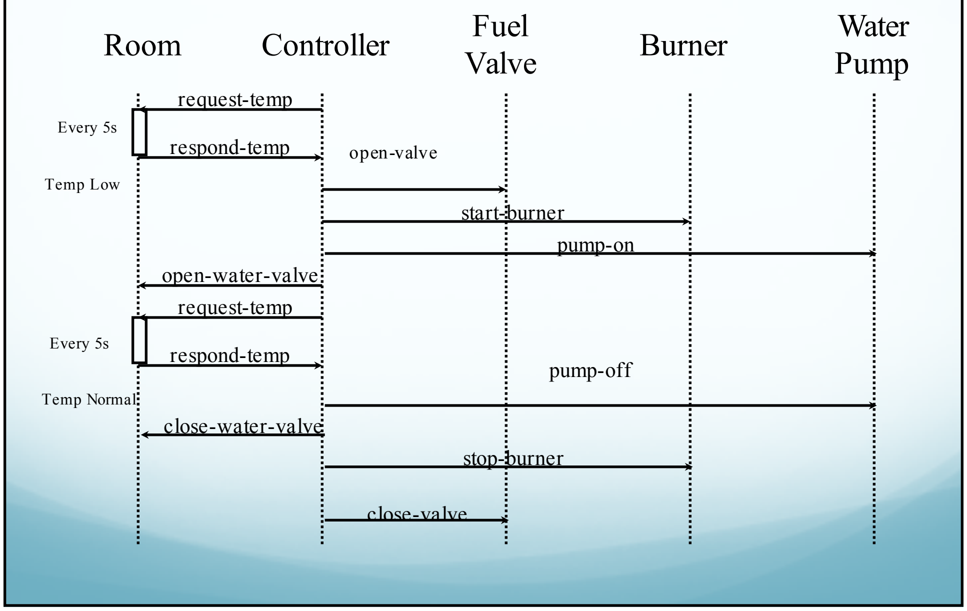
Coming up with the State Diagrams



Modeling Approach

- Prepare scenarios
  - Work with the customer
  - Start with normal scenarios
  - Add abnormal scenarios
- Identify events (often messages)
  - Group into event classes
- Draw some sequence diagrams
  - Find objects with complex functionality you want to understand better
- Build a state diagram for the complex classes

Scenario-1



Room

Controller

Fuel Valve

Burner

Water Pump

request-temp

Every 5s

respond-temp

open-valve

Temp Low

start-burner

pump-on

open-water-valve

request-temp

Every 5s

respond-temp

pump-off

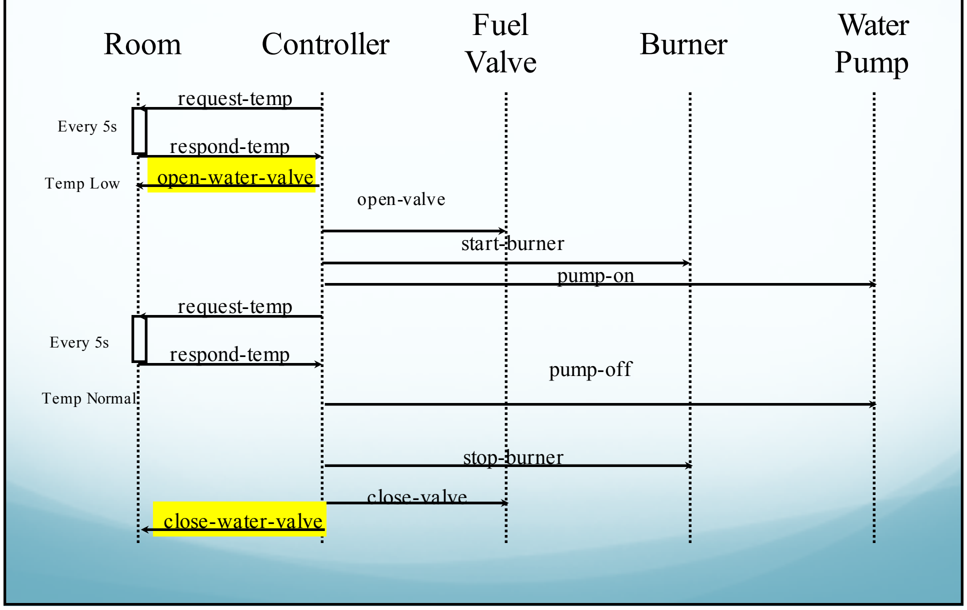
Temp Normal

close-water-valve

stop-burner

close-valve

Scenario-1, v2



Room

Controller

Fuel Valve

Burner

Water Pump

request-temp

Every 5s

respond-temp

open-water-valve

Temp Low

open-valve

start-burner

pump-on

request-temp

Every 5s

respond-temp

pump-off

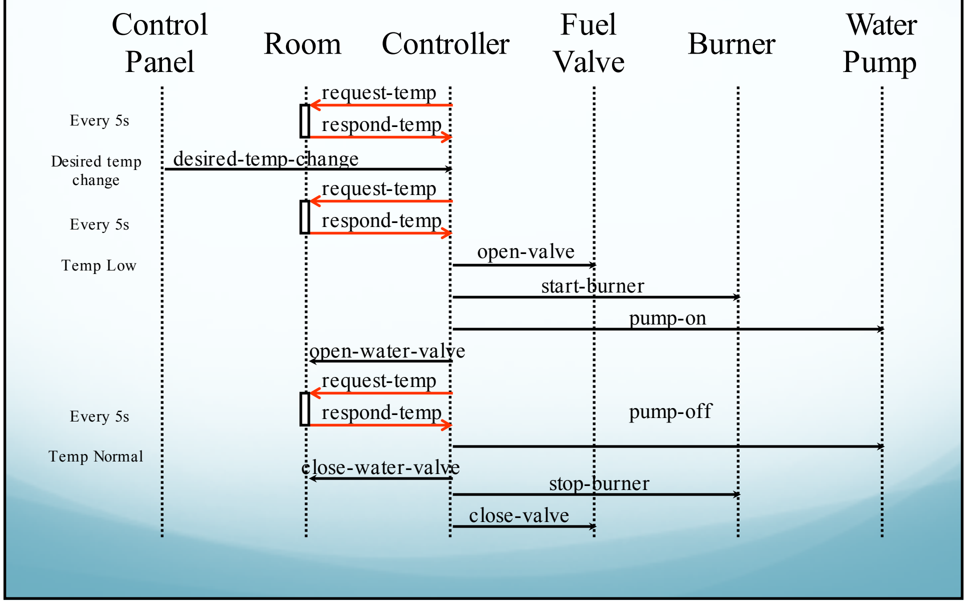
Temp Normal

stop-burner

close-valve

close-water-valve

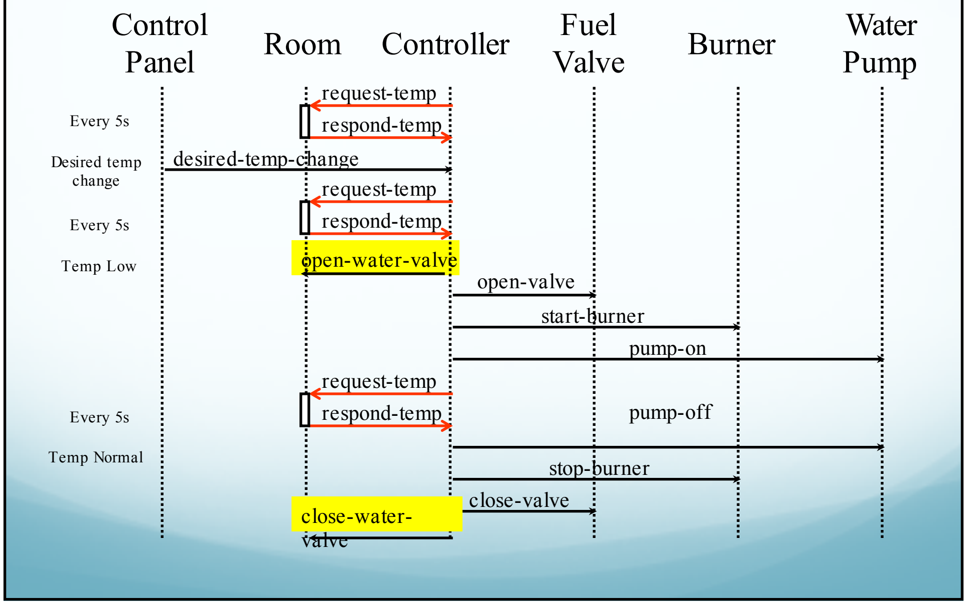
Scenario-2



Control Panel
Room
Controller
Fuel Valve
Burner
Water Pump

request-temp
Every 5s
respond-temp
Desired temp change
desired-temp-change
request-temp
Every 5s
respond-temp
Temp Low
open-valve
start-burner
pump-on
open-water-valve
request-temp
Every 5s
respond-temp
pump-off
Temp Normal
close-water-valve
stop-burner
close-valve

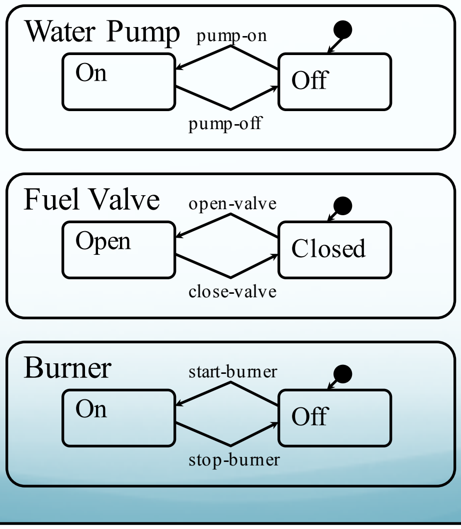
Scenario-2, v2



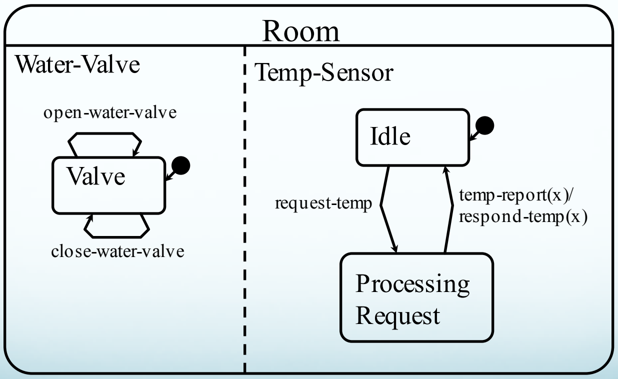
Control Panel
Room
Controller
Fuel Valve
Burner
Water Pump

request-temp
Every 5s
respond-temp
Desired temp change
desired-temp-change
request-temp
Every 5s
respond-temp
Temp Low
open-water-valve
open-valve
start-burner
pump-on
request-temp
Every 5s
respond-temp
pump-off
Temp Normal
stop-burner
close-valve
close-water-valve

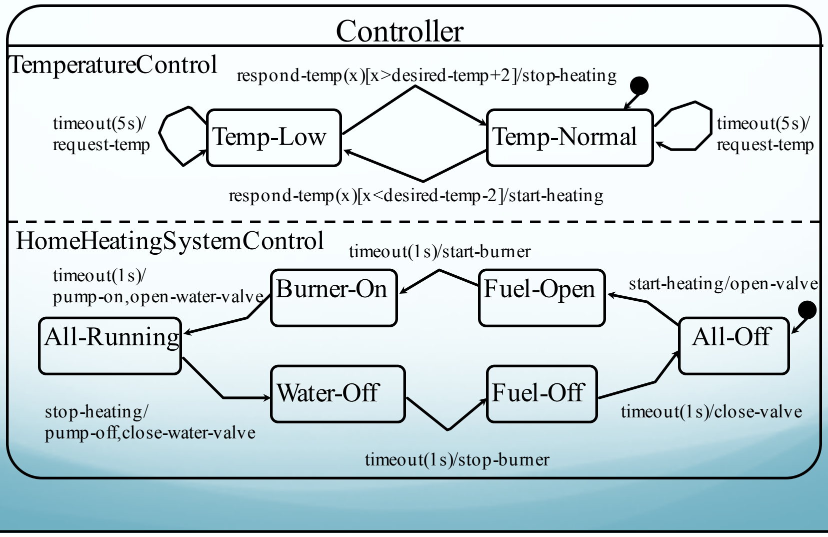
Dynamic Model



More Dynamic Model



Even More Dynamic Model



Controller

TemperatureControl

respond-temp(x)[x>desired-temp+2]/stop-heating

timeout(5s)/
request-temp

Temp-Low

Temp-Normal

timeout(5s)/
request-temp

respond-temp(x)[x<desired-temp-2]/start-heating

HomeHeatingSystemControl

timeout(1s)/start-burner

timeout(1s)/
pump-on,open-water-valve

Burner-On

Fuel-Open

start-heating/open-valve

All-Running

All-Off

Water-Off

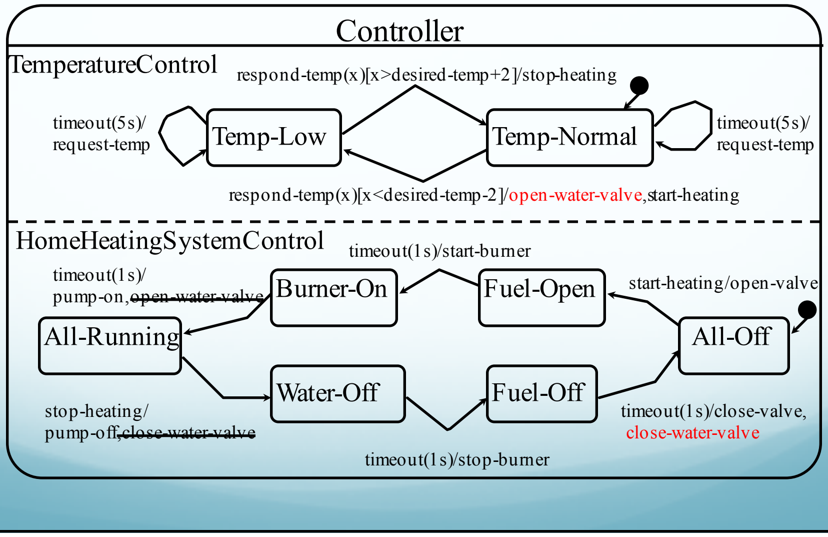
Fuel-Off

stop-heating/
pump-off,close-water-valve

timeout(1s)/close-valve

timeout(1s)/stop-burner

Even More Dynamic Model, v2



Controller

TemperatureControl

respond-temp(x)[x>desired-temp+2]/stop-heating

timeout(5s)/
request-temp

Temp-Low

Temp-Normal

timeout(5s)/
request-temp

respond-temp(x)[x<desired-temp-2]/open-water-valve,start-heating

HomeHeatingSystemControl

timeout(1s)/start-burner

timeout(1s)/
pump-on,~~open-water-valve~~

Burner-On

Fuel-Open

start-heating/open-valve

All-Running

All-Off

Water-Off

Fuel-Off

stop-heating/
pump-off,~~close-water-valve~~

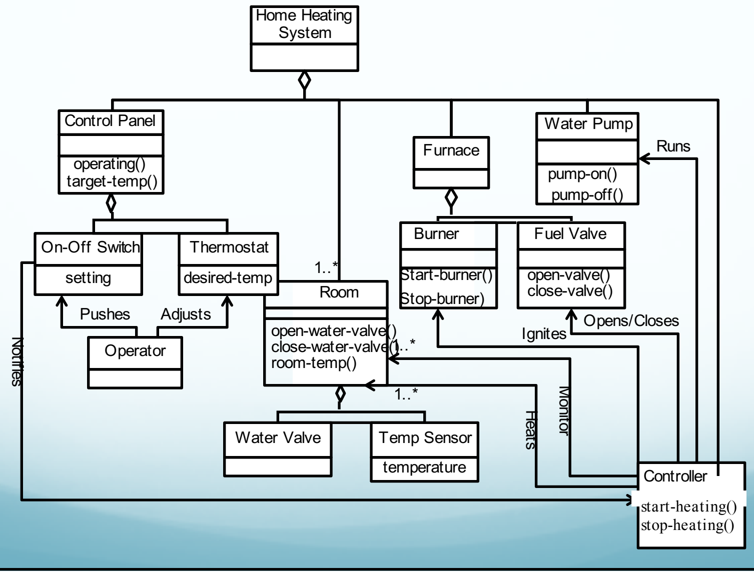
timeout(1s)/close-valve,
close-water-valve

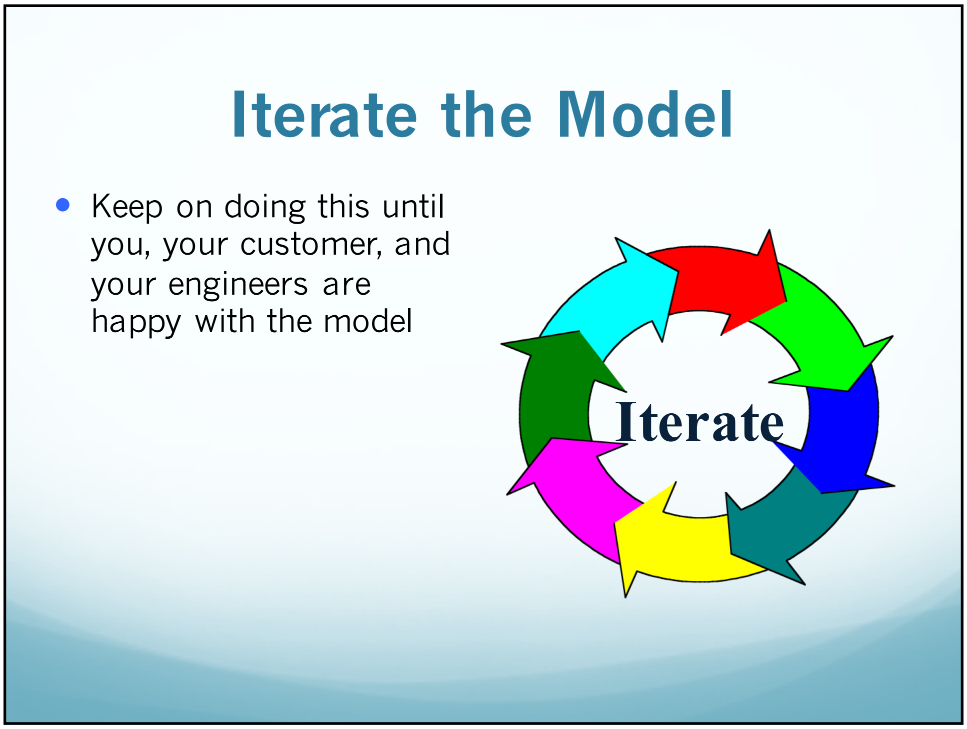
timeout(1s)/stop-burner

Identify Key Operations

- Operations from the object model
  - Accessing and setting attributes and associations (often not shown)
- Operations from events
  - All events represent some operation
- Operations from actions and activities
  - Actions and activities represent some processing activity within some object
- Operations from functions
  - Each function typically represent one or more operations
- Shopping list operations
  - Inherent operations (what should be there)

Complete OO Model

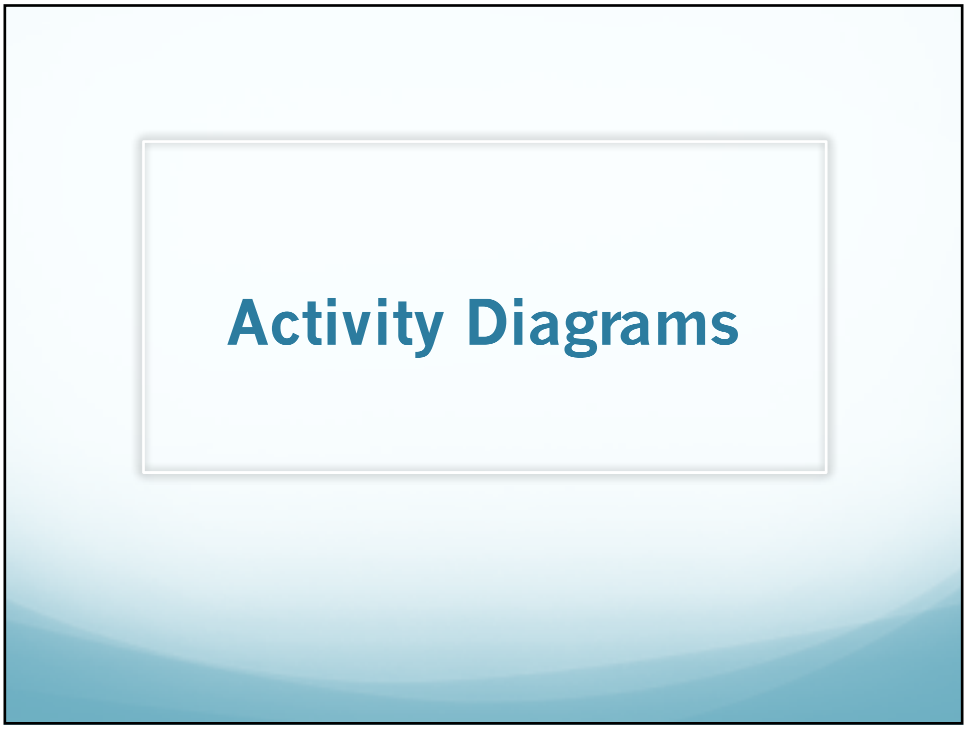




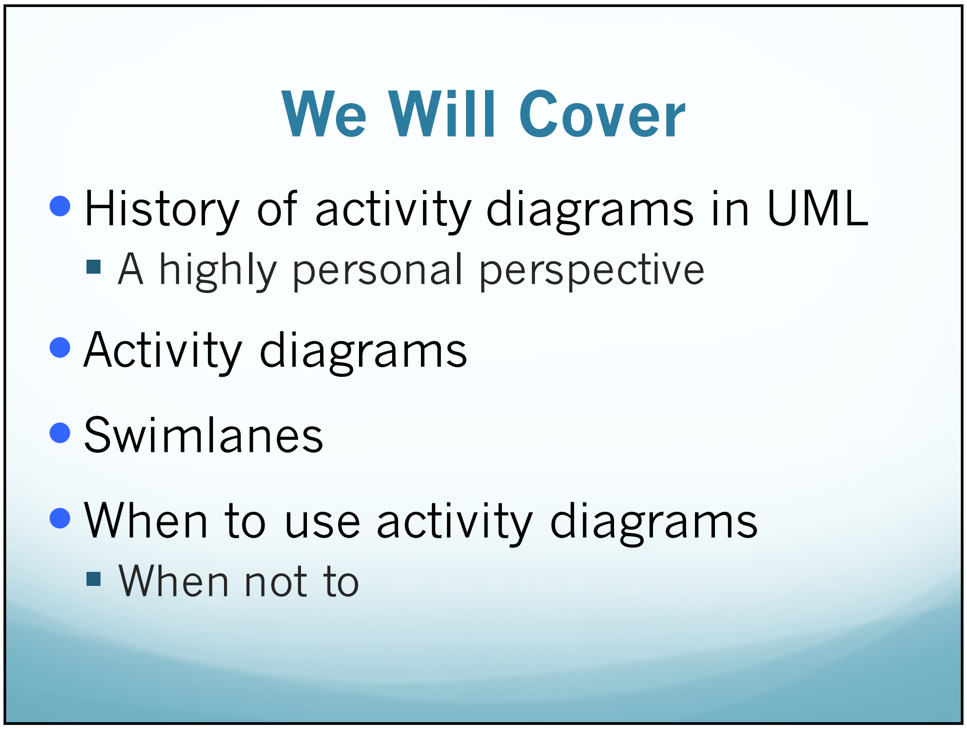
Iterate the Model

- Keep on doing this until you, your customer, and your engineers are happy with the model

Iterate

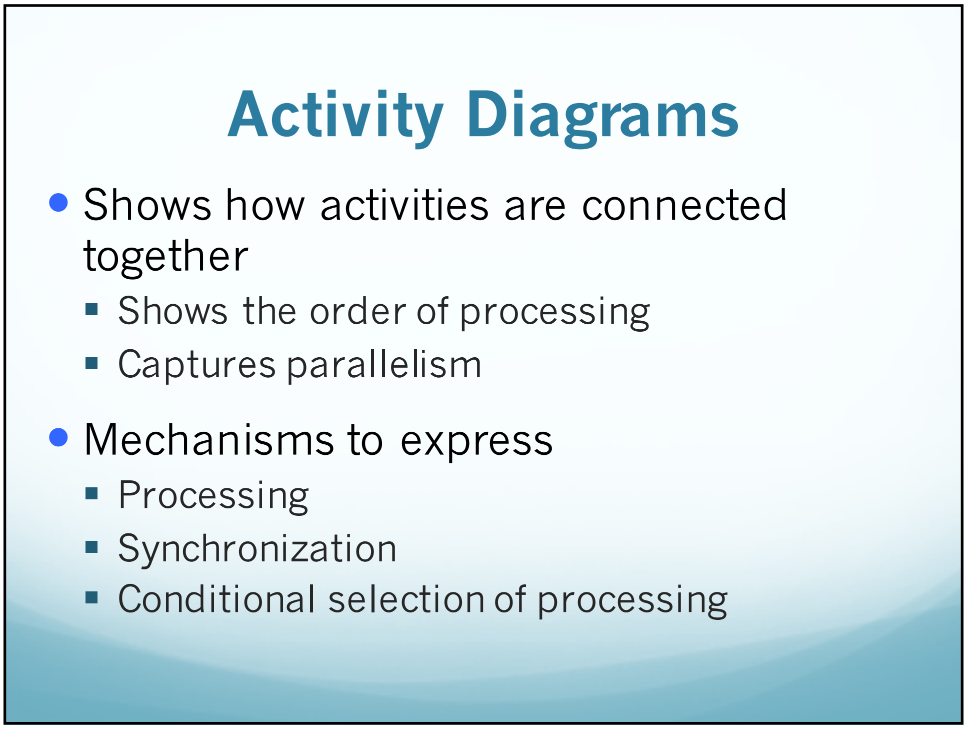


Activity Diagrams



We Will Cover

- History of activity diagrams in UML
  - A highly personal perspective
- Activity diagrams
- Swimlanes
- When to use activity diagrams
  - When not to



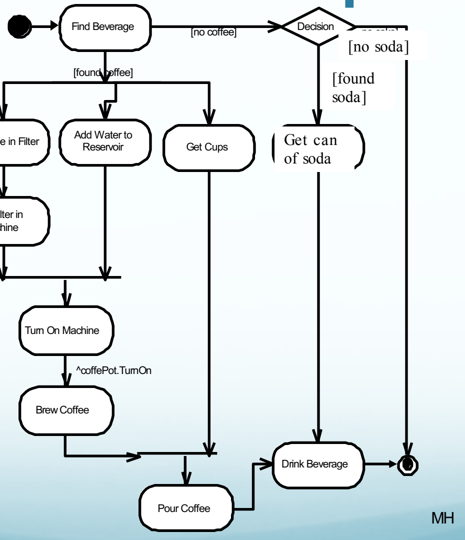
Activity Diagrams

- Shows how activities are connected together
  - Shows the order of processing
  - Captures parallelism
- Mechanisms to express
  - Processing
  - Synchronization
  - Conditional selection of processing

Why Activity Diagrams

- Very good question
  - Not part of any previous (UML related) method
  - To make UML more inclusive of business modeling needs
- Suitable for modeling of business activities
  - UML and OO is becoming more prevalent in business applications
  - Object frameworks are making an inroad
  - Stay within one development approach and notation
  - Notation similar to process modeling languages

Coffee Example



MH

HACS Use-Cases

Use case: **Distribute Assignments**

Actors: Instructor (initiator), Student

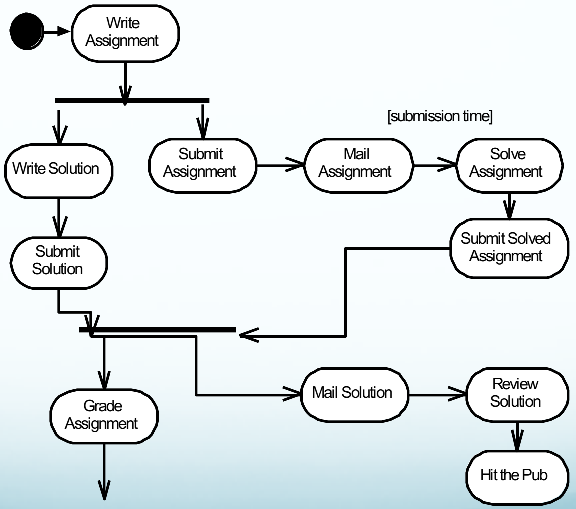
Type: Primary and essential

Description: The Instructor completes an assignment and submits it to the system. The instructor will also submit the delivery date, due date, and the class the assignment is assigned for. The system will at the due date mail the assignment to the student.

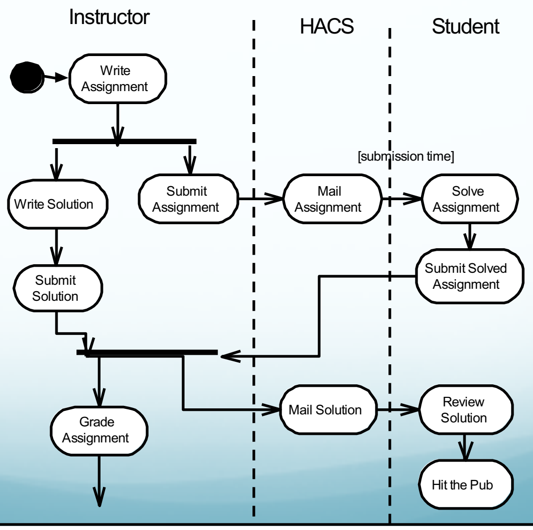
Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: *Configure HACS* must be done before any user (Instructor or Student) can use HACS

Activity Diagrams for Use Cases

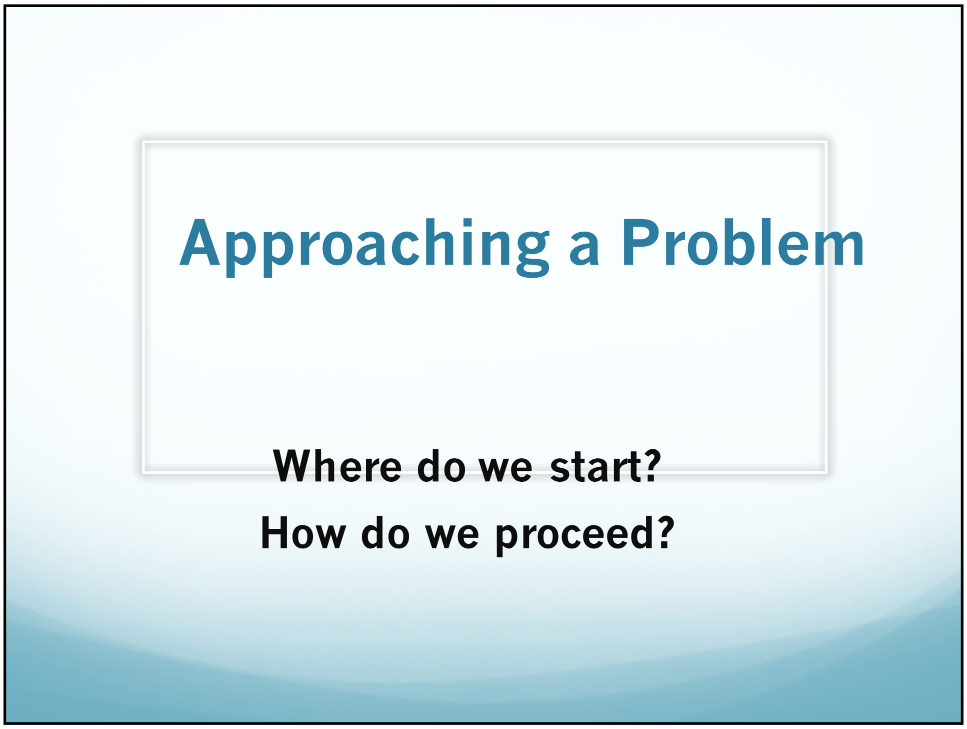


Swimlanes (Who Does What?)



When to Use Activity Diagrams

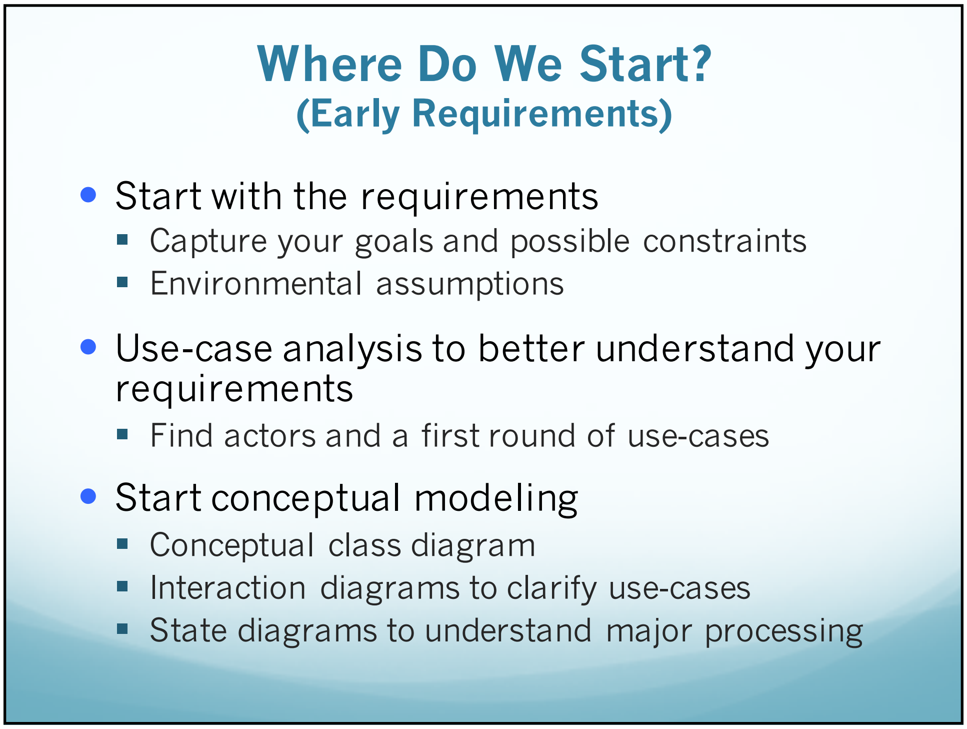
- Useful when
  - Analyzing a use case (or collection of use cases)
  - Understanding workflow in an organization
  - Working with multi-threaded applications
    - For instance, process control applications
  - Do not use activity diagrams
    - To figure out how objects collaborate
    - See how objects behave over time



Approaching a Problem

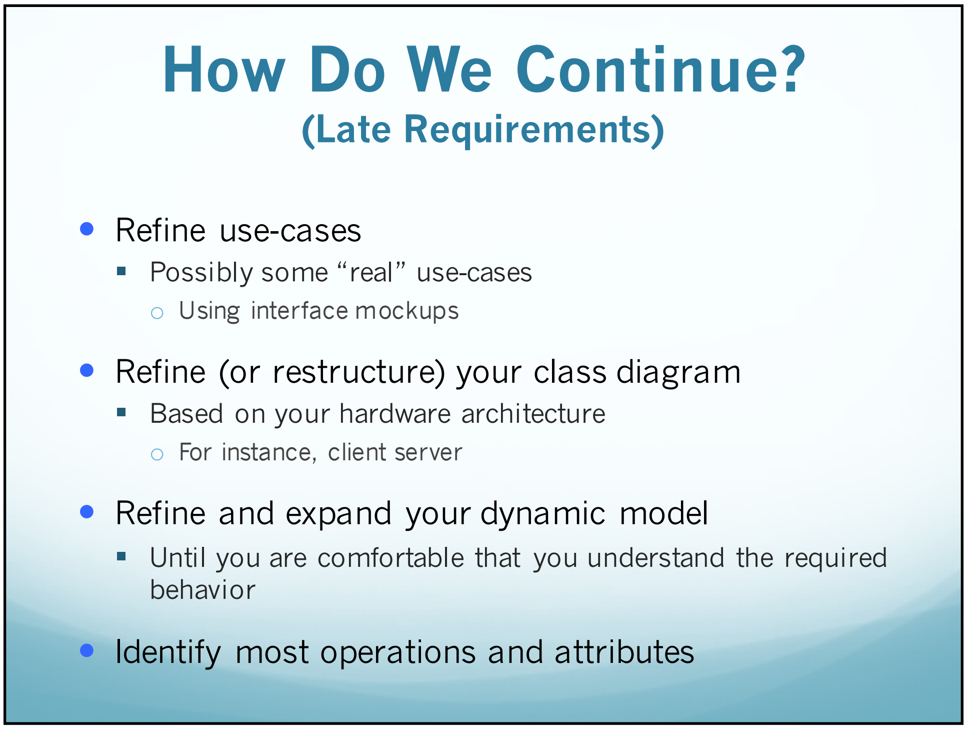
Where do we start?

How do we proceed?



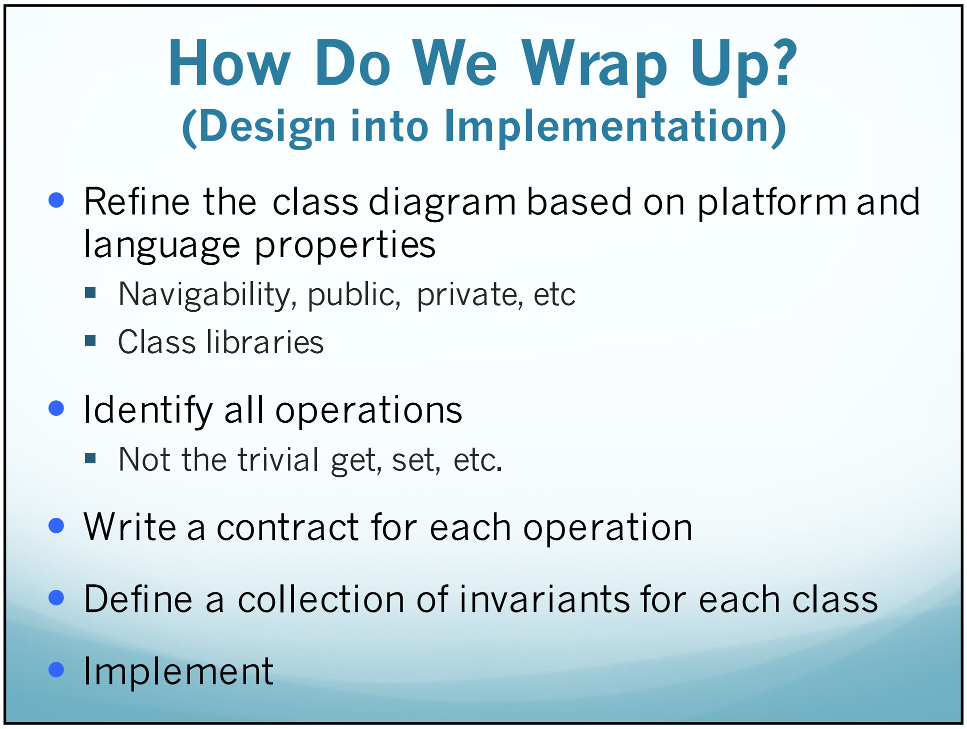
Where Do We Start?
(Early Requirements)

- Start with the requirements
  - Capture your goals and possible constraints
  - Environmental assumptions
- Use-case analysis to better understand your requirements
  - Find actors and a first round of use-cases
- Start conceptual modeling
  - Conceptual class diagram
  - Interaction diagrams to clarify use-cases
  - State diagrams to understand major processing



How Do We Continue?
(Late Requirements)

- Refine use-cases
  - Possibly some “real” use-cases
    - Using interface mockups
- Refine (or restructure) your class diagram
  - Based on your hardware architecture
    - For instance, client server
- Refine and expand your dynamic model
  - Until you are comfortable that you understand the required behavior
- Identify most operations and attributes



How Do We Wrap Up?
(Design into Implementation)

- Refine the class diagram based on platform and language properties
  - Navigability, public, private, etc
  - Class libraries
- Identify all operations
  - Not the trivial get, set, etc.
- Write a contract for each operation
- Define a collection of invariants for each class
- Implement