


Object Modeling Approach

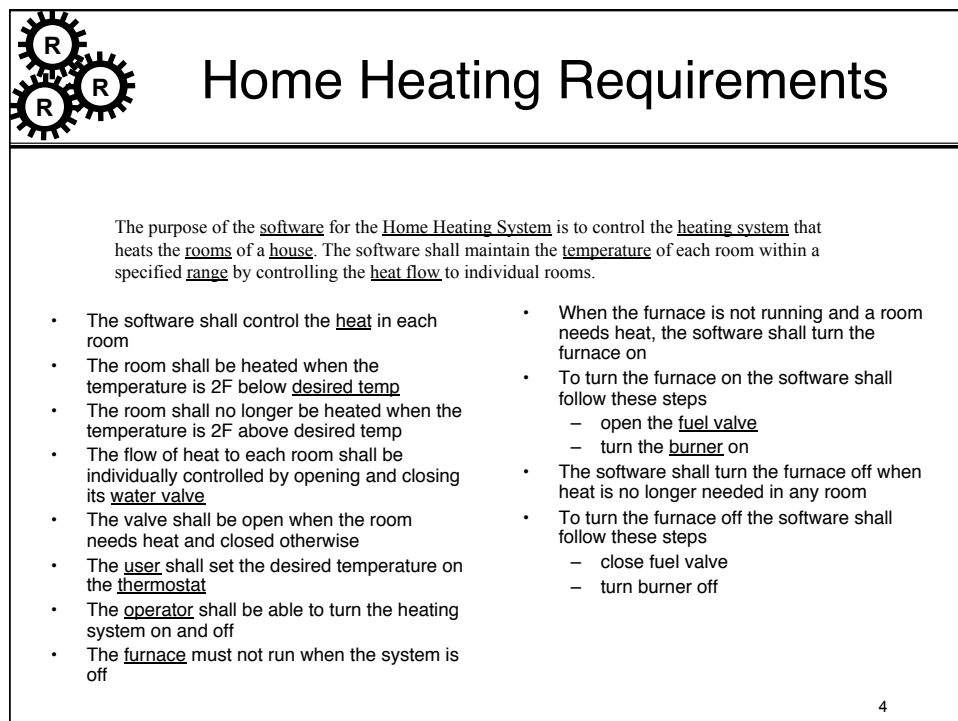
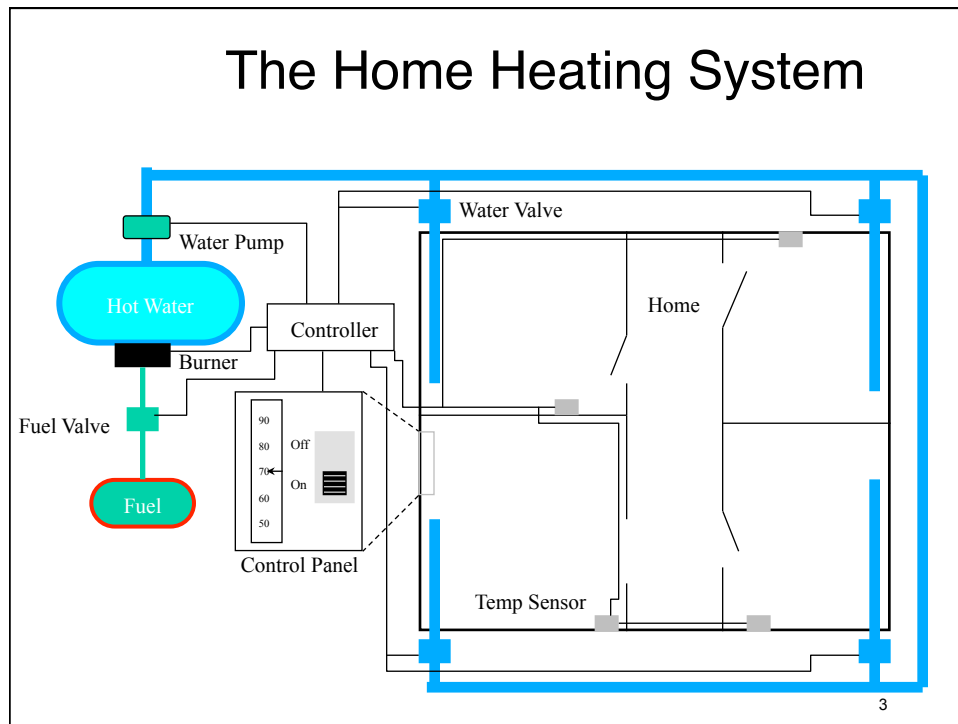
1

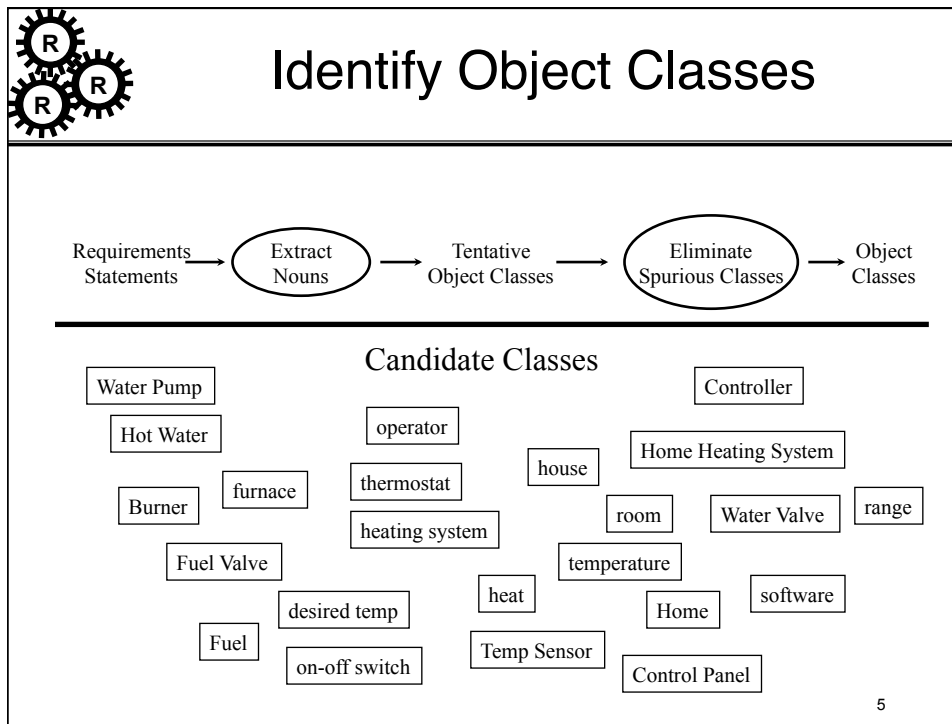


Object Modeling Approach

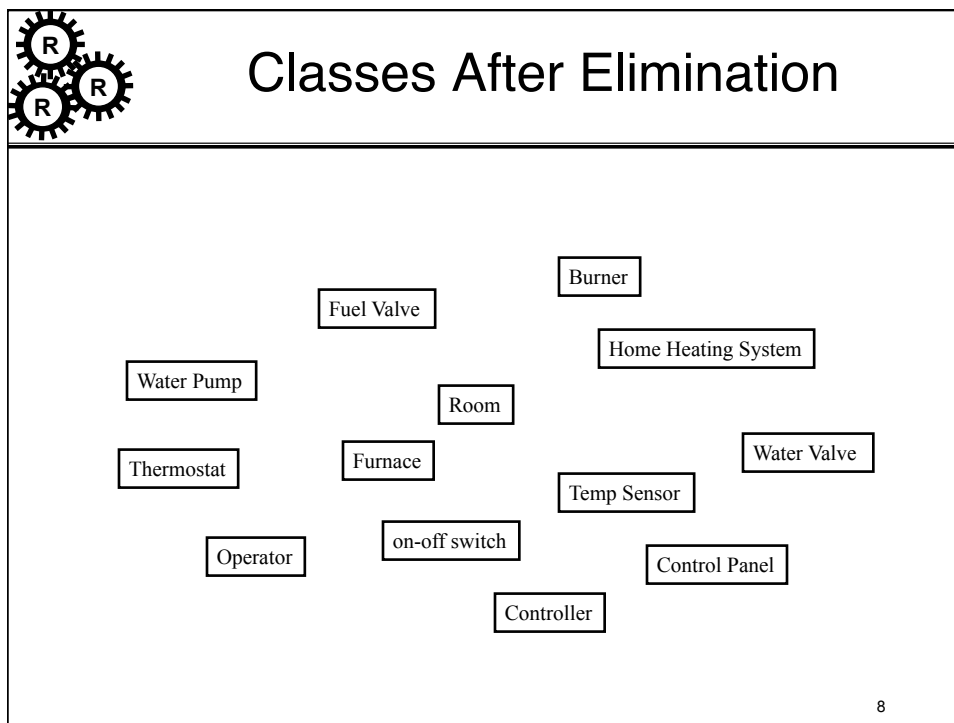
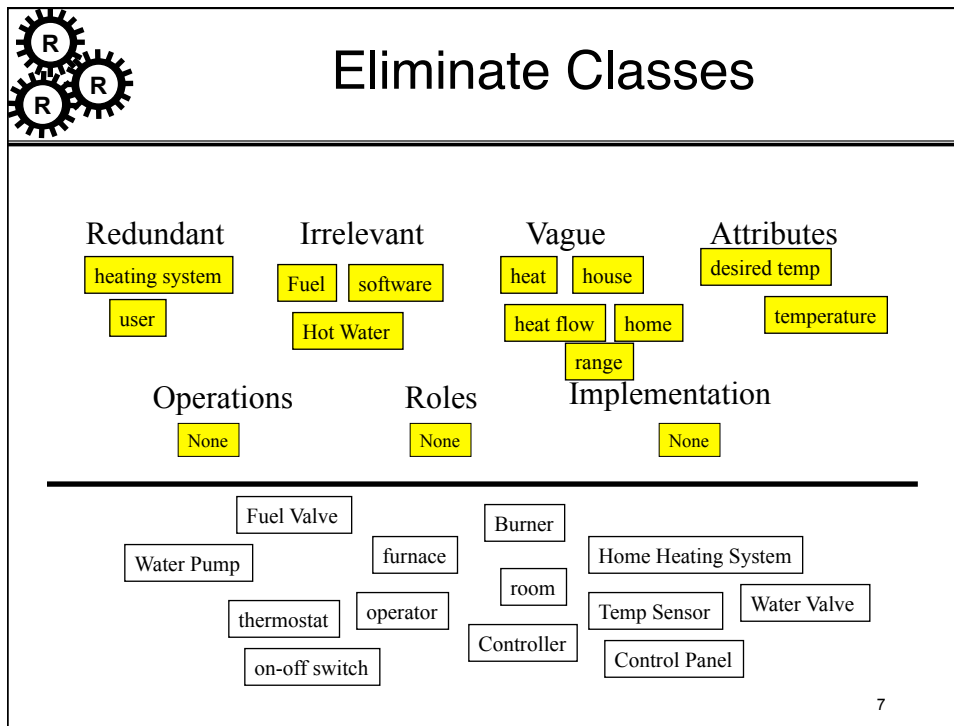
- Start with a problem statement
 - High-level requirements
- Define object model
 - Identify objects and classes
 - Prepare data dictionary
 - Identify associations and aggregations
 - Identify attributes of objects and links
 - Organize and simplify using inheritance
 - Iterate and refine the model
 - Group classes into modules

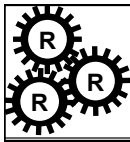
2





-
- 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
- 6

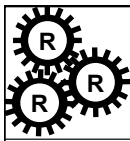




Prepare Data Dictionary

- 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

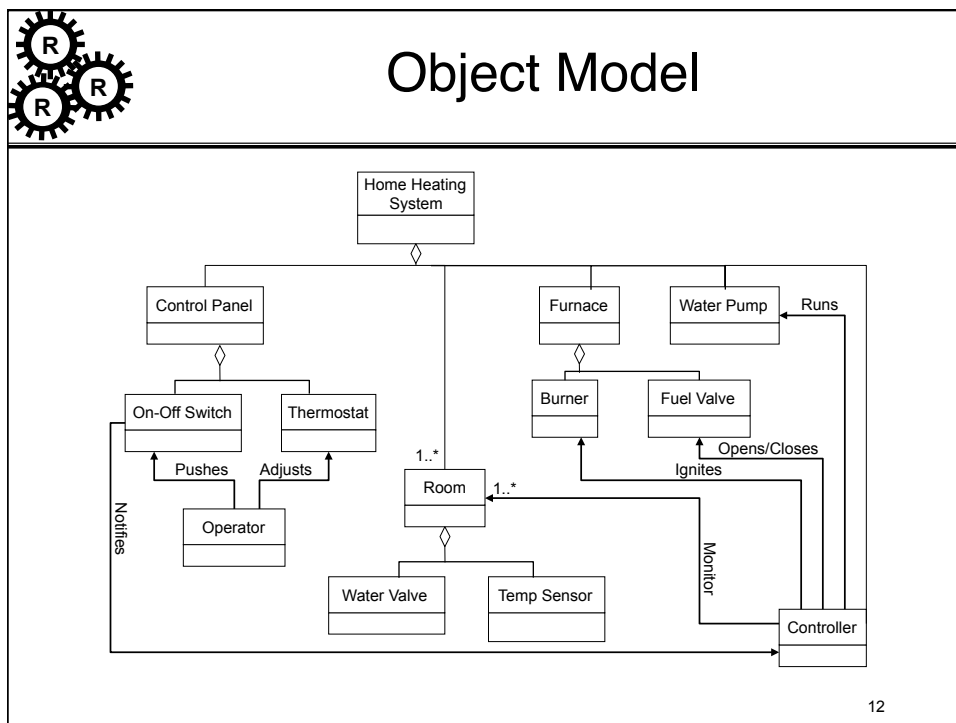
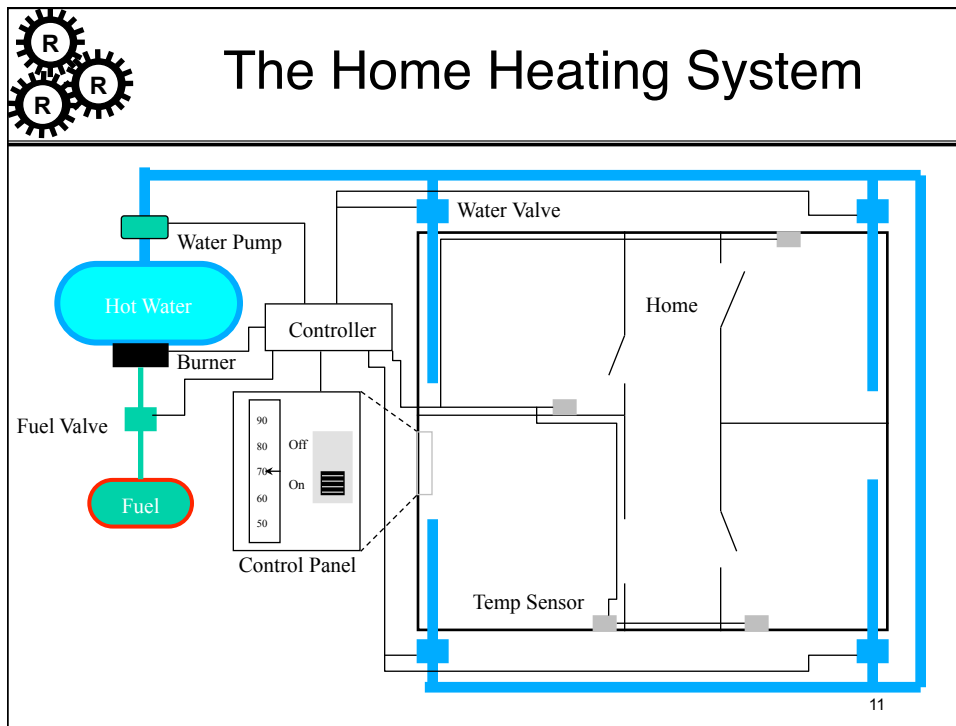
9

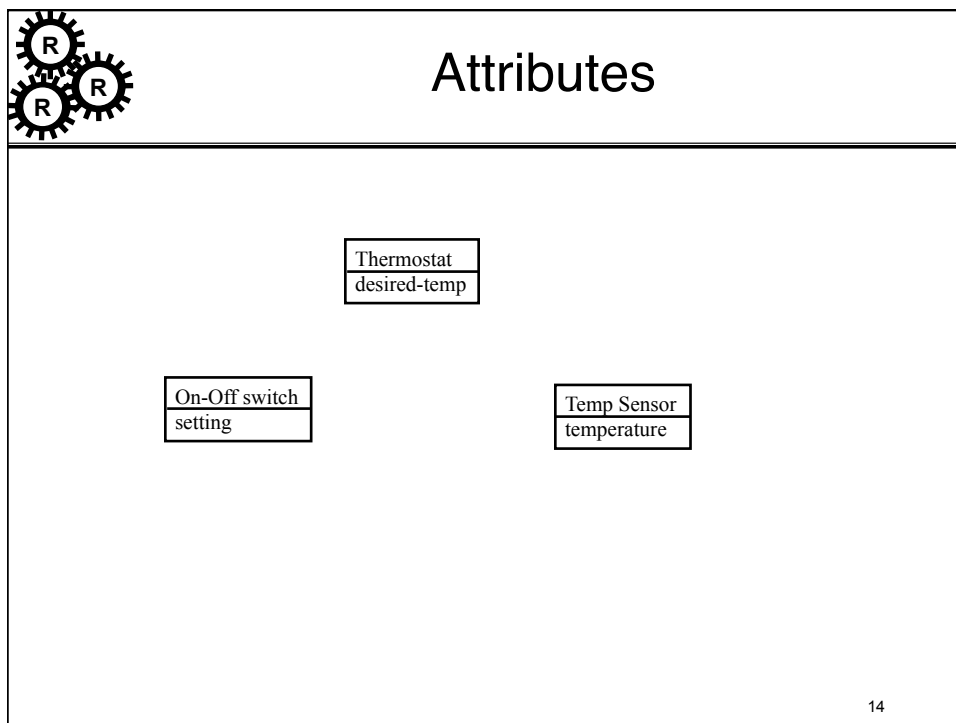
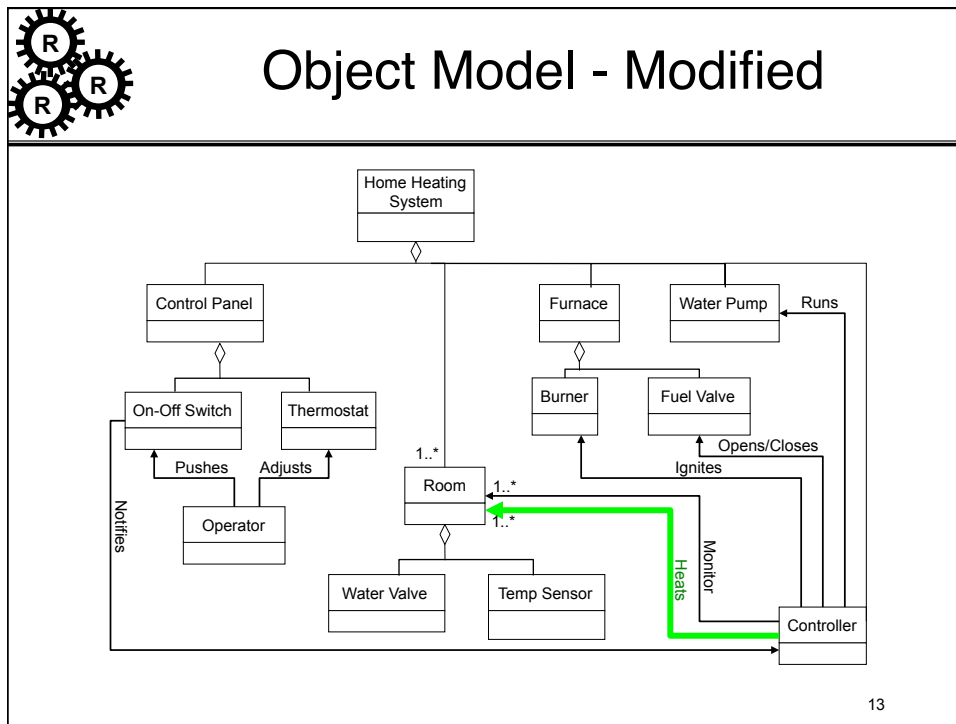


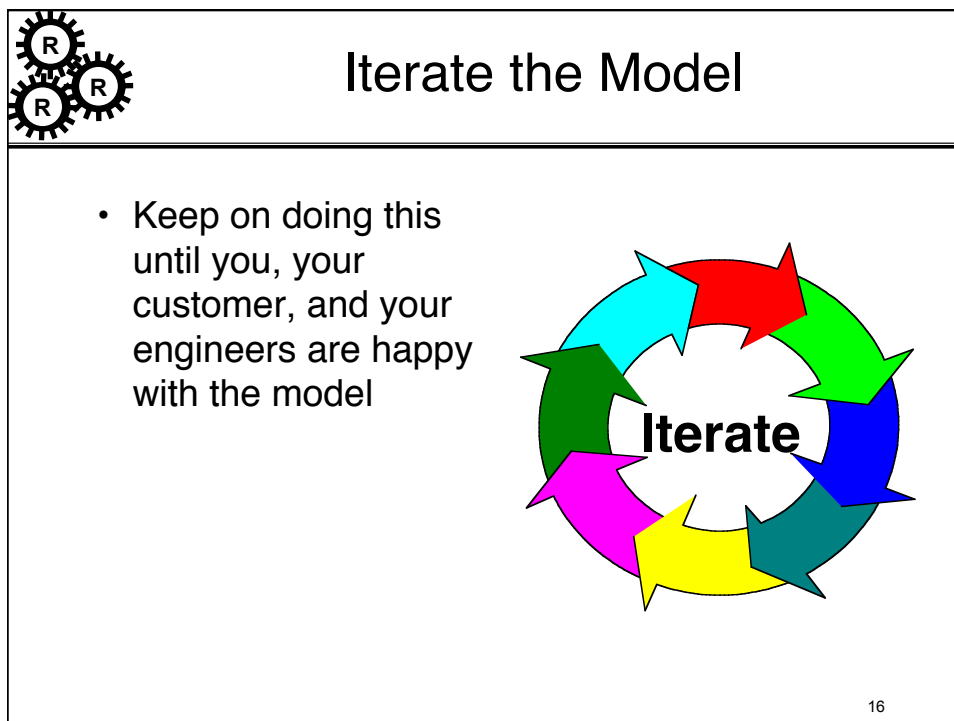
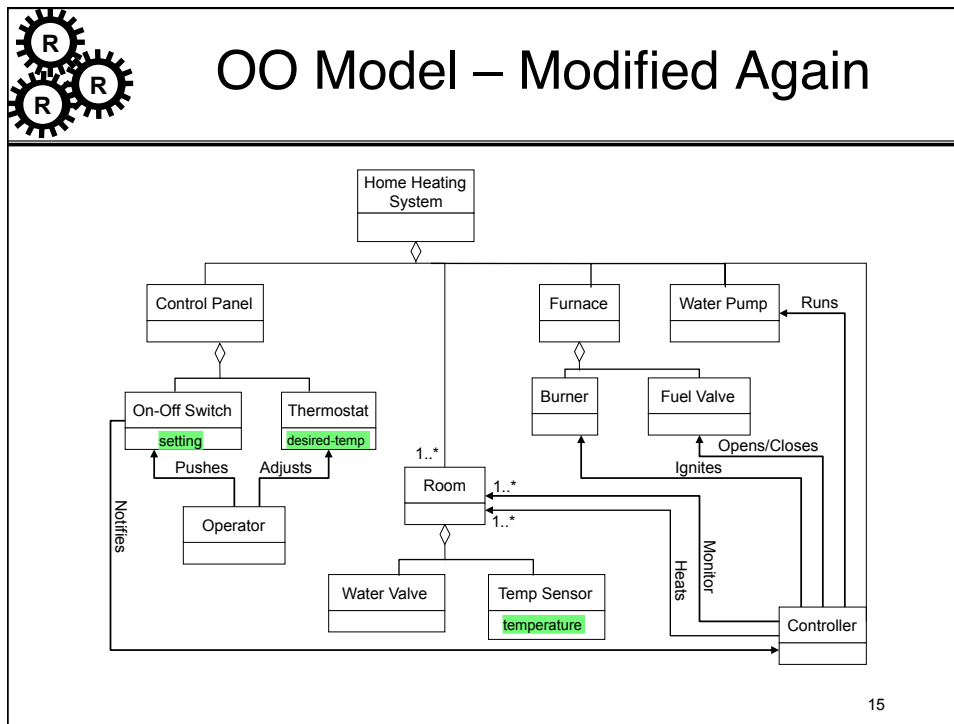
Possible Associations

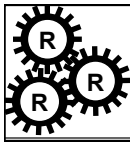
- Not much information from the prose requirements
- A lot of information from the system design

- 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





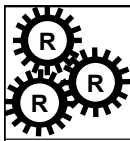




Operation vs Method

- **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**.

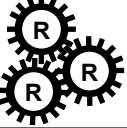
17



OO Using UML: Dynamic Models

Defining how the objects behave

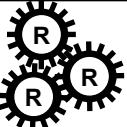
18



Overview

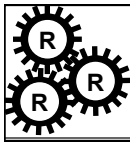
- The object model describes the structure of the system (objects, attributes, and operations)
- The dynamic model describes how the objects change state (how the attributes change) and in which order the state changes can take place
- Several models used to find the appropriate dynamic behavior
 - Interaction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

19



Interaction Diagrams

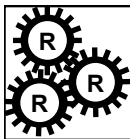
20



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


21



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

22



Home Heating Use-Case

Use case: Power Up

Actors: Home Owner (initiator)

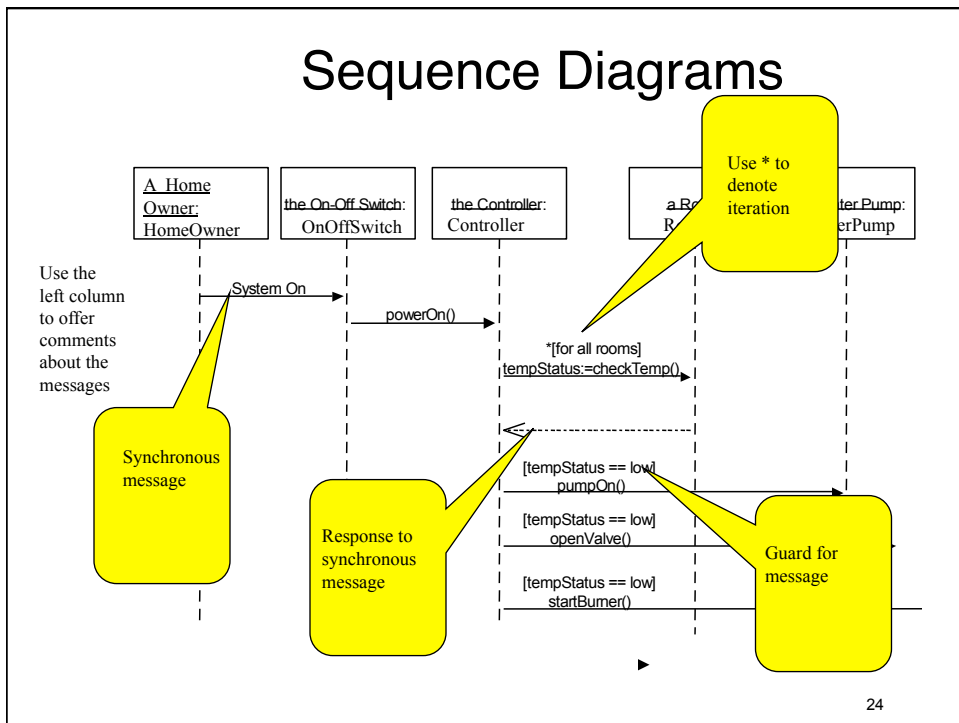
Type: Primary and essential

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.

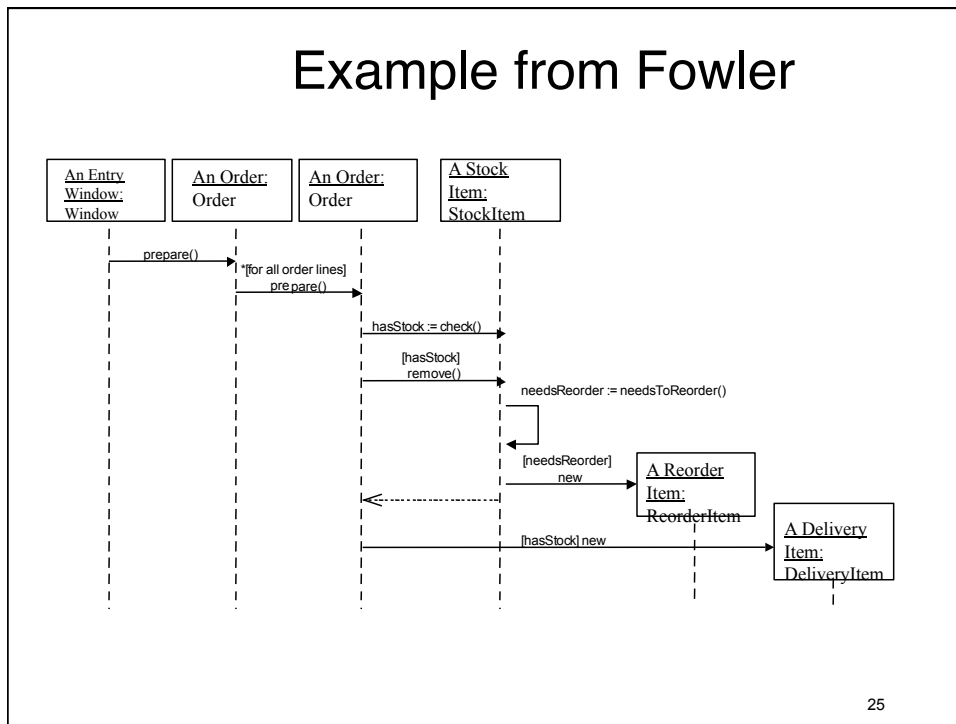
Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: None

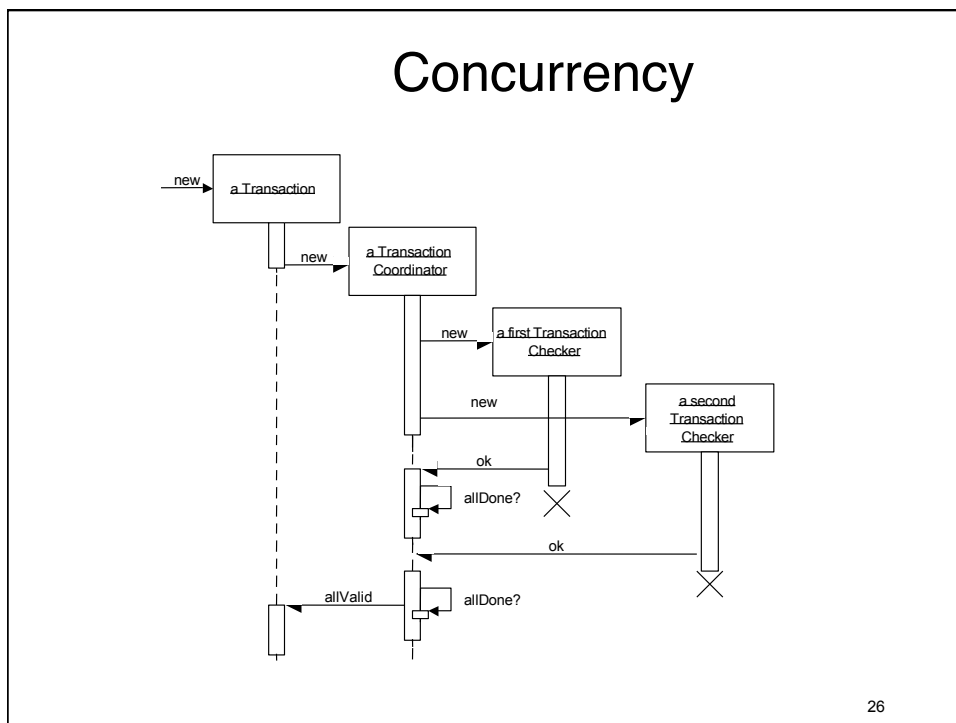
23

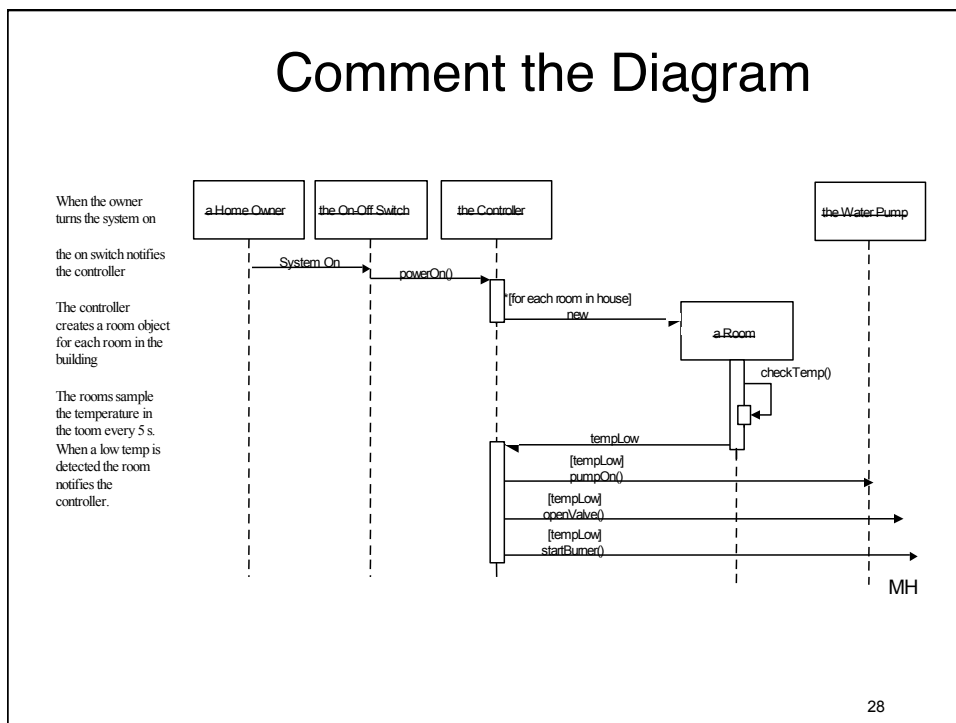
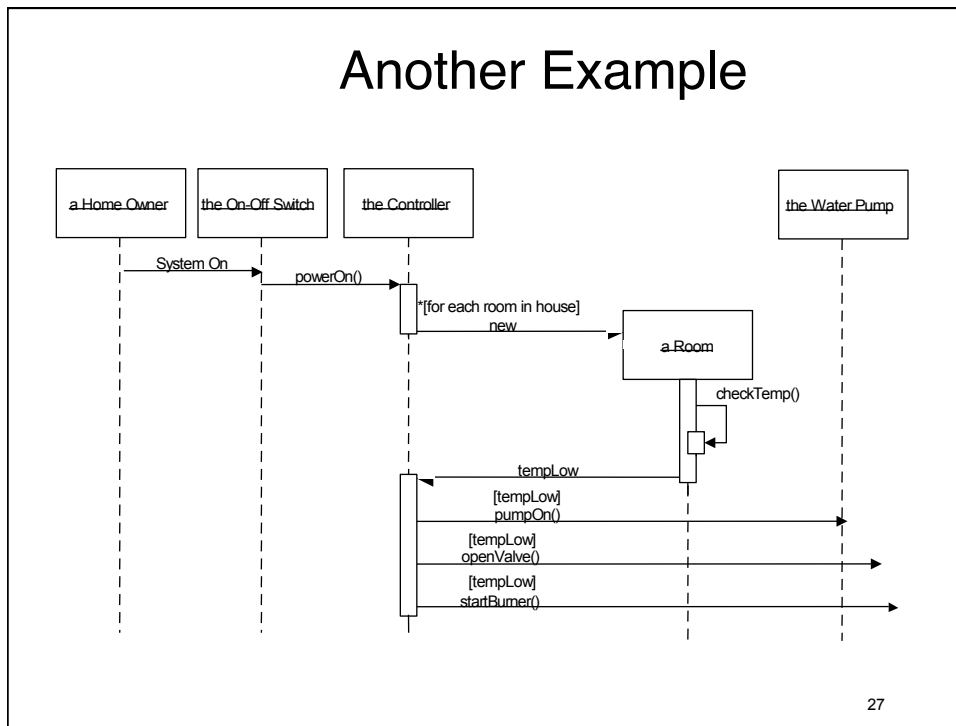


Example from Fowler

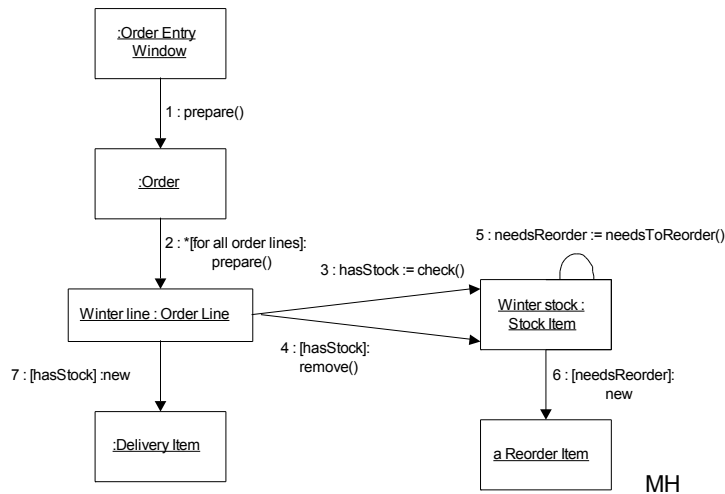


Concurrency

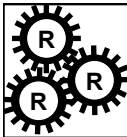




Collaboration Diagrams



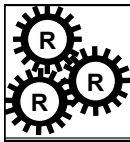
29



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
- **Remember, clarity is the goal!**

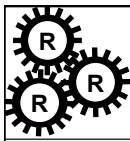
30



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

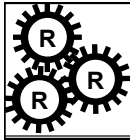
31



When to Use Interaction Diagrams

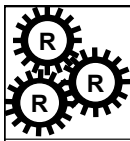
- When you want to clarify and explore single use-cases involving several objects
 - Quickly becomes unruly if you do not watch it
- If you are interested in one object over many use-cases -- **state transition diagrams**
- If you are interested in many objects over many use cases -- **activity diagrams**

32



State Diagrams

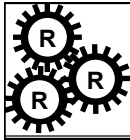
33



We Will Cover

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

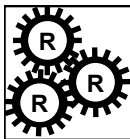
34



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

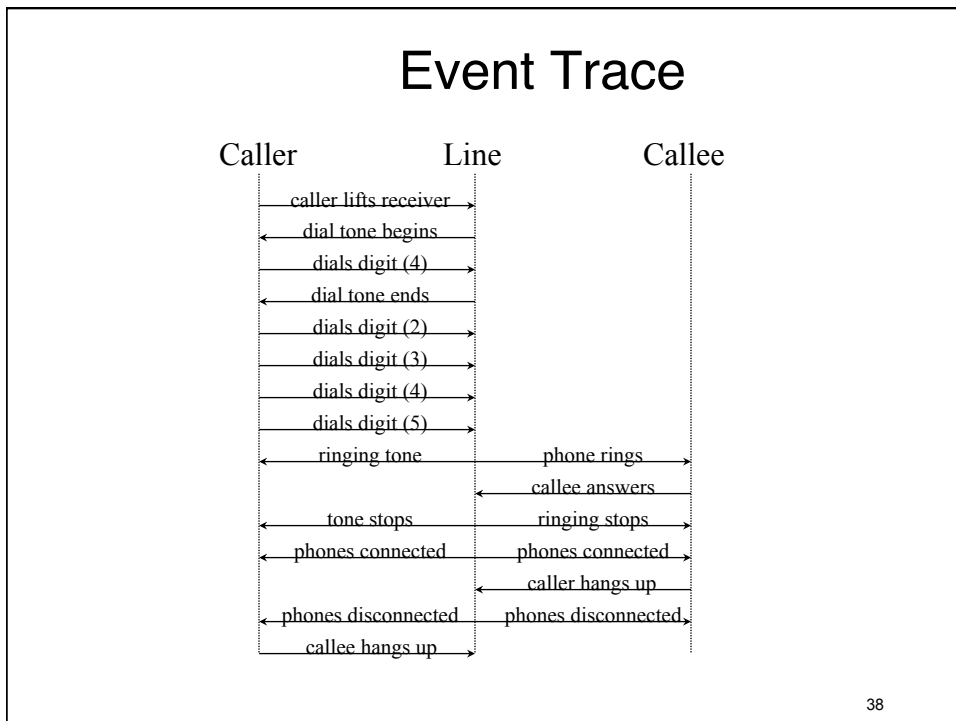
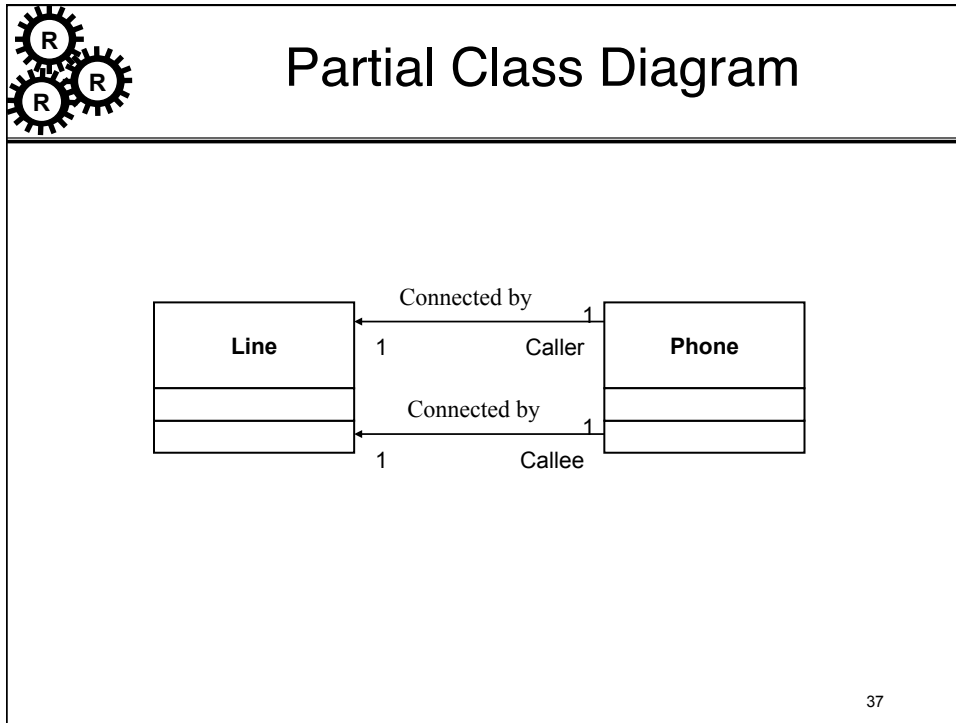
35



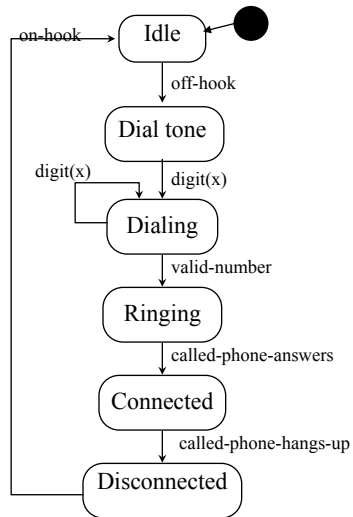
Making a Phone Call Scenario

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.

36

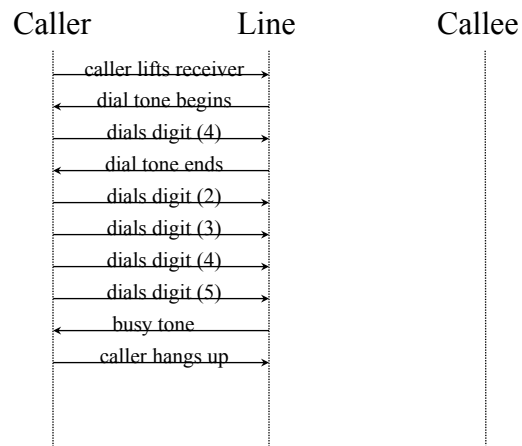


State Diagram for Scenario

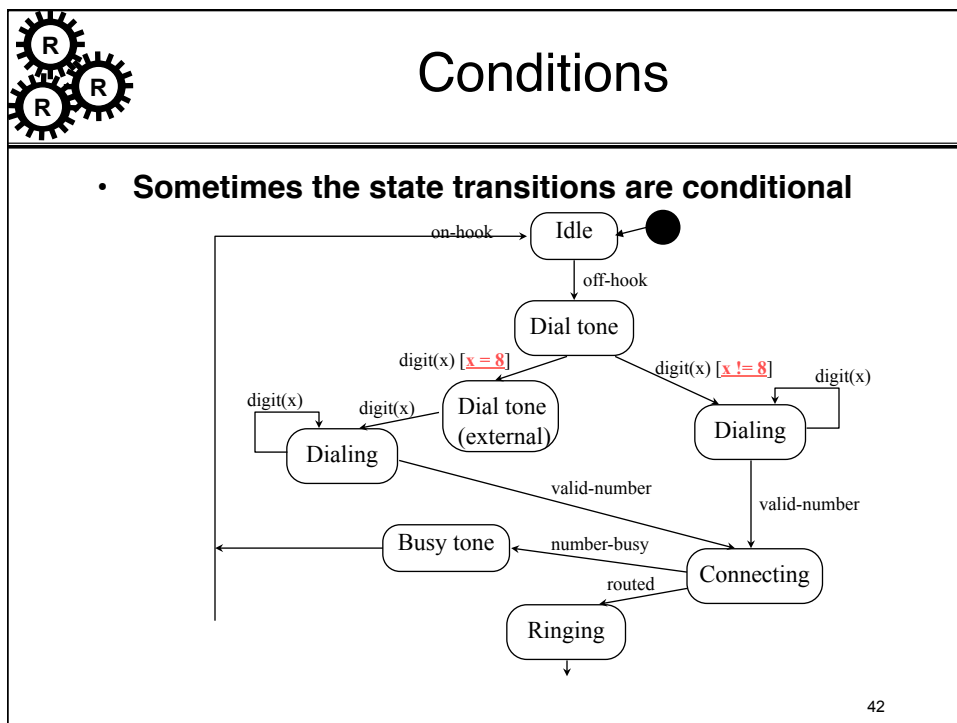
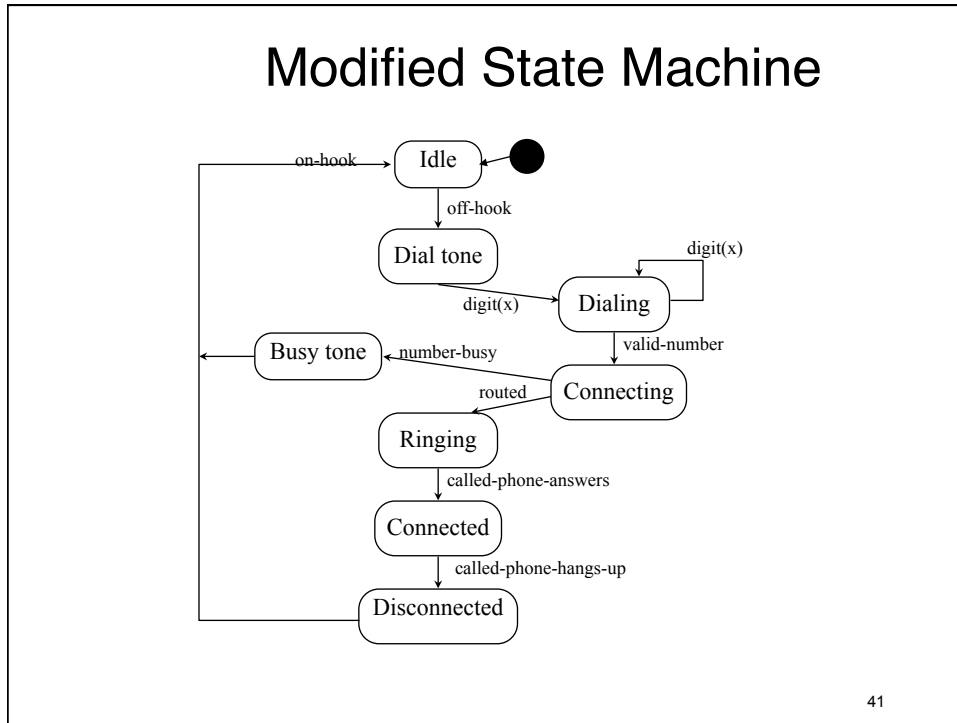


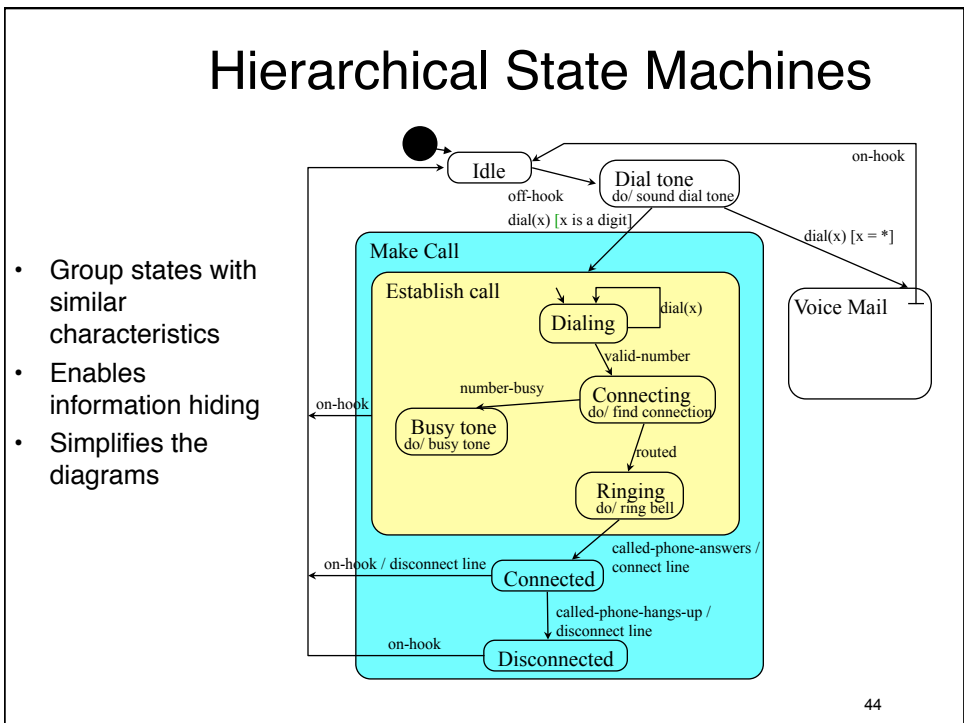
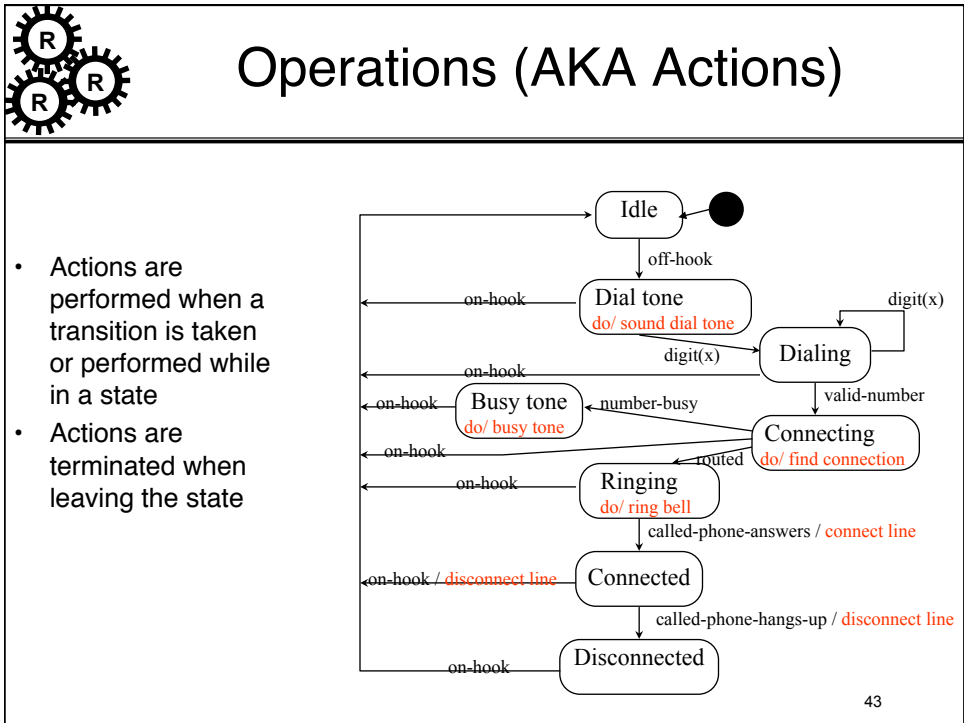
39

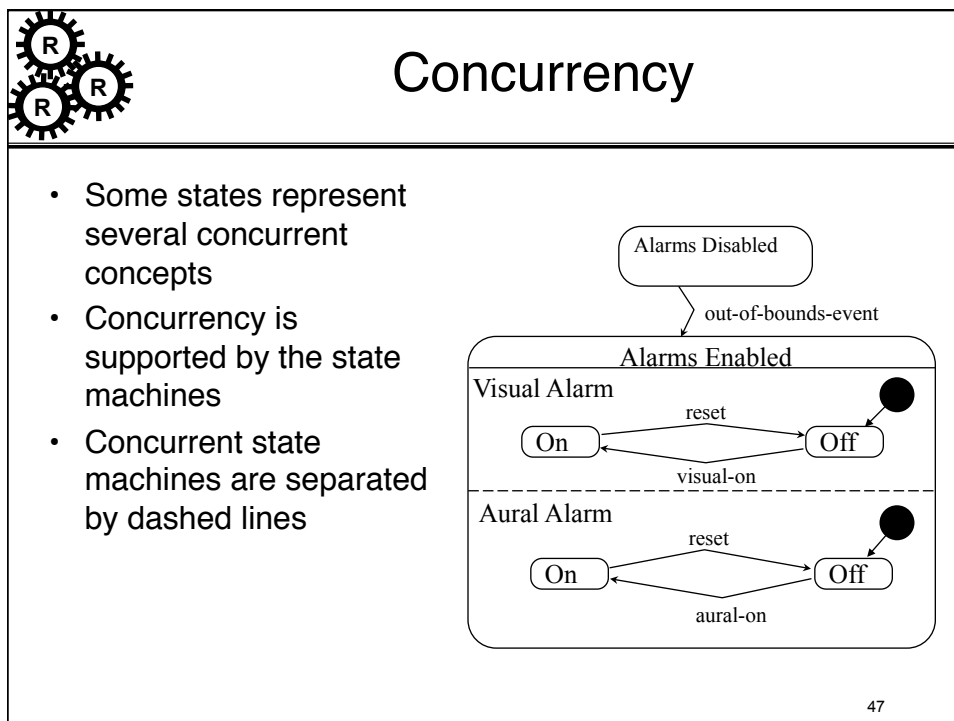
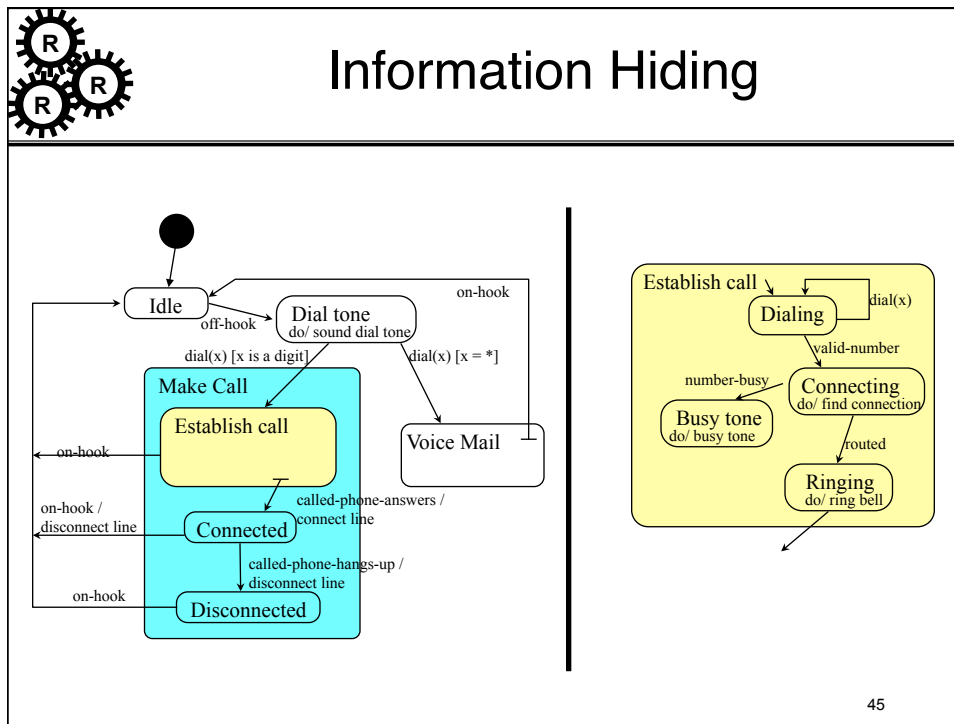
Scenario 2

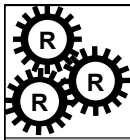


40





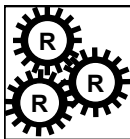




State Machines - Summary

- 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

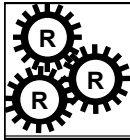
48



When to use State Machines

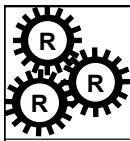
- When you want to describe the behavior of one object for all (or at least many) scenarios that affect that object
- Not good at showing the interaction between objects
 - Use interaction diagrams or activity diagrams
- Do not use them for all classes
 - Some methods prescribe this
 - Very time consuming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

49



Coming up with the State Diagrams

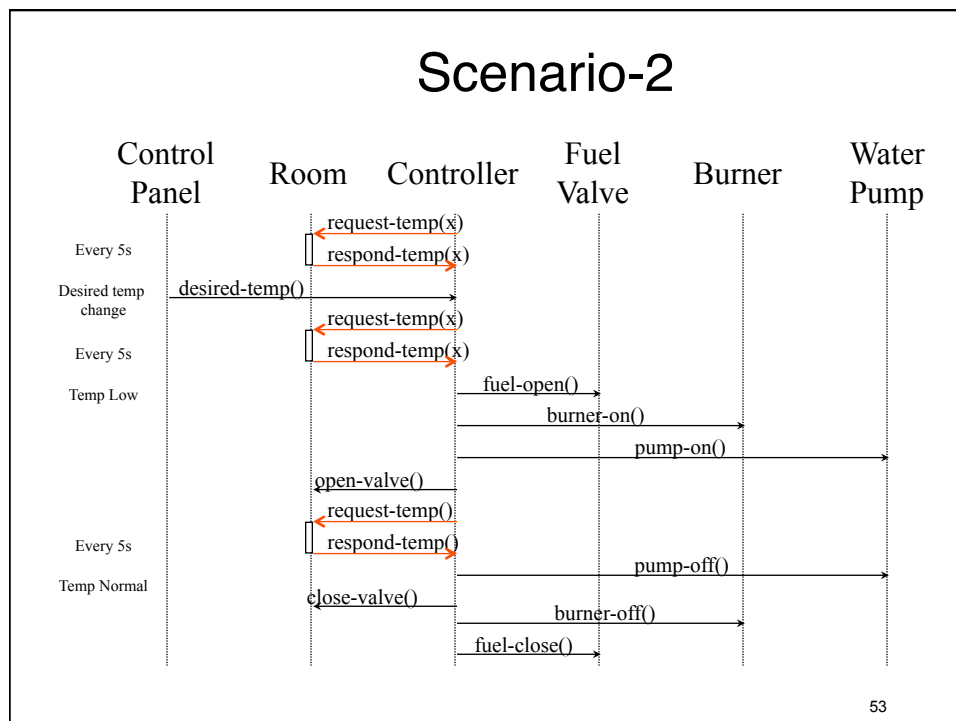
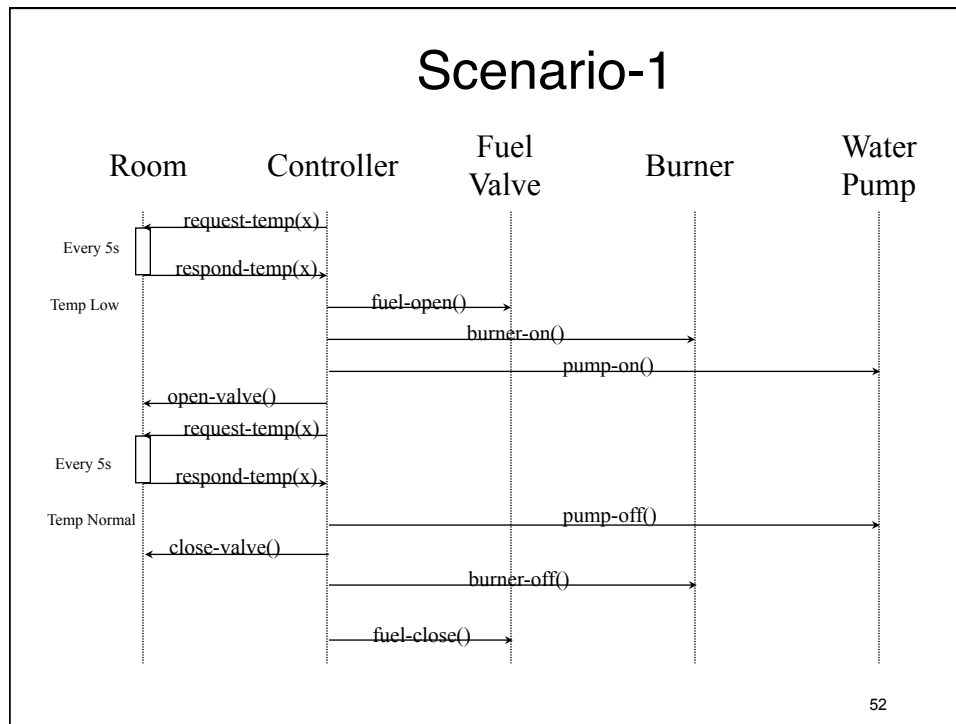
50

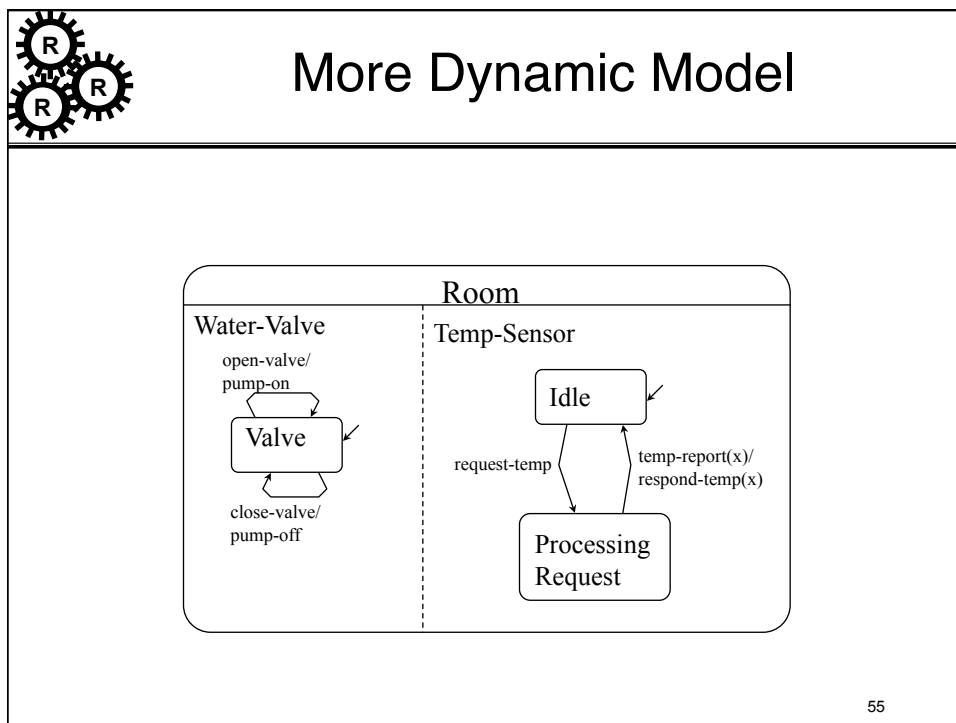
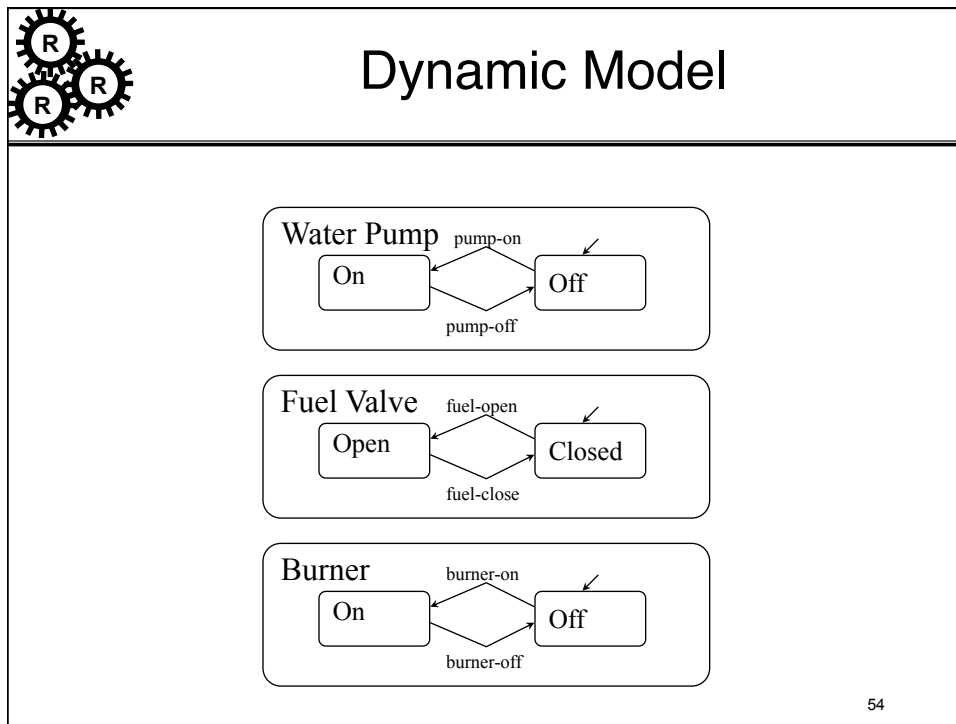


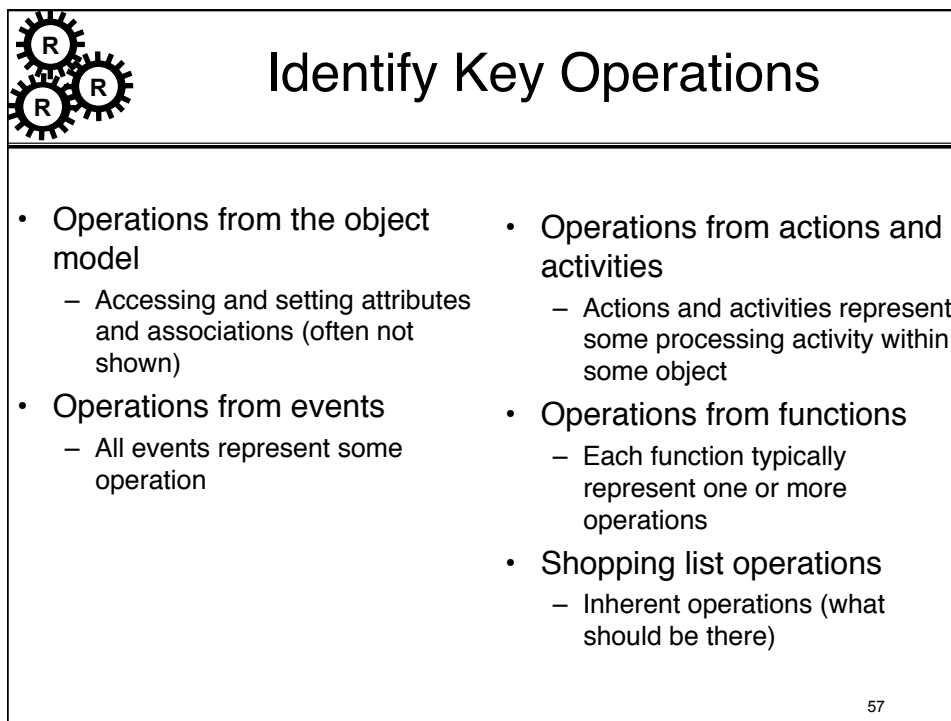
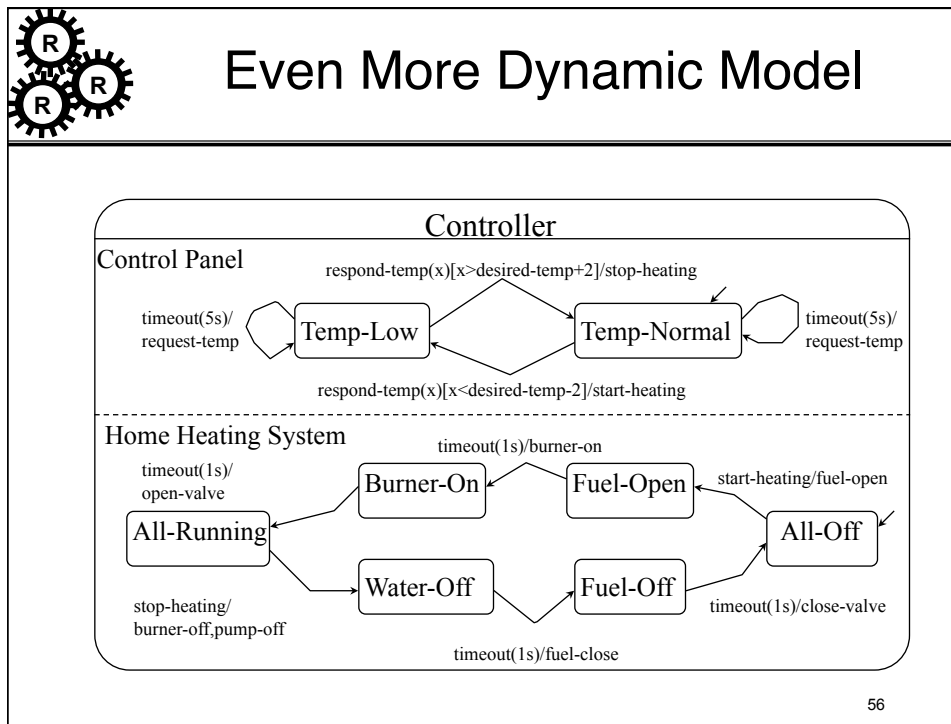
Modeling Approach

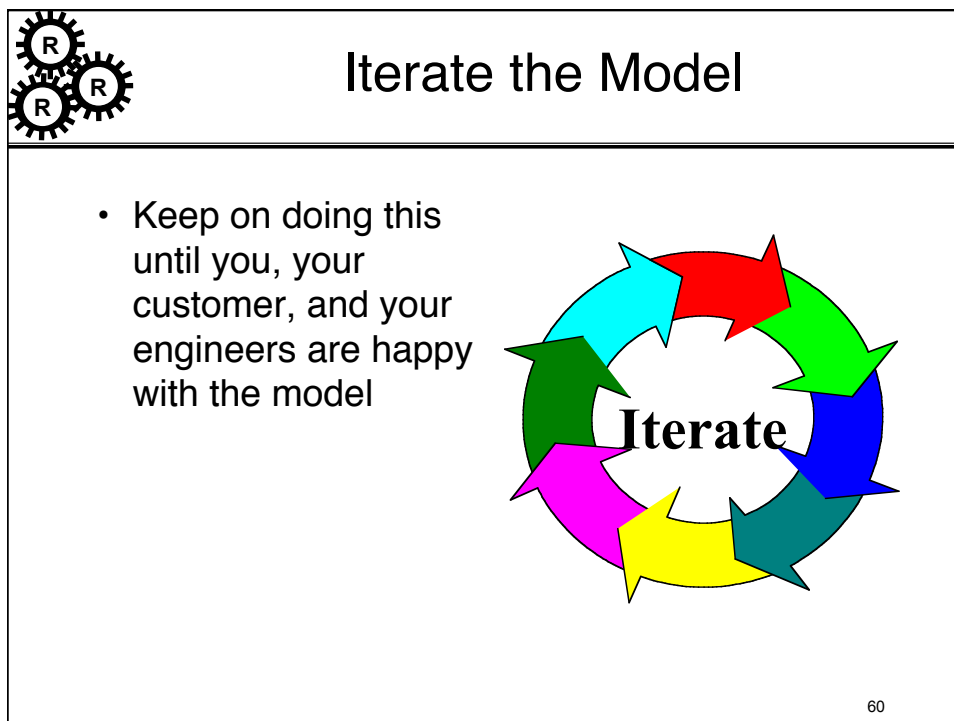
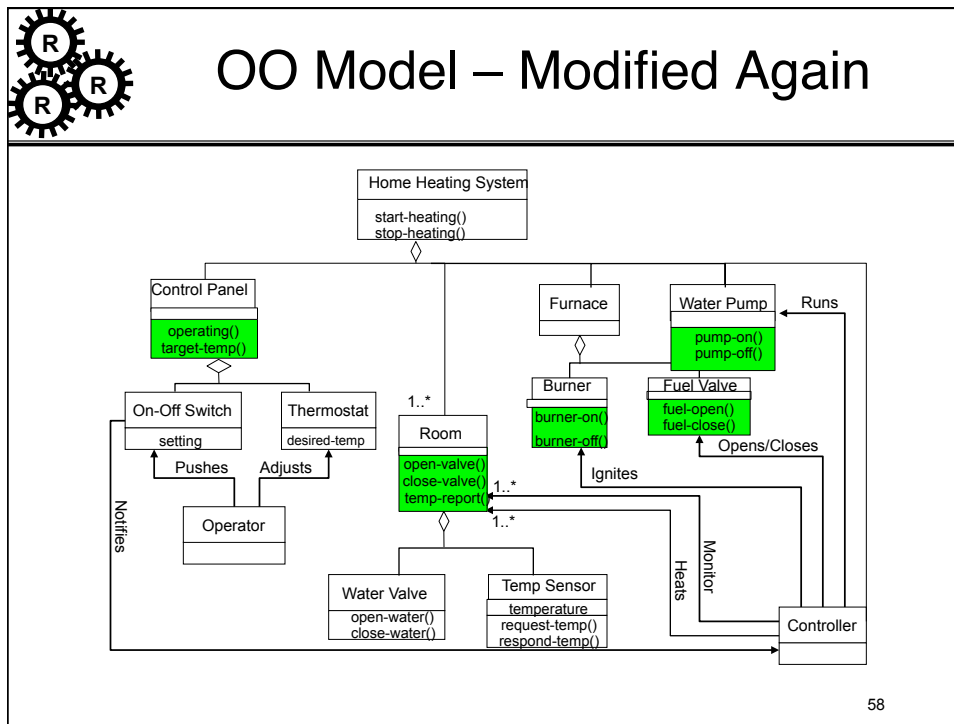
- Prepare scenarios
 - Work with the customer
 - Start with normal scenarios
 - Add abnormal scenarios
- Identify events (often messages)
 - Group into event classes
- Draw some sequence diagrams
 - Find objects with complex functionality you want to understand better
- Build a state diagram for the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

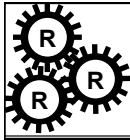
51





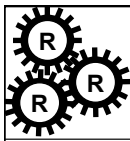






Activity Diagrams

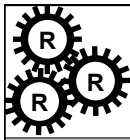
61



We Will Cover

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

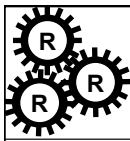
62



Activity Diagrams

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

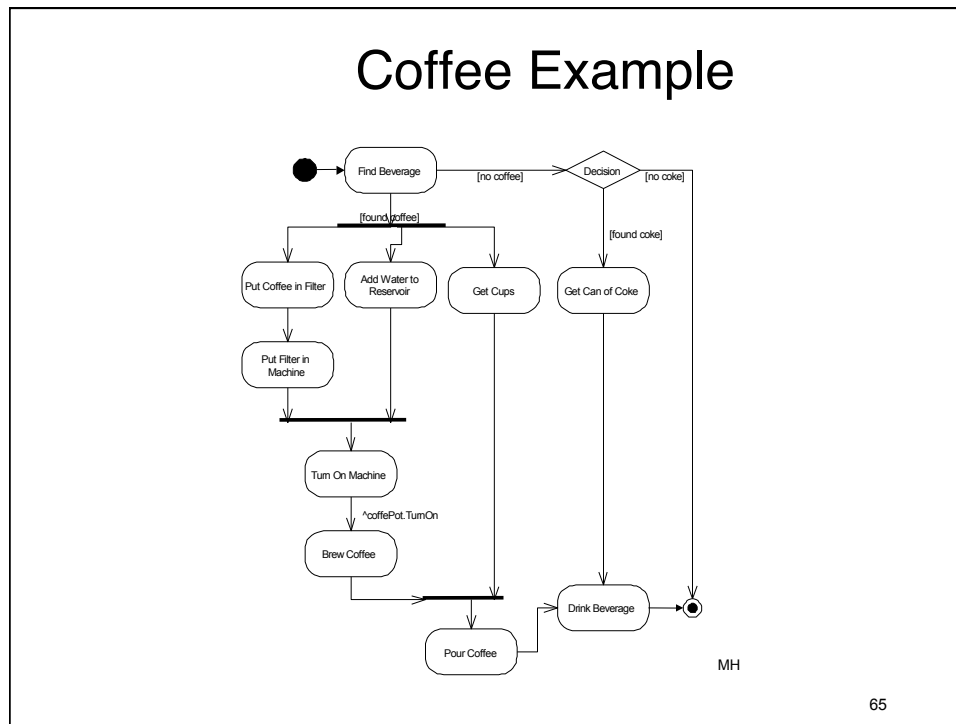
63



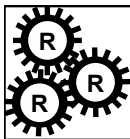
Why Activity Diagrams

- Very good question
 - Not part of any previous (UML related) method
 - Introduced to sell products
- 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
- Generally a flowchart and I do not really see the need in OO modeling
 - Probably because I do not do business systems

64



65



HACS Use-Cases

Use case: **Distribute Assignments**

Actors: Instructor (initiator), Student

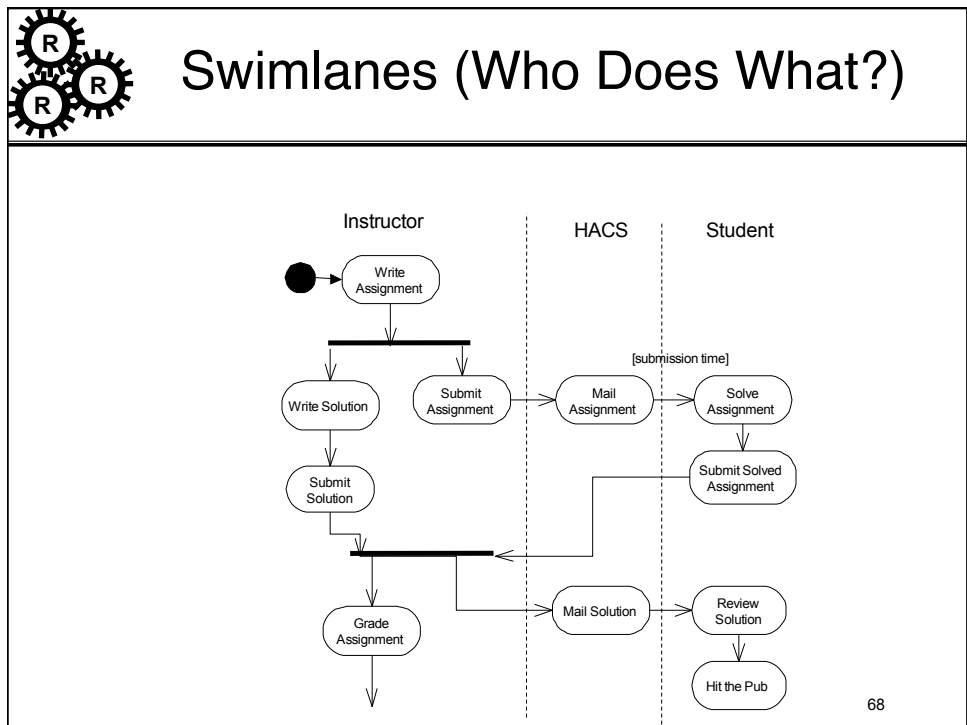
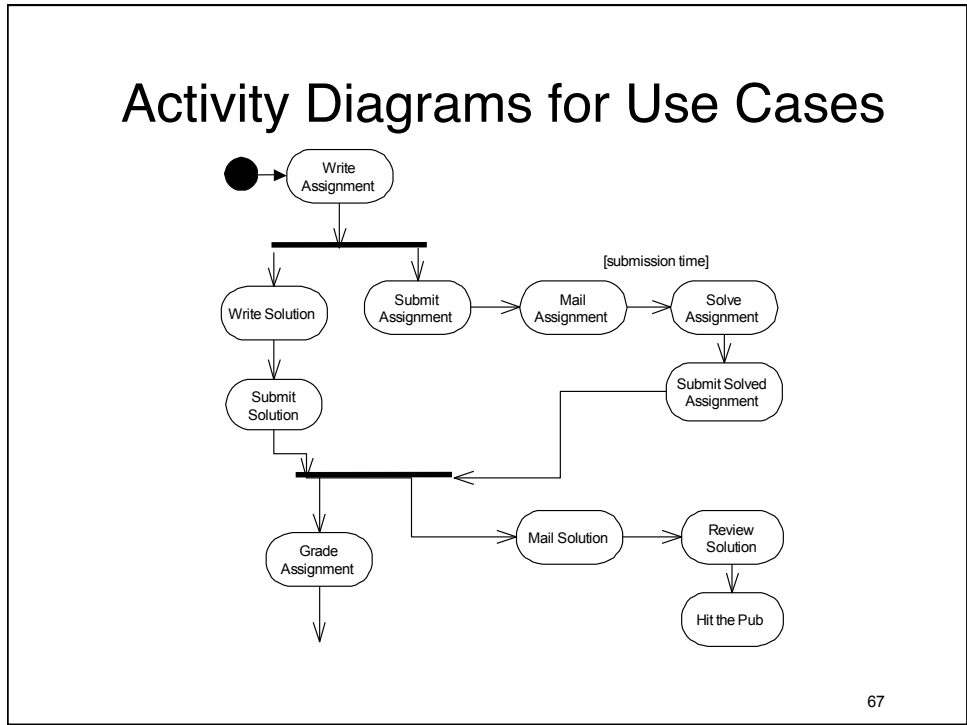
Type: Primary and essential

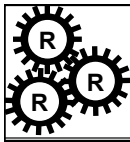
Description: The Instructor completes an assignment and submits it to the system. The instructor will also submit the delivery date, due date, and the class the assignment is assigned for. The system will at the due date mail the assignment to the student.

Cross Ref.: Requirements XX, YY, and ZZ

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

66

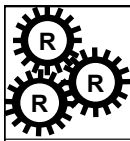




Problems with Activity Diagrams

- They are glorified flowcharts
 - Very easy to make a traditional data-flow oriented design
- Switching to the OO paradigm is hard enough as it is
 - Extensive use of activity charts can make this shift even harder
- However...
 - Very powerful when you know how to use them correctly

69



When to Use Activity Diagrams

- Not clear how useful in OO modeling
 - Particularly when modeling control systems
- 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

70