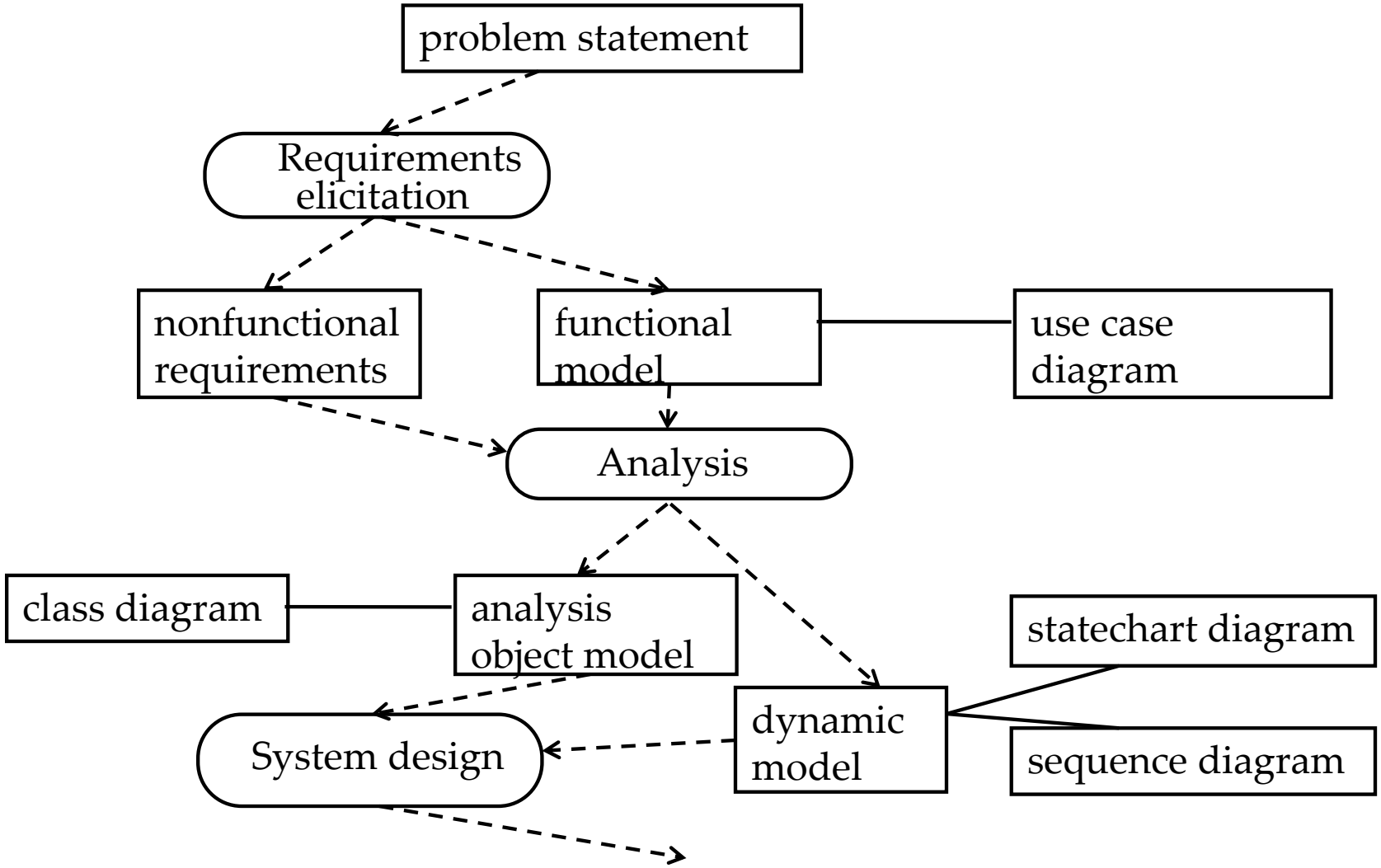
Object-Oriented Software Engineering
Using UML, Patterns, and Java

Lecture 5: Analysis, Object and Dynamic Modeling

Compiled By: Dr. Osama Hosameldeen

An overview of OOSE development activities and their products



Object Modeling

Pieces of an Object Model

- Classes
- Associations (Relations)
  - Generic associations
  - Canonical associations
    - Part of- Hierarchy (Aggregation)
    - Kind of-Hierarchy (Generalization)
- Attributes
  - Detection of attributes
  - Application specific
  - Attributes in one system can be classes in another system
  - Turning attributes to classes
- Operations
  - Detection of operations
  - Generic operations: Get/Set, General world knowledge, design patterns
  - Domain operations: Dynamic model, Functional model

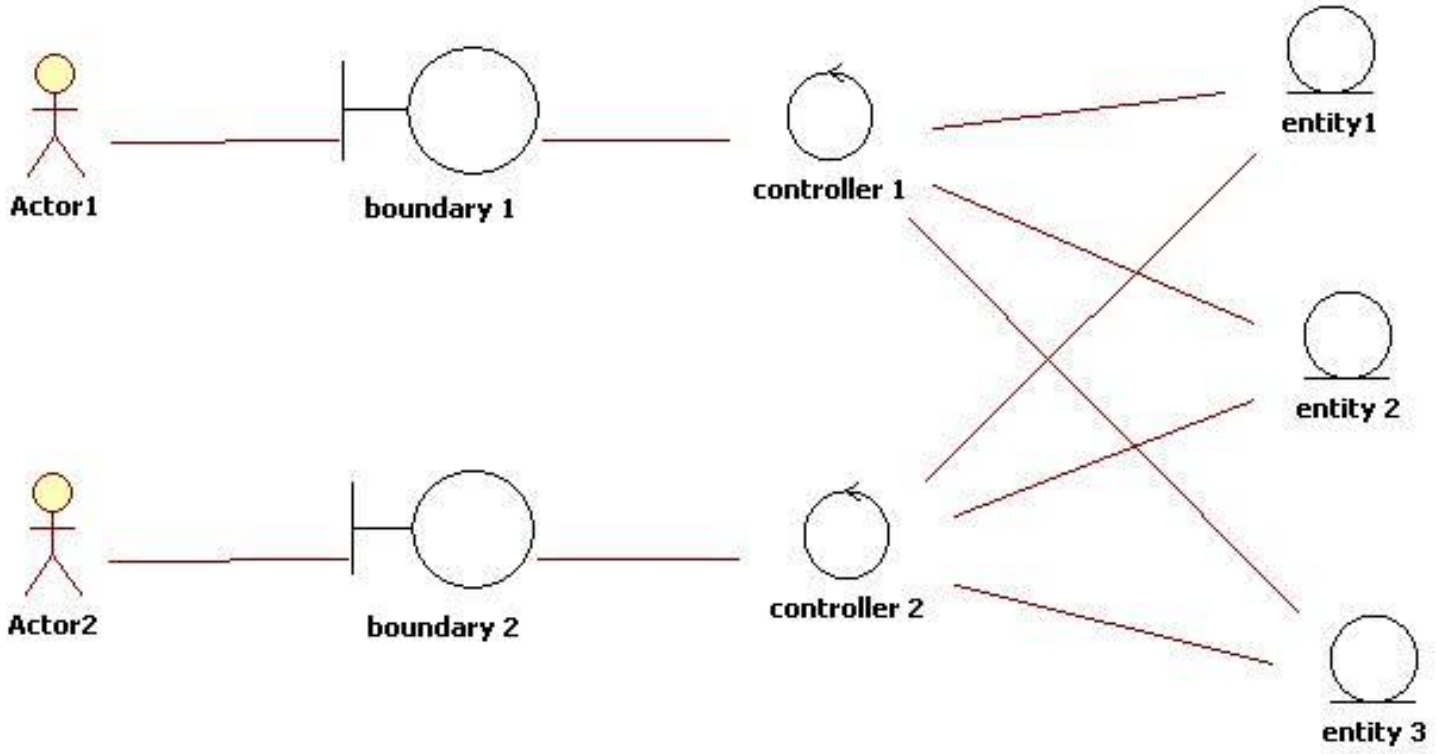
Object vs Class

- Object (instance): Exactly one thing
  - This lecture on Software Modeling on June 26 from 8:40-10:30
- A class describes a group of objects with similar properties
  - Game, Tournament, mechanic, car, database
- *Object diagram:* A graphic notation for modeling objects, classes and their relationships ("associations"):
  - ***Class diagram****:* Template for describing many instances of data. Useful for taxonomies, patters, schemata...
  - ***Instance diagram****:* A particular set of objects relating to each other. Useful for discussing scenarios, test cases and examples

There are different types of Objects

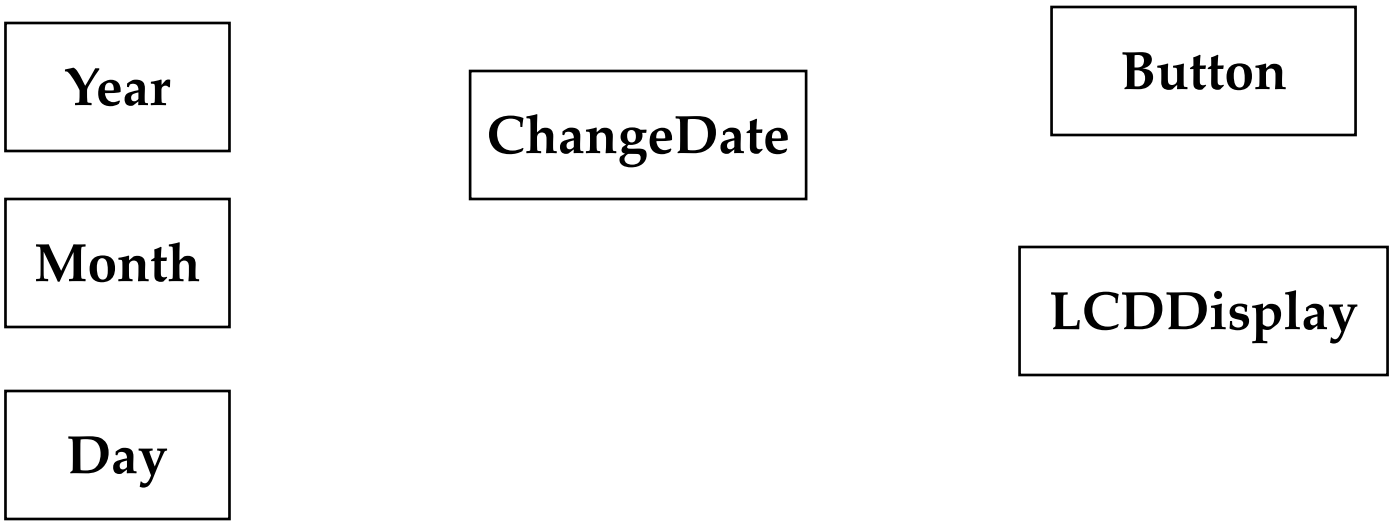
- Entity Objects
  - Represent the persistent **information** tracked by the system (Application domain objects, also called "Business objects")
- Boundary Objects
  - Represent the **interaction** between the user and the system
- Control Objects
  - Represent the **control** tasks performed by the system.

Entity-Control-Boundary Pattern



Example: 2BWatch Modeling

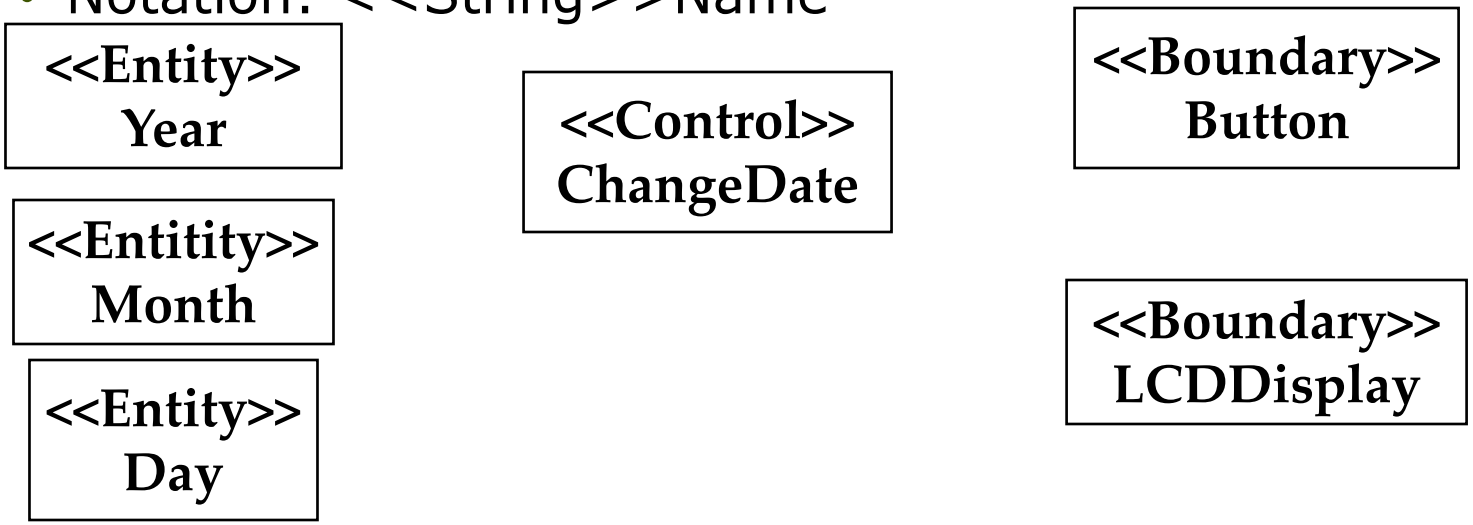
To distinguish different object types in a model we can use the UML Stereotype mechanism



Entity Objects Control Object Boundary Objects

Naming Object Types in UML

- UML provides the stereotype mechanism to introduce new types of modeling elements
  - A stereotype is drawn as a name enclosed by angled double-quotes (<<, >>) and placed before the name of a UML element (class, method, attribute, ….)
  - Notation: <<String>>Name



Entity Object Control Object Boundary Object

Banking

ATM Buttons	Boundary
ATM Screen	Boundary
Transfer Funds	Control
Teller's terminal	Boundary
Withdraw Funds	Control
Account Balance	Entity

UML is an Extensible Language

- Stereotypes allow you to extend the vocabulary of the UML so that you can create new model elements, derived from existing ones
- Examples:
  - Stereotypes can also be used to classify method behavior such as <<constructor>>, <<getter>> or <<setter>>
  - To indicate the interface of a subsystem or system, one can use the stereotype <<interface>> (Lecture System Design)
- Stereotypes can be represented with icons and graphics:
  - This can increase the readability of UML diagrams.

Object Types allow us to deal with Change

- Having three types of objects leads to models that are more resilient to change
  - The interface of a system changes more likely than the control
  - The way the system is controlled changes more likely than entities in the application domain
- Object types originated in Smalltalk:
  - Model, View, Controller (MVC)

Model <-> Entity Object

View <-> Boundary Object

Controller <-> Control Object

Finding Participating Objects in Use Cases

- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis)
  - Nouns are candidates for objects/classes
  - Verbs are candidates for operations
  - This is also called Abbott's Technique

- After objects/classes are found, identify their types
  - Identify real world entities that the system needs to keep track of (FieldOfficer Entity Object)
  - Identify real world procedures that the system needs to keep track of (EmergencyPlan Control Object)
  - Identify interface artifacts (PoliceStation Boundary Object).

Example for using the Technique

Flow of Events:

- The customer enters the store to buy a toy.
- It has to be a toy that his daughter likes and it must cost less than 50 Euros.
- He tries a videogame, which uses a data glove and a head-mounted display. He likes it.
- An assistant helps him.
- The suitability of the game depends on the age of the child.
- His daughter is only 3 years old.
- The assistant recommends another type of toy, namely the boardgame “Monopoly".

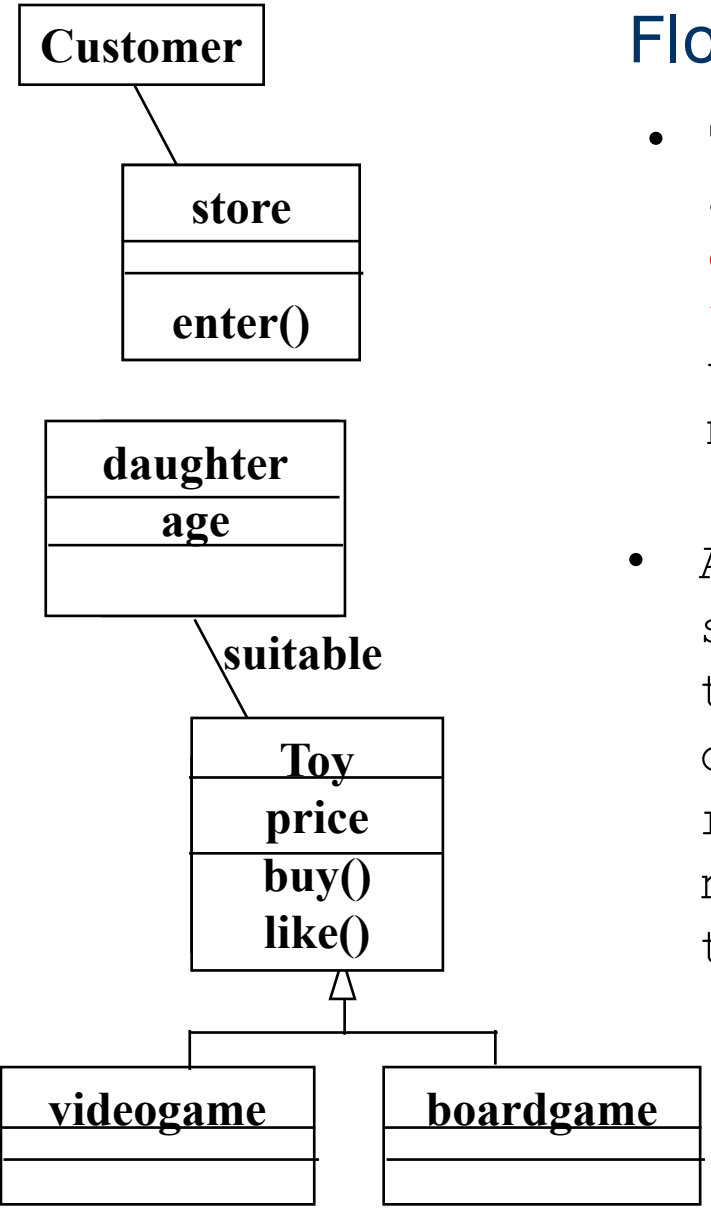
Mapping parts of speech to model components (Abbott's Technique)

Example	*Part of speech*	*UML model component*
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	constraint, class, association

Textual Analysis using Abbott's technique

Example	*Grammatical construct*	*UML Component*
"Monopoly"	Concrete Person, Thing	Object
"toy"	noun	class
"3 years old"	Adjective	Attribute
"enters"	verb	Operation
"depends on…."	Intransitive verb	Operation (Event)
"is a" ,"either..or", "kind of…"	Classifying verb	Inheritance
"Has a ", "consists of"	Possessive Verb	Aggregation
"must be", "less than…"	modal Verb	Constraint

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

- An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Dynamic Modeling

Dynamic Modeling

- Definition of Dynamic modeling
- Sequence diagrams
- Statechart Diagrams
- Activity Diagrams
- Requirements analysis model validation

Dynamic Modeling

- Definition of a dynamic model:
  - Describes the components of the system that have interesting dynamic behavior
- Purpose:
  - Detect and supply operations for the object model.
- How do we do this?
  - Start with use case or scenario
  - Model interaction between objects => sequence diagram
  - Model dynamic behavior of a single object => statechart diagram

Dynamic Modeling with UML

- The dynamic model is described with
  - Sequence diagrams: For the interaction between classes
    - Dynamic behavior of a set of objects arranged in time sequence.
    - Good for real-time specifications and complex scenarios
  - State diagrams: One state diagram for each class with interesting dynamic behavior
    - Classes without interesting dynamic behavior are not modeled with state diagrams
  - Activity diagrams: A special type of statechart diagram, where all states are action states (captured by a use case)

Sequence Diagram

- From the flow of events in the use case or scenario proceed to the sequence diagram
- A sequence diagram is a graphical description of the objects participating in a use case using a DAG (directed acyclic graph) notation
- Relation to object identification:
  - Objects/classes have already been identified during object modeling
  - Objects are identified as a result of dynamic modeling
- Heuristic for finding participating objects:
  - A event always has a **sender** and a **receiver**
  - The representation of the event is sometimes called a message
  - Find them for each event => These are the objects participating in the use case.

An Example

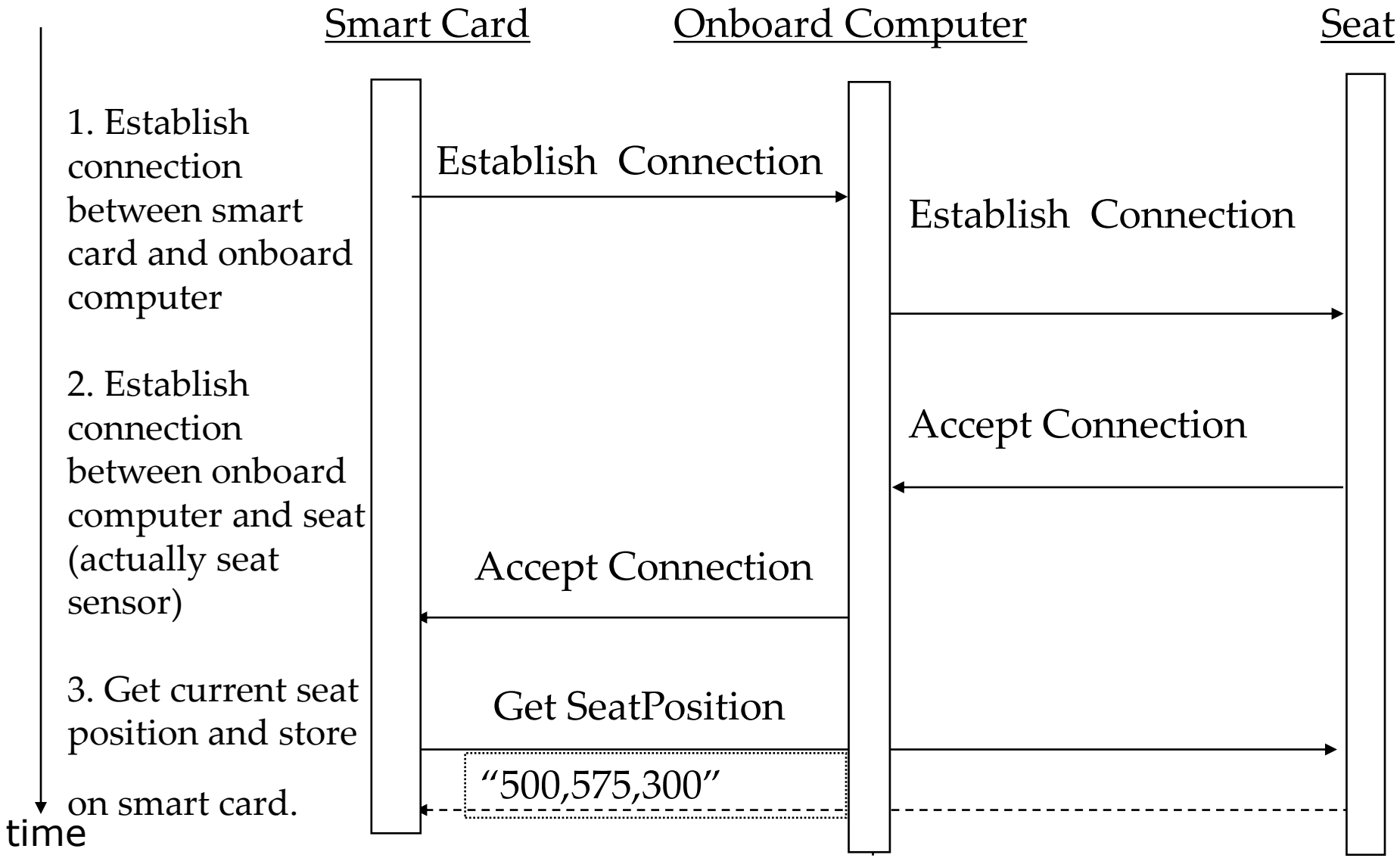
GetSeatPosition: Passenger tries to find an empty seat in a train using an onboard computer connected to seat sensors and a smart card.

- Flow of events in "GetSeatPosition" use case :

  1. Establish connection between smart card and onboard computer
  2. Establish connection between onboard computer and sensor for seat
  3. Get current seat position and store on smart card

- Where are the objects?

Sequence Diagram for "GetSeatPosition"



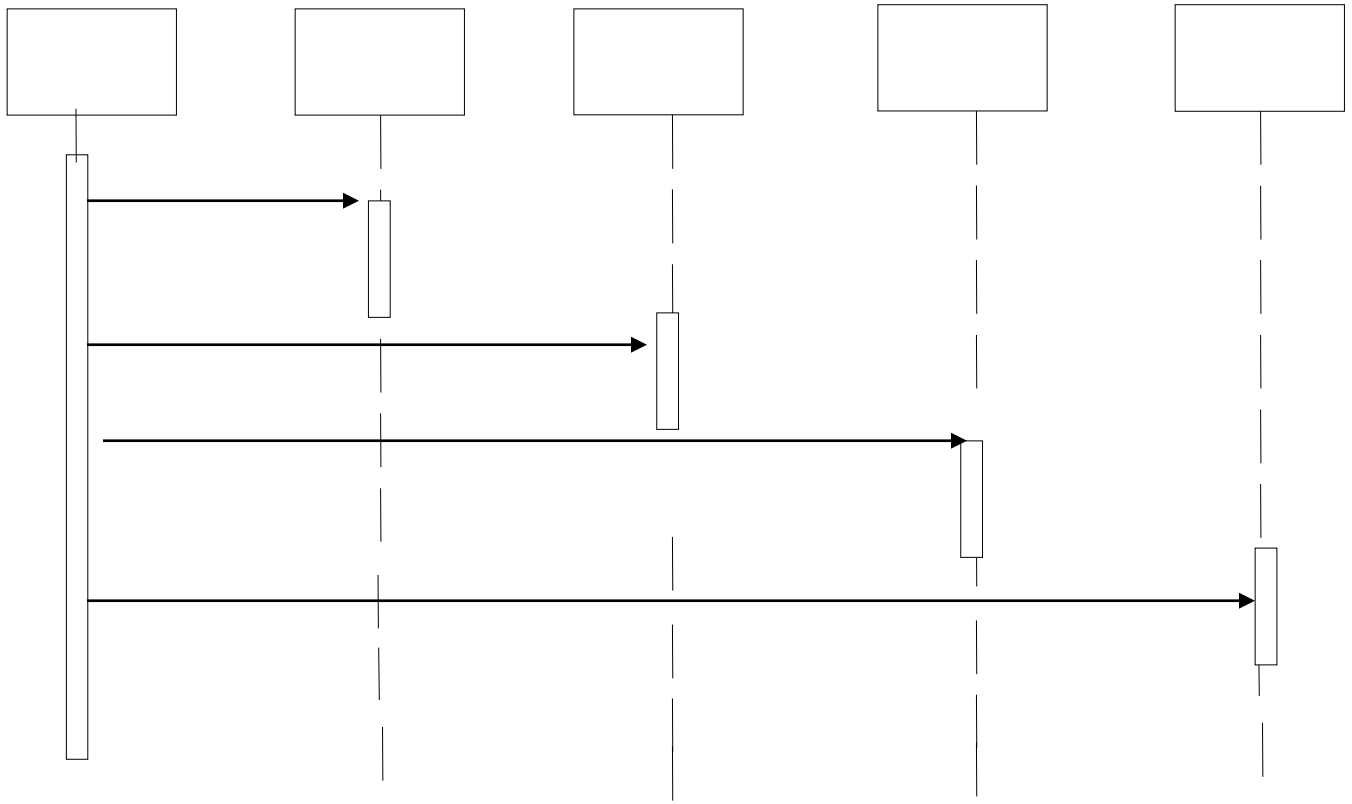
Heuristics for Sequence Diagrams

- Layout:
  1st column: Should be the actor of the use case
  2nd column: Should be a boundary object
  3rd column: Should be the control object that manages the rest of the use case

- Creation of objects:
  - Create control objects at beginning of event flow
  - The control objects create the boundary objects
- Access of objects:
  - Entity objects can be accessed by control and boundary objects
  - Entity objects should not access boundary or control objects.

Fork Diagram

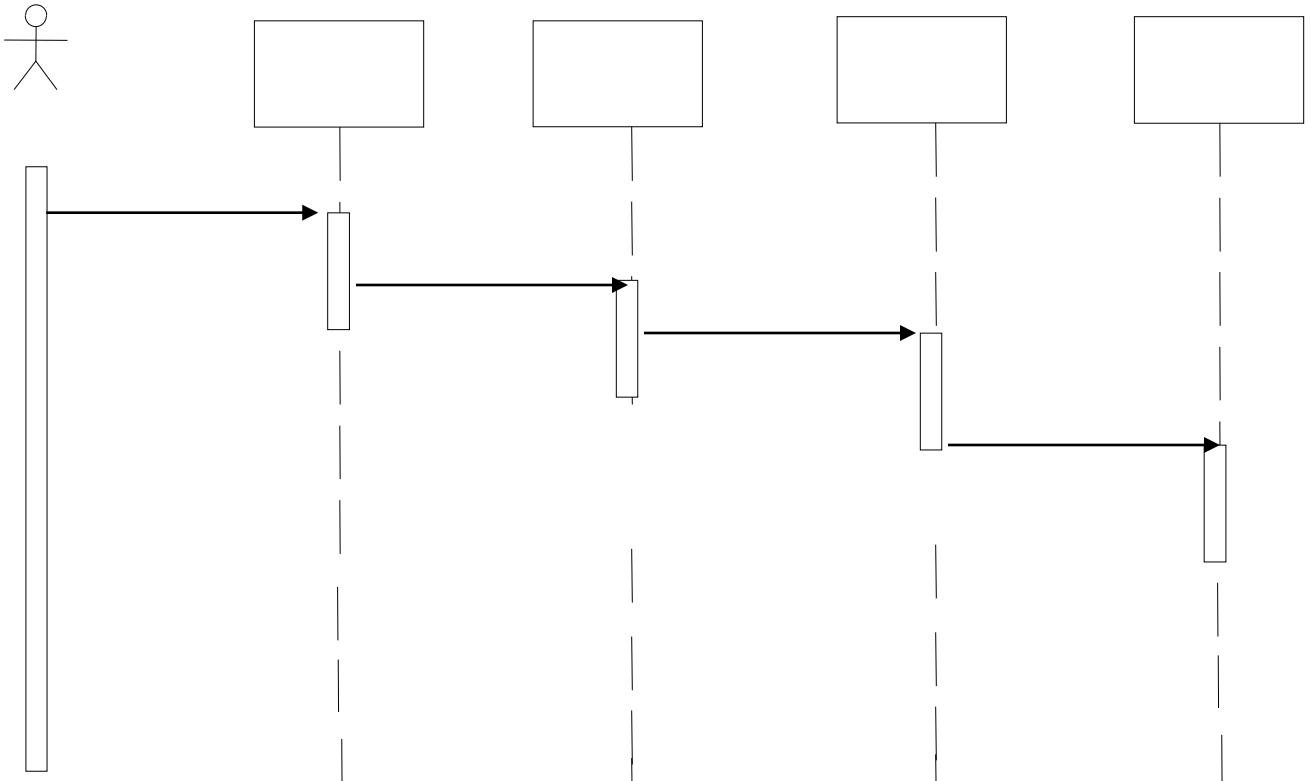
- Much of the dynamic behavior is placed in a single object, usually the control object. It knows all the other objects and often uses them for direct questions and commands.



Sequence Diagram:

Stair Diagram

- The dynamic behavior is distributed. Each object delegates some responsibility to other objects. Each object knows only a few of the other objects and knows which objects can help with a specific behavior.



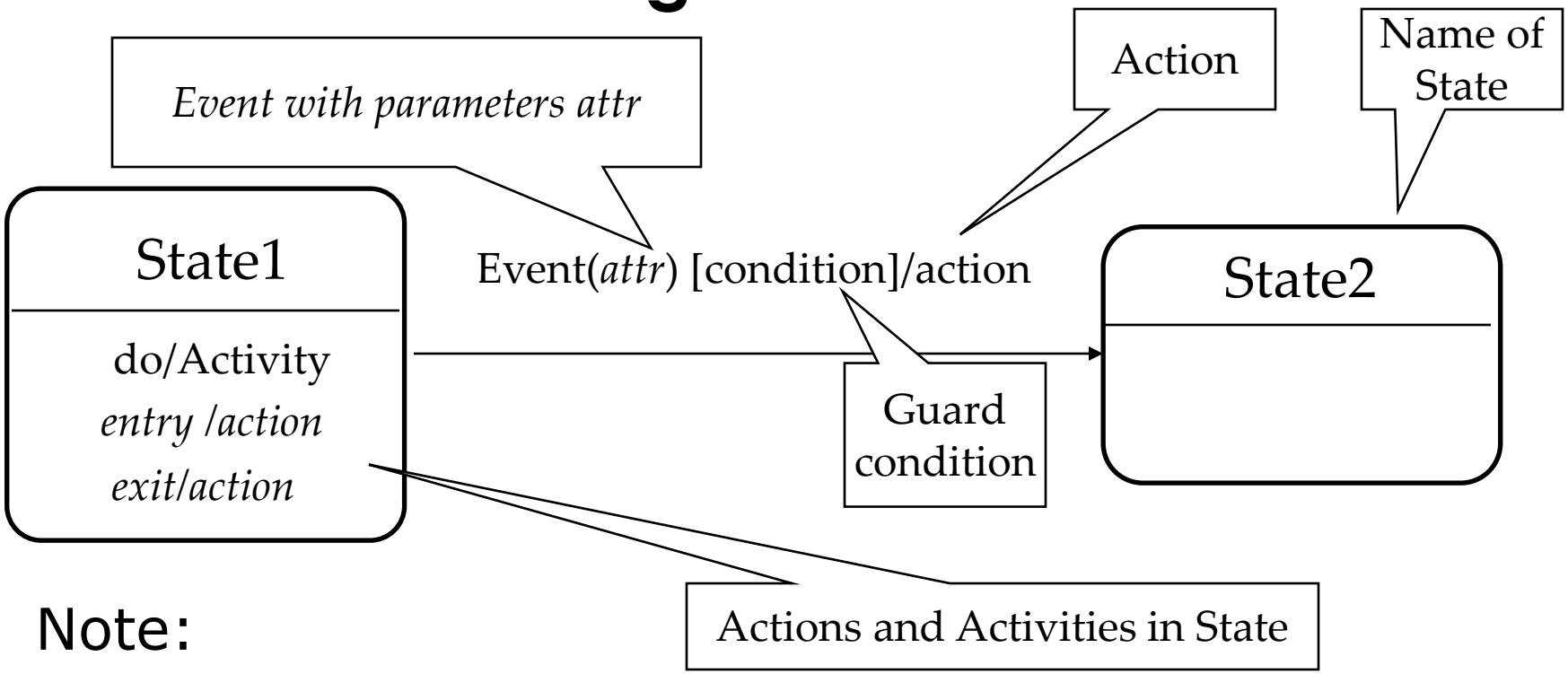
Statechart Diagrams

- Graph whose nodes are states and whose directed arcs are transitions labeled by event names.
- We distinguish between two types of operations:
  - Activity: Operation that takes time to complete
    - associated with states
  - Action: Instantaneous operation
    - associated with events
- A state chart diagram relates events and states for one class
- An object model with several classes with interesting behavior has *a set* of state diagrams

State

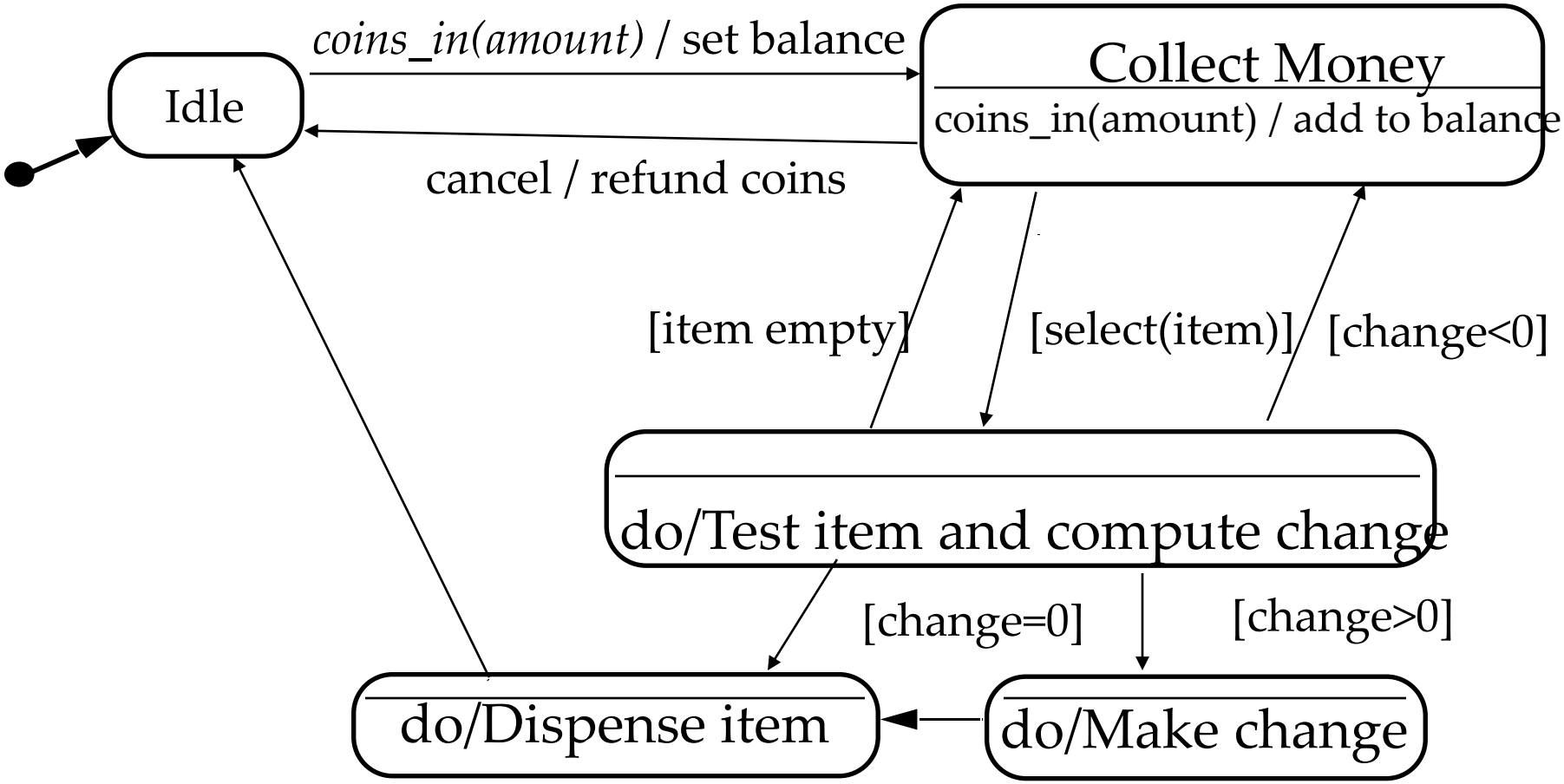
- An abstraction of the attributes of a class
  - State is the aggregation of several attributes a class
- A state is an equivalence class of all those attribute values and links that do no need to be distinguished
  - Example: State of a bank
- State has duration

UML Statechart Diagram Notation



- Note:
  - *Events are italics*
  - Conditions are enclosed with brackets: []
  - Actions and activities are prefixed with a slash /
- Notation is based on work by Harel
- Added are a few object-oriented modifications.

Example of a StateChart Diagram

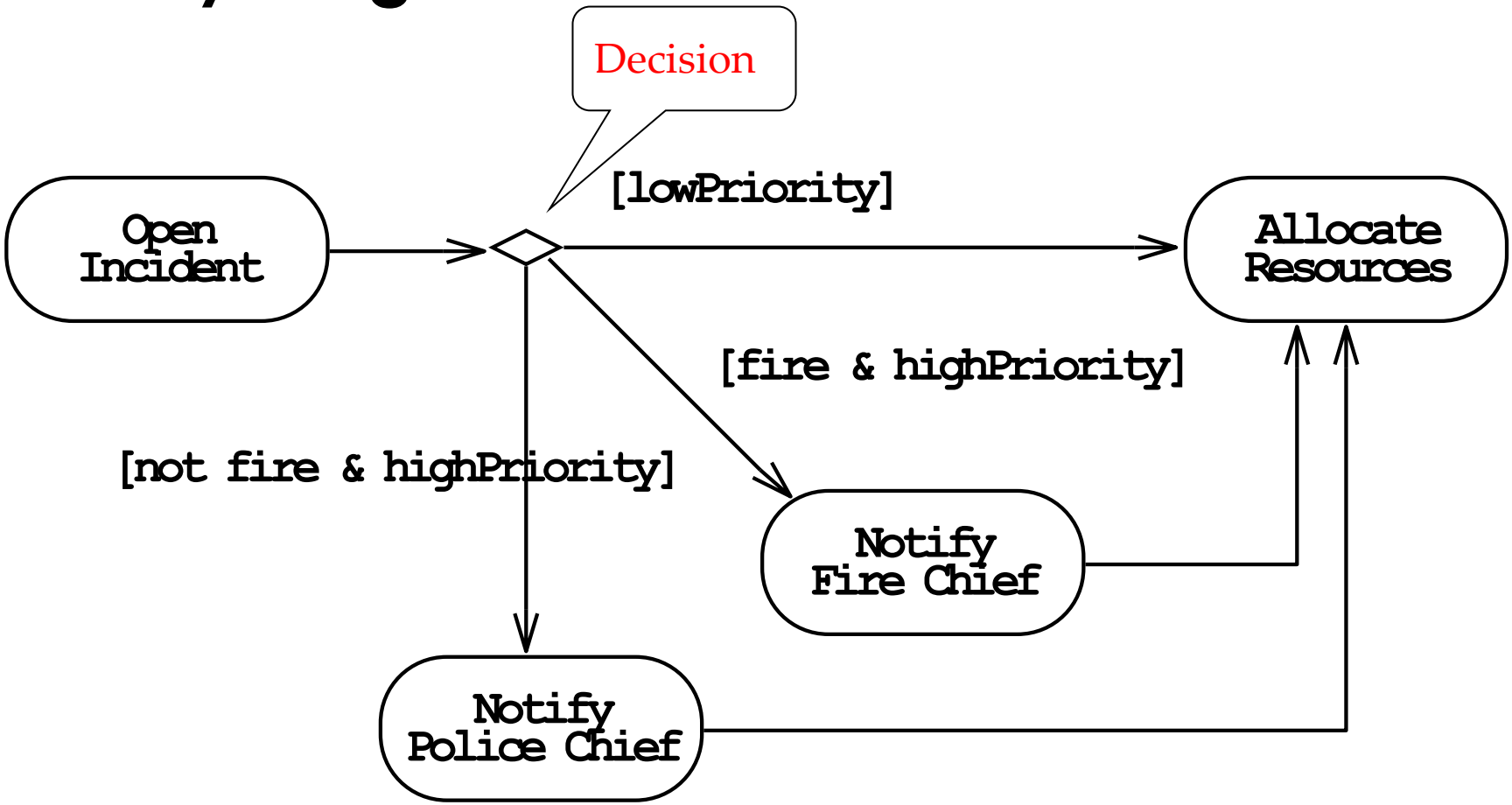


Vending Machine

Activity Diagrams

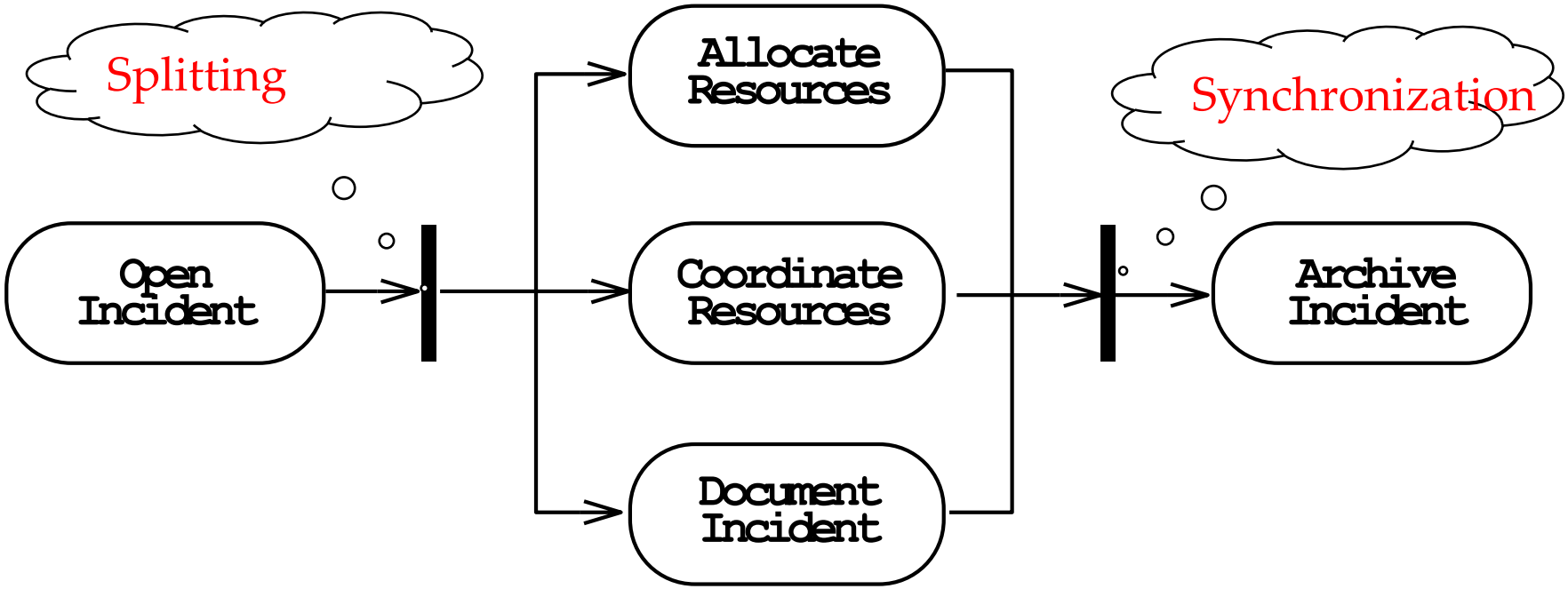
- An activity diagram is useful to depict the workflow in a system

Activity Diagrams allow to model Decisions



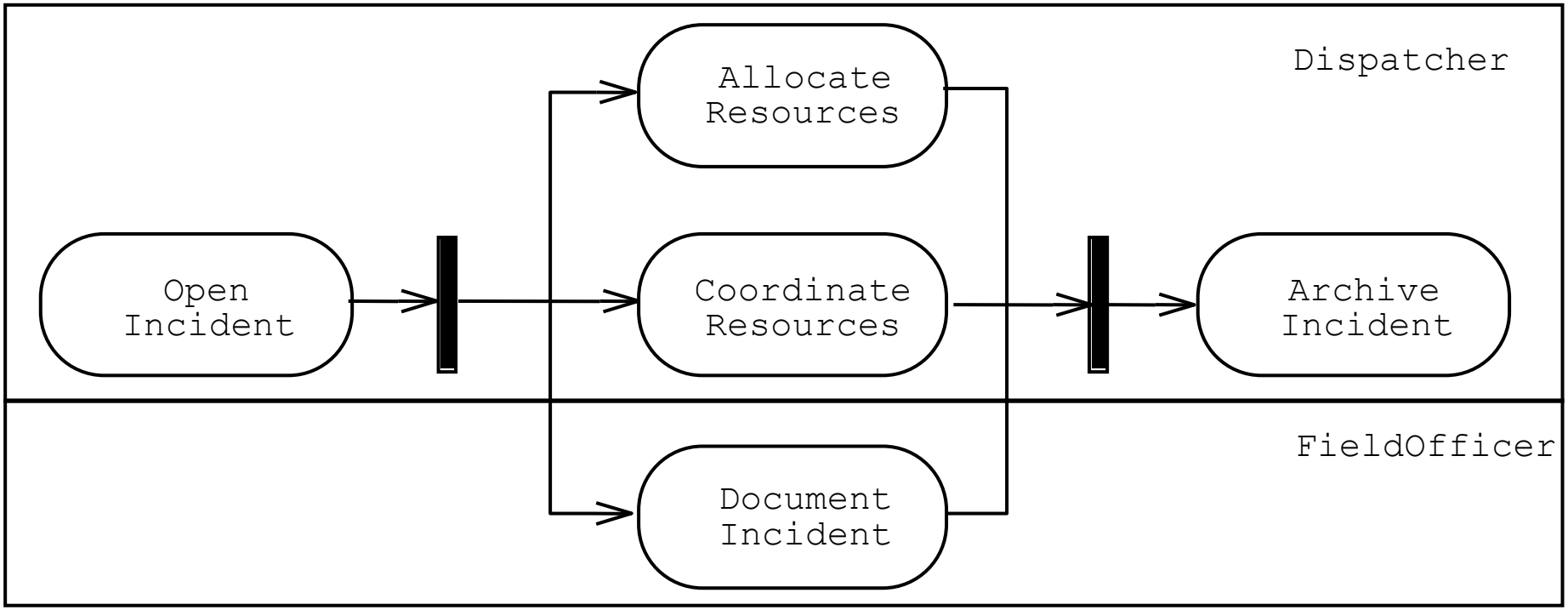
Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads



Activity Diagrams: Grouping of Activities

- Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.



Problem Statement: Direction Control for a Toy Car

- Power is turned on
  - Car moves forward and car headlight shines
- Power is turned off
  - Car stops and headlight goes out.
- Power is turned on
  - Headlight shines
- Power is turned off
  - Headlight goes out.
- Power is turned on
  - Car runs backward with its headlight shining.

- Power is turned off
  - Car stops and headlight goes out.
- Power is turned on
  - Headlight shines
- Power is turned off
  - Headlight goes out.
- Power is turned on
  - Car runs forward with its headlight shining.

Find the Functional Model: Do Use Case Modeling

- Use case 1: System Initialization
  - Entry condition: Power is off, car is not moving
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves forward, headlight is on
- Use case 2: Turn headlight off
  - Entry condition: Car moves forward with headlights on
  - Flow of events:
    - Driver turns power off, car stops and headlight goes out.
    - Driver turns power on, headlight shines and car does not move.
    - Driver turns power off, headlight goes out
  - Exit condition: Car does not move, headlight is out

Use Cases continued

- Use case 3: Move car backward
  - Entry condition: Car is stationary, headlights off
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves backward, headlight on
- Use case 4: Stop backward moving car
  - Entry condition: Car moves backward, headlights on
  - Flow of events:
    - Driver turns power off, car stops, headlight goes out.
    - Power is turned on, headlight shines and car does not move.
    - Power is turned off, headlight goes out.
  - Exit condition: Car does not move, headlight is out.
- Use case 5: Move car forward
  - Entry condition: Car does not move, headlight is out
  - Flow of events
    - Driver turns power on
  - Exit condition:
    - Car runs forward with its headlight shining.

Use Case Pruning

- Do we need use case 5?

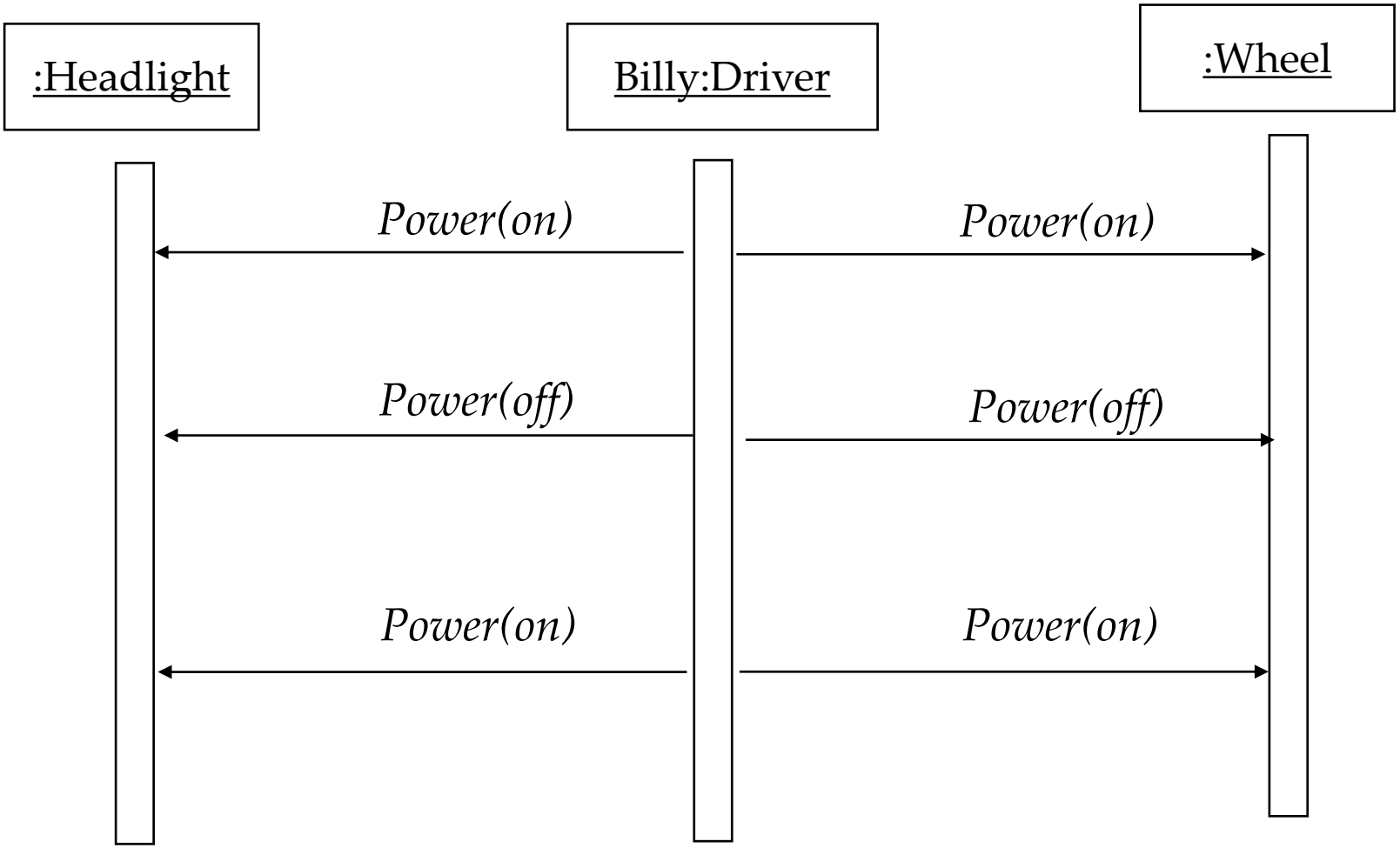
- Use case 1: System Initialization
  - Entry condition: Power is off, car is not moving
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves forward, headlight is on

- Use case 5: Move car forward
  - Entry condition: Car does not move, headlight is out
  - Flow of events
    - Driver turns power on
  - Exit condition:
    - Car runs forward with its headlight shining.

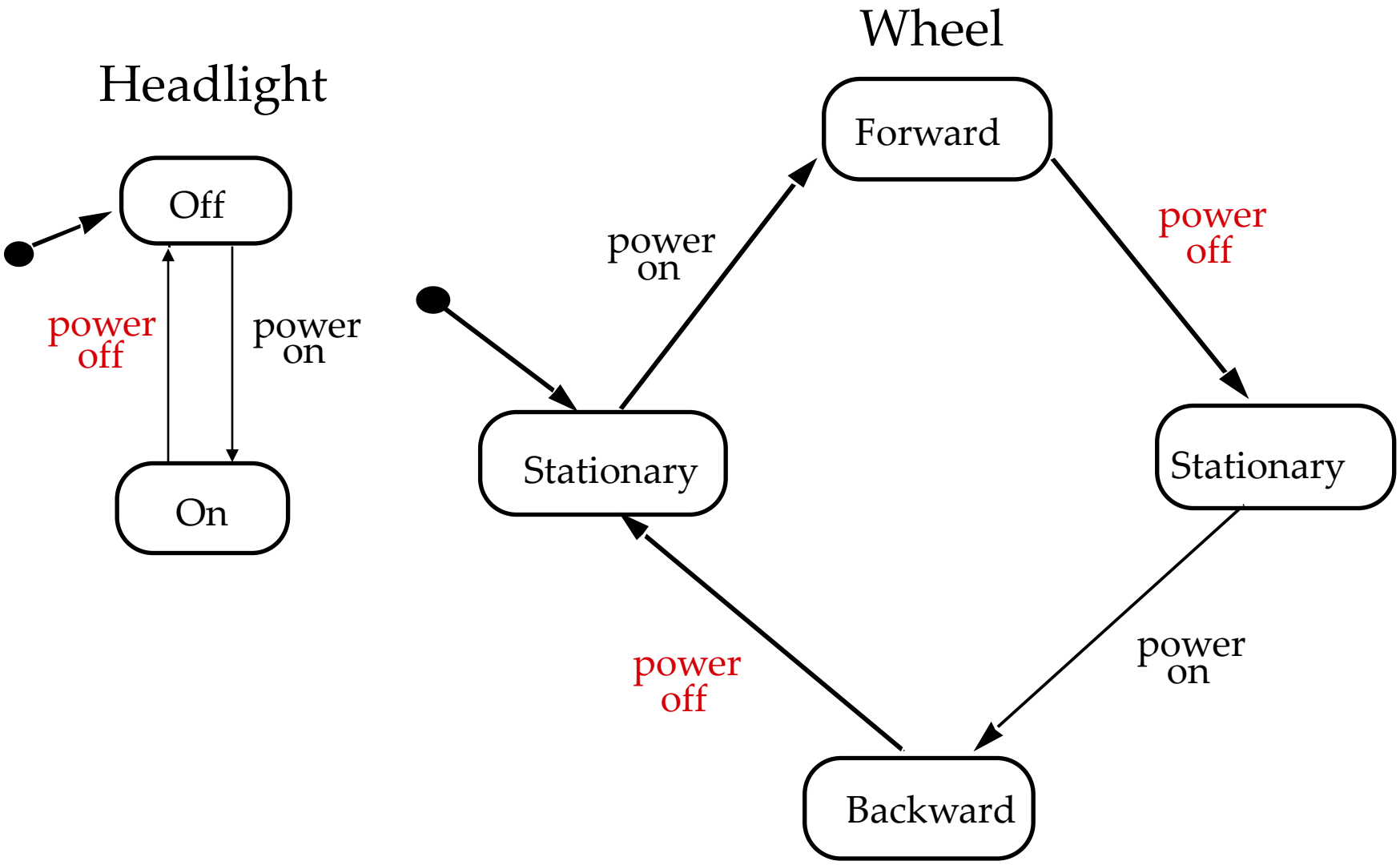
Find the Dynamic Model: Create sequence diagram

- Name: Drive Car
- Sequence of events:
  - Billy turns power on
  - Headlight goes on
  - Wheels starts moving forward
  - Wheels keeps moving forward
  - Billy turns power off
  - Headlight goes off
  - Wheels stops moving
  - . . .

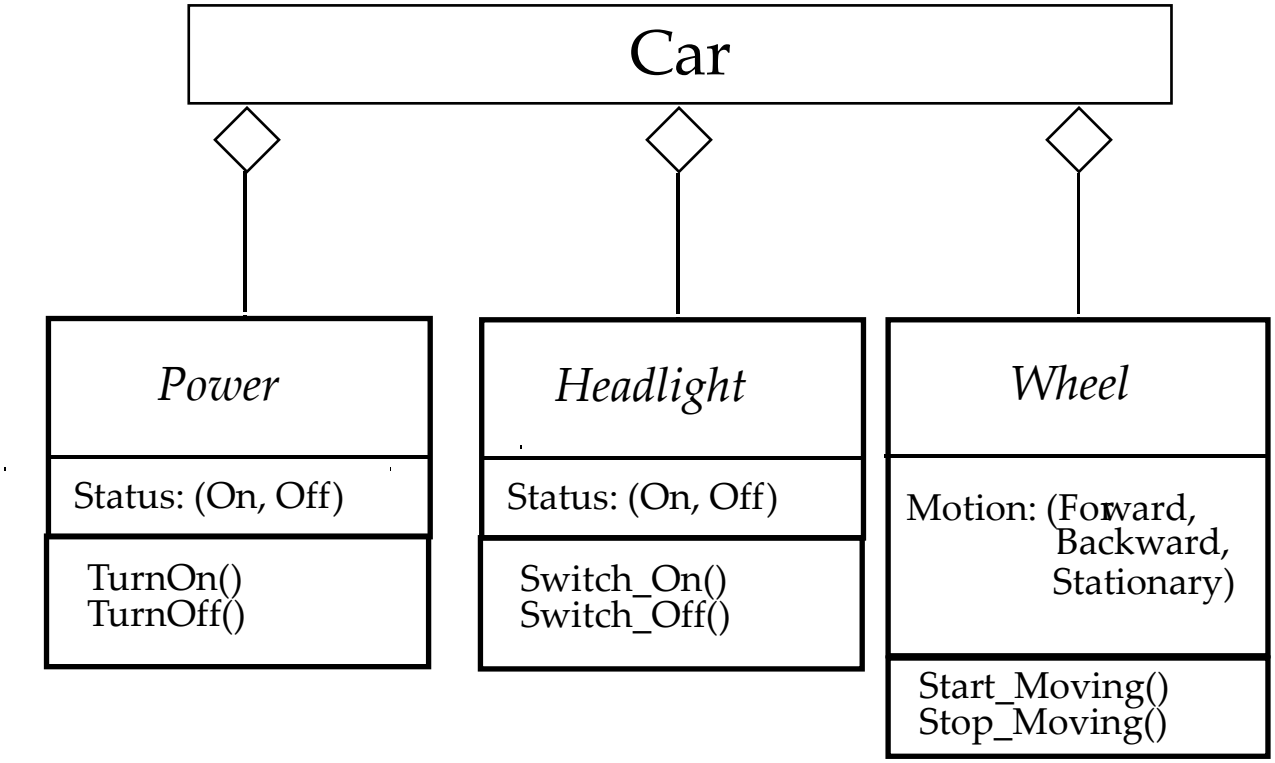
Sequence Diagram for Drive Car Scenario



Toy Car: Dynamic Model



Toy Car: Object Model



Summary of dynamic models

- In this lecture, we reviewed the construction of the dynamic model from use case and object models. In particular, we described:
- Sequence and statechart diagrams for identifying new classes and operations.
- Activity diagrams for describing complex business rules/logic inside operations.
- In addition, we described the requirements analysis document and its components.