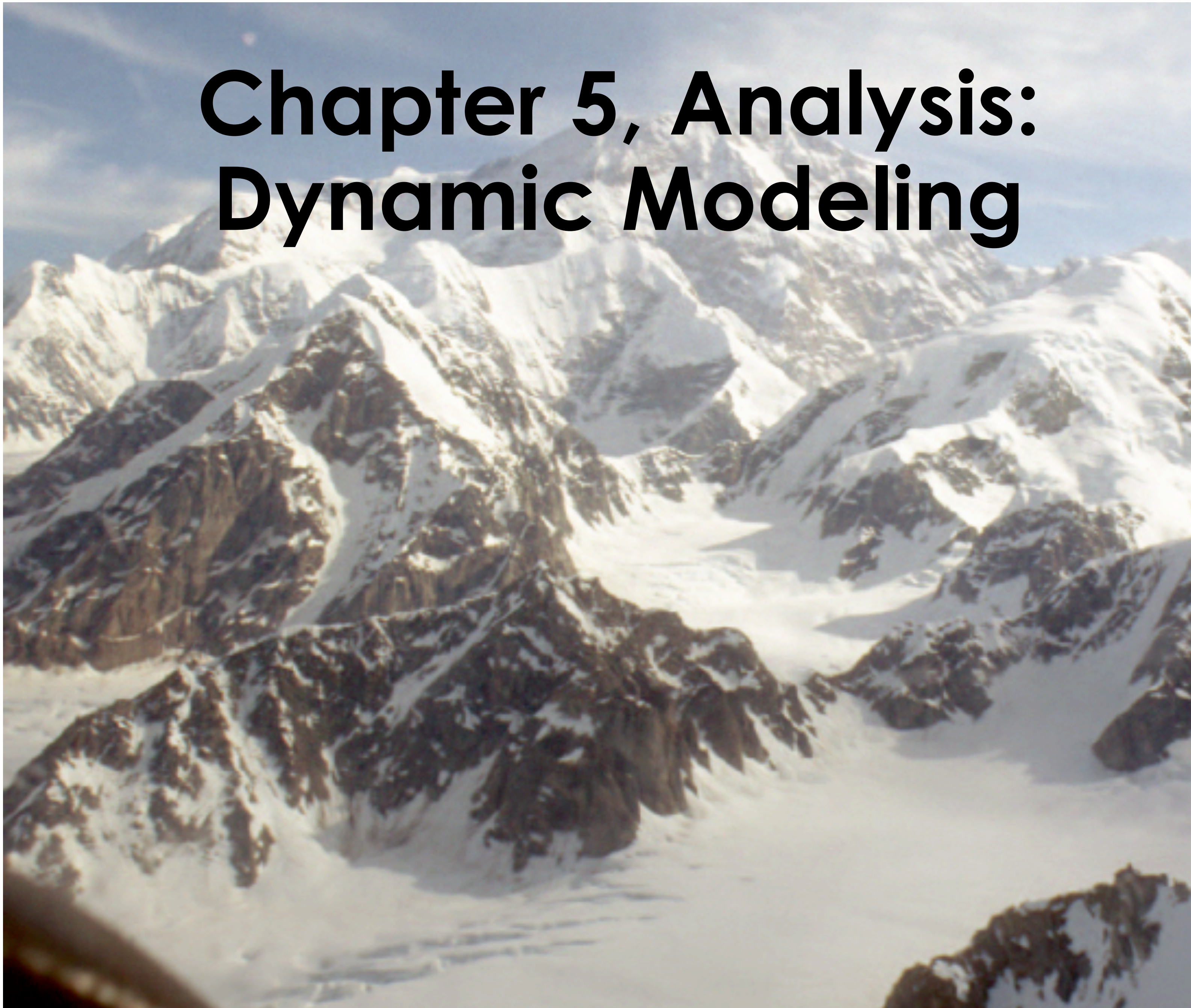
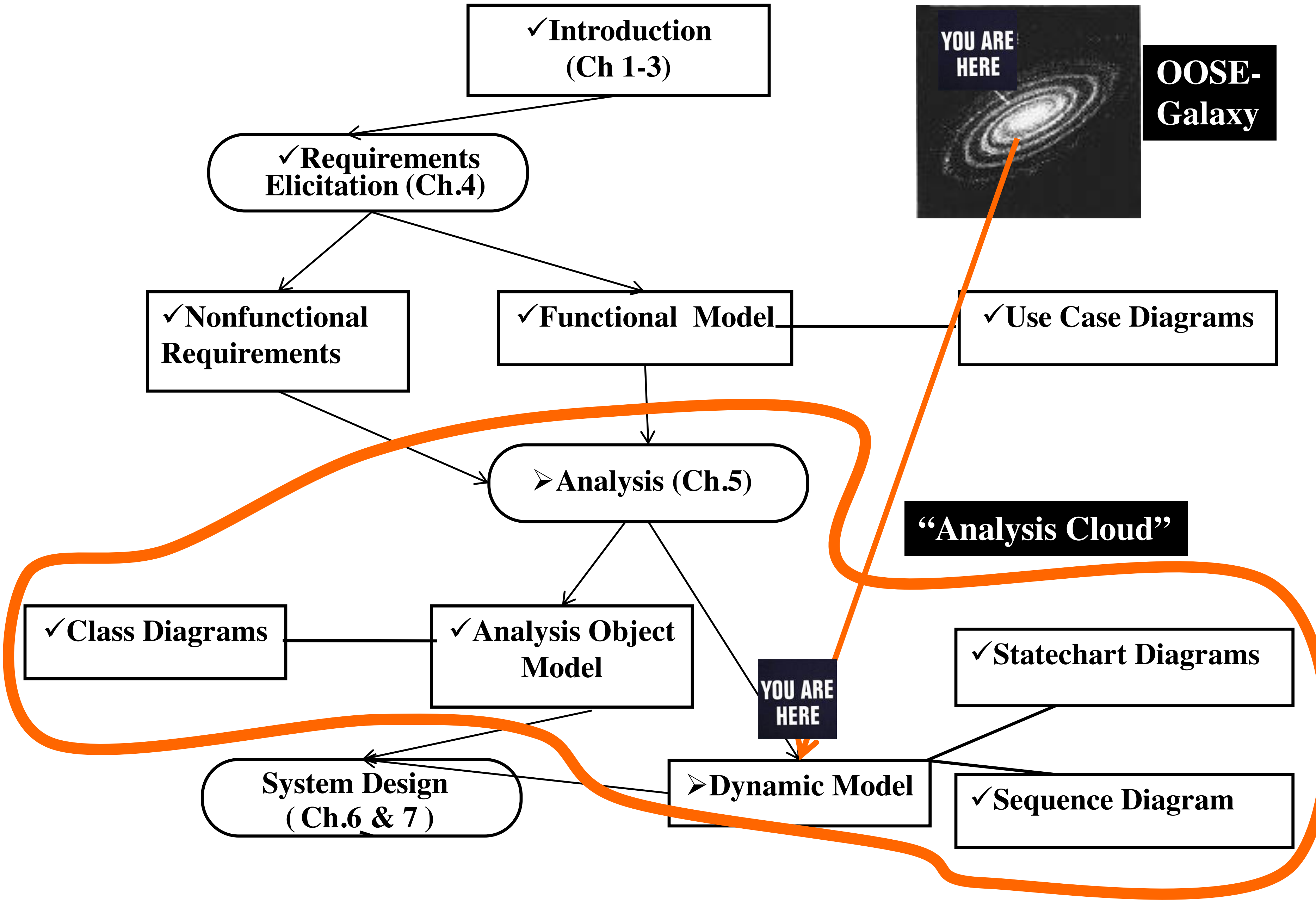
Chapter 5, Analysis: Dynamic Modeling

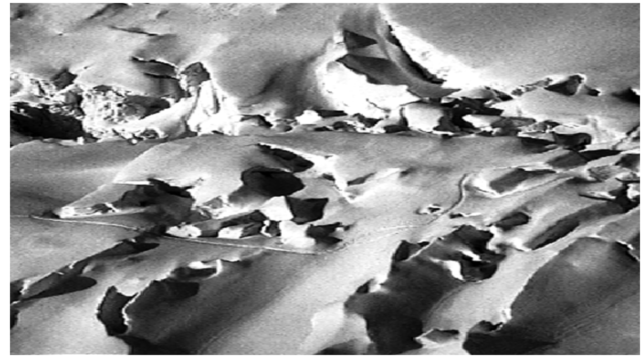


I cannot follow the lectures. Where are we?

- We have covered Ch 1 – 4
- We are in the middle of Chapter 5
  - Review functional and structural modeling and the corresponding UML notations:
    - Read again Ch 3, pp. 43 – 61, Ch 5.3
- From use cases to class diagrams
  - Identify participatory objects in flow of events descriptions
    - Exercise: Apply Abbot's technique to Fig. 5-7, p. 175
  - Identify entity, control and boundary objects
    - Heuristics to find these types: Ch 5, Section 5.4
- We are now moving into dynamic modeling
- Notations for dynamic models are
  - Interaction-, Communication-, Statechart-, Activity diagrams, Reread Ch. 2, pp. 62-68


✓Introduction
(Ch 1-3)
YOU ARE HERE
OOSE-Galaxy
✓Requirements Elicitation (Ch.4)
✓Nonfunctional Requirements
✓Functional Model
✓Use Case Diagrams
➢Analysis (Ch.5)
"Analysis Cloud"
✓Class Diagrams
✓Analysis Object Model
YOU ARE HERE
✓Statechart Diagrams
System Design
(Ch.6 & 7)
➢Dynamic Model
✓Sequence Diagram

Ways to Go



System design
(Ch. 6 & 7)

Subsystem Decomposition

Design Goals

Object design
(Ch. 8 & 9)

Class Diagram

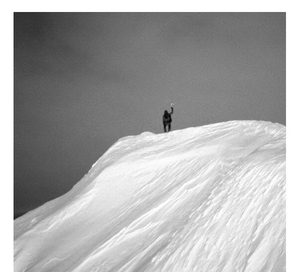
Object Design Model

Implementation
(Ch. 10)

Source Code

Testing (Ch. 11)

Deliverable System



Outline of the Lecture

- Dynamic modeling
  - Interaction Diagrams (Choreography)
    - Sequence diagrams
    - Communication diagrams
    - BPMN & DCR Choreographies
  - State diagrams
  - DCR Graphs
- Requirements analysis model validation
- Analysis Example

Dynamic Modeling with UML

- Two UML diagrams types for describing dynamic models:
  - Statechart diagrams describe the dynamic behavior of a *single object (Also called process models)*
  - Interaction diagrams describe the dynamic behavior *between objects. (Also called choreographies)*

UML State Chart Diagram

- State Chart Diagram
  - A *notation* for a state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events)
- State Machine
  - A *model* of behavior composed of a finite number of states, transitions between those states, and actions
- Moore Machine
  - A special type of state machine, where the output depends only on the state
- Mealy Machine
  - A special type of state machine where the output depends on the condition, event, action of the transition and the state.

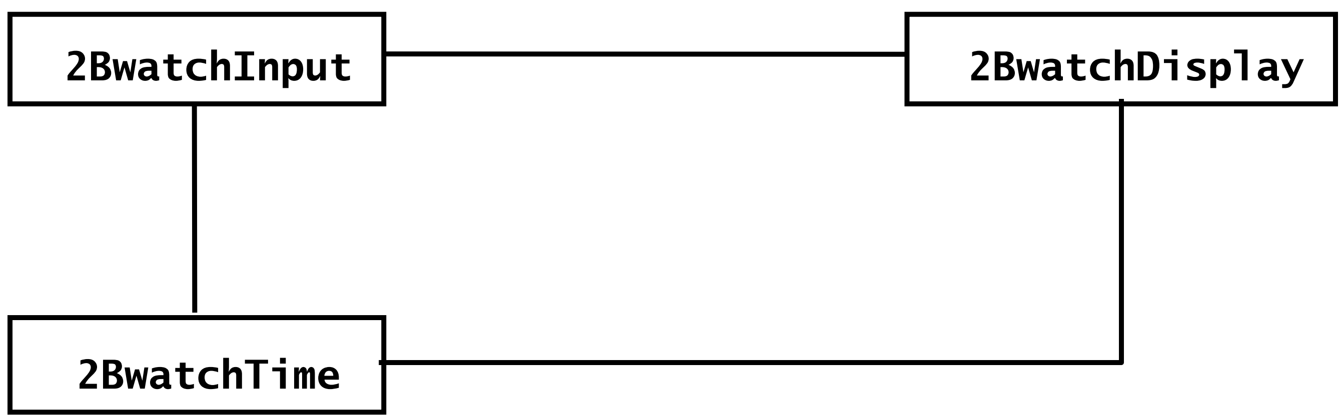
UML Interaction Diagrams

- Two types of interaction diagrams:
  - Communication Diagram:
    - Shows the temporal relationship among objects
    - Position of objects is identical to the position of the classes in the corresponding UML class diagram
    - Good for identifying the protocol between objects
    - Does not show time
  - Sequence Diagram:
    - Describes the dynamic behavior *between* several objects over time
    - Good for real-time specifications.

Example of a Communication Diagram

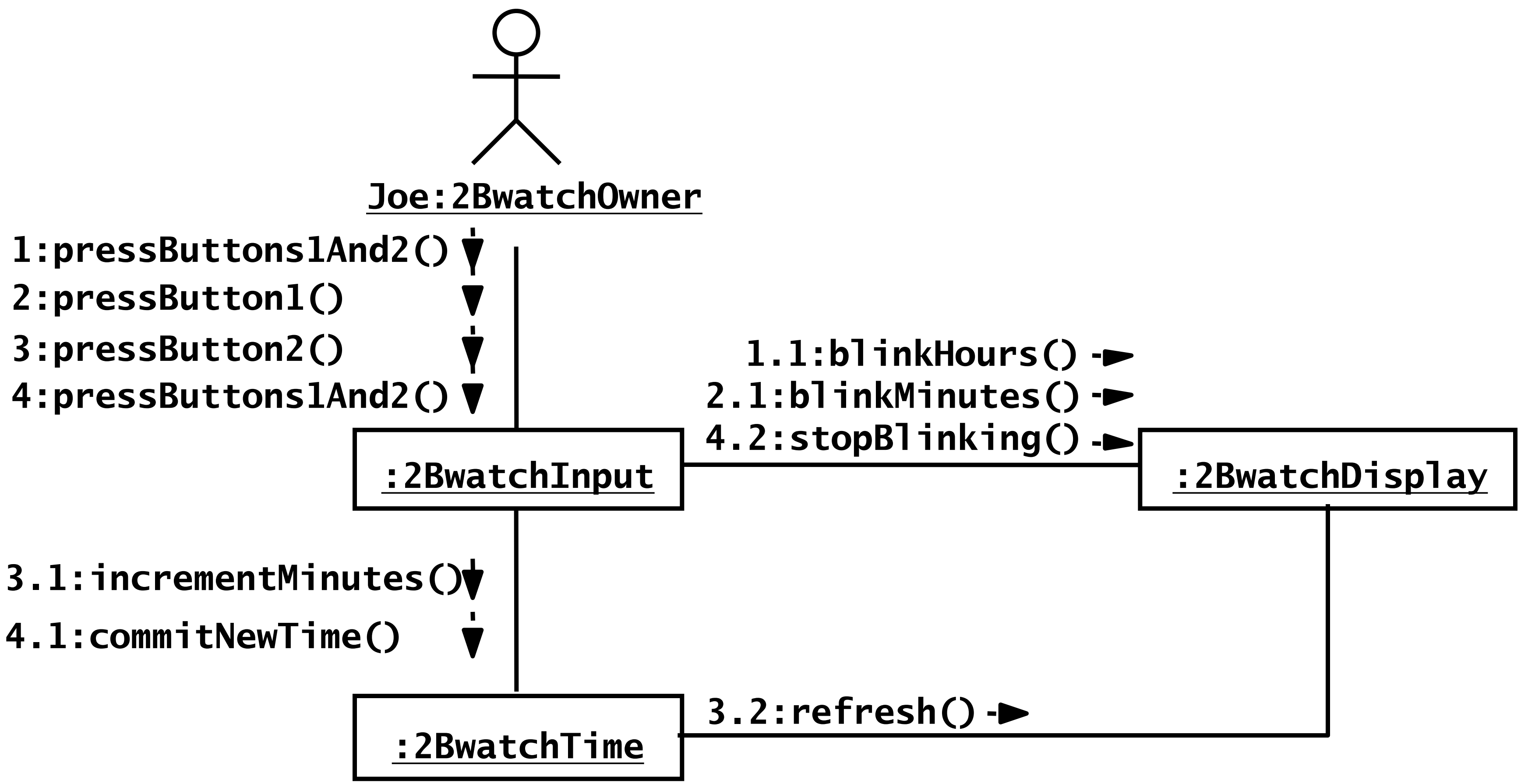
1) We start with the Class Diagram for 2Bwatch

2) Then we look at the sequence of events when Joe sets the time on 2Bwatch



Example of a Communication Diagram

Joe sets the time on 2Bwatch



Dynamic Modeling

- Definition of a dynamic model:
  - Describes the components of the system that have interesting dynamic behavior
- The dynamic model is described with
  - State diagrams: One state diagram for each class with interesting dynamic behavior
    - Classes without interesting dynamic behavior are not modeled with state diagrams
  - Sequence and communication diagrams: For the interaction between classes
- Purpose:
  - Identify new classes in the object model and supply operations for the classes.

Identify Classes and Operations

- We have already established several sources for class identification:
  - Application domain analysis: We find classes by talking to the client and identify abstractions by observing the end user
  - General world knowledge and intuition
  - Textual analysis of event flow in use cases (Abbot)
- Two additional heuristics for identifying classes from dynamic models:
  - Actions in state chart diagrams are candidates for public operations in classes
  - Activity lines in sequence diagrams are candidates for objects.

How do we detect Operations?

- We look for objects, who are interacting and extract their “protocol”
- We look for objects, who have interesting behavior on their own
- Good starting point: Flow of events in a use case description
- From the flow of events we proceed to the sequence diagram to find the participating objects.

How do we detect Operations?

- We look for objects, who are interacting and extract their "protocol"
- We look for objects, who have interesting behavior on their own
- Good starting point: Flow of events in a use case description
- From the flow of events we proceed to the sequence diagram to find the participating objects.

What is an Event?

- Something that happens at a point in time
- An event sends information from one object to another
- Events can have associations with each other:
  - Causally related:
    - An event happens always before another event
    - An event happens always after another event
  - Causally unrelated:
    - Events that happen concurrently
- Events can also be grouped in event classes with a hierarchical structure => Event taxonomy.

The term ‘Event’ is often used in two ways

- Instance of an event class:
  - “Slide 14 shown on Friday November 27 at 9:23”
  - Event class “Lecture Given”, Subclass “Slide Shown”
- Attribute of an event class
  - Slide Update(8:55 AM, 11/27/2009)
  - Train_Leaves(4:45pm, Manhattan)
  - Mouse button down(button#, tablet-location).

Finding Participating Objects

- Heuristic for finding participating objects:
  - A event always has a sender and a receiver
  - Find the sender and receiver for each event => These are the objects participating in the use case.

An Example

- Flow of events in "Get SeatPosition" 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?

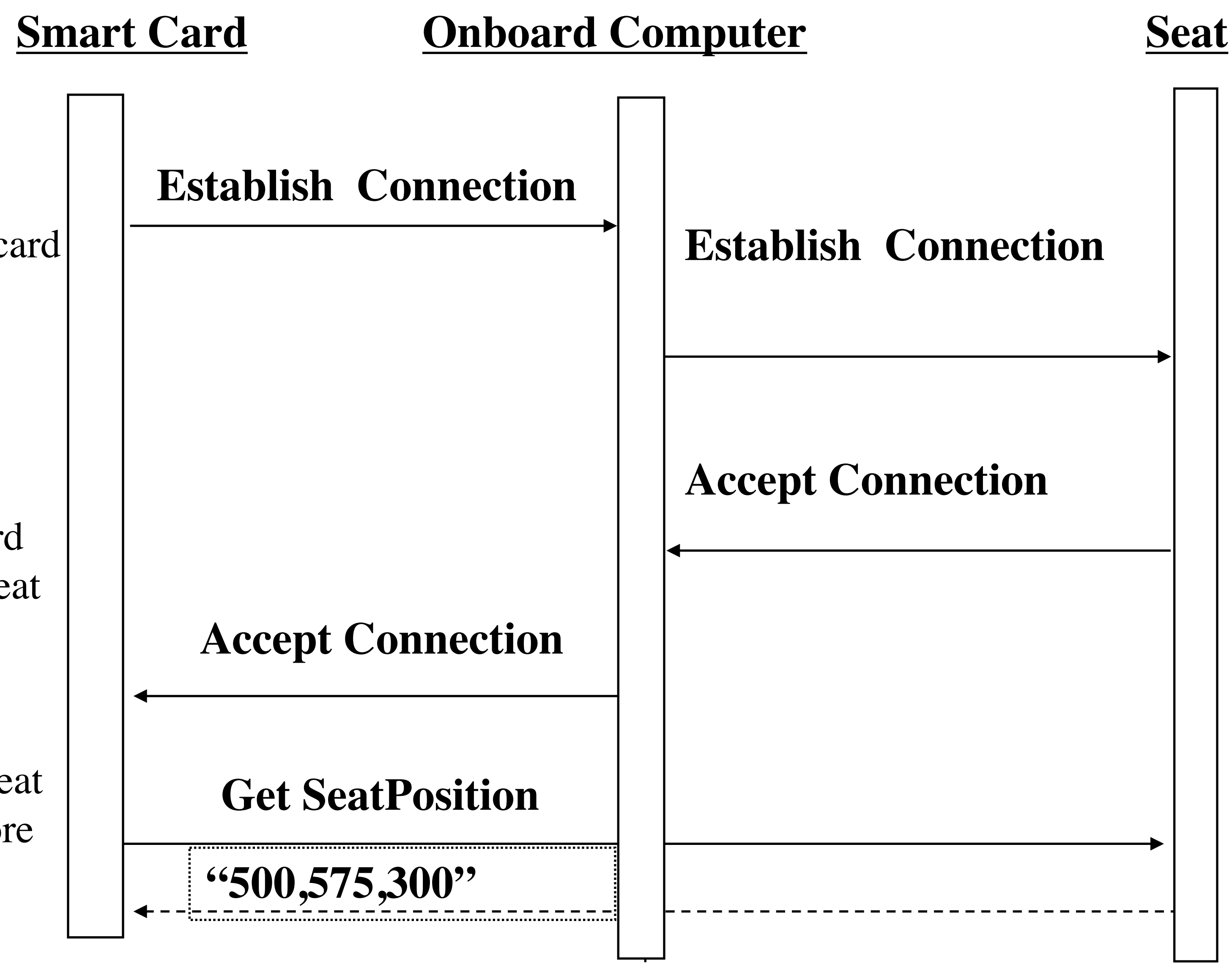
Sequence Diagram for "Get SeatPosition"

1. Establish connection between smart card and onboard computer

2. Establish connection between onboard computer and seat (actually seat sensor)

3. Get current seat position and store on smart card.

time



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

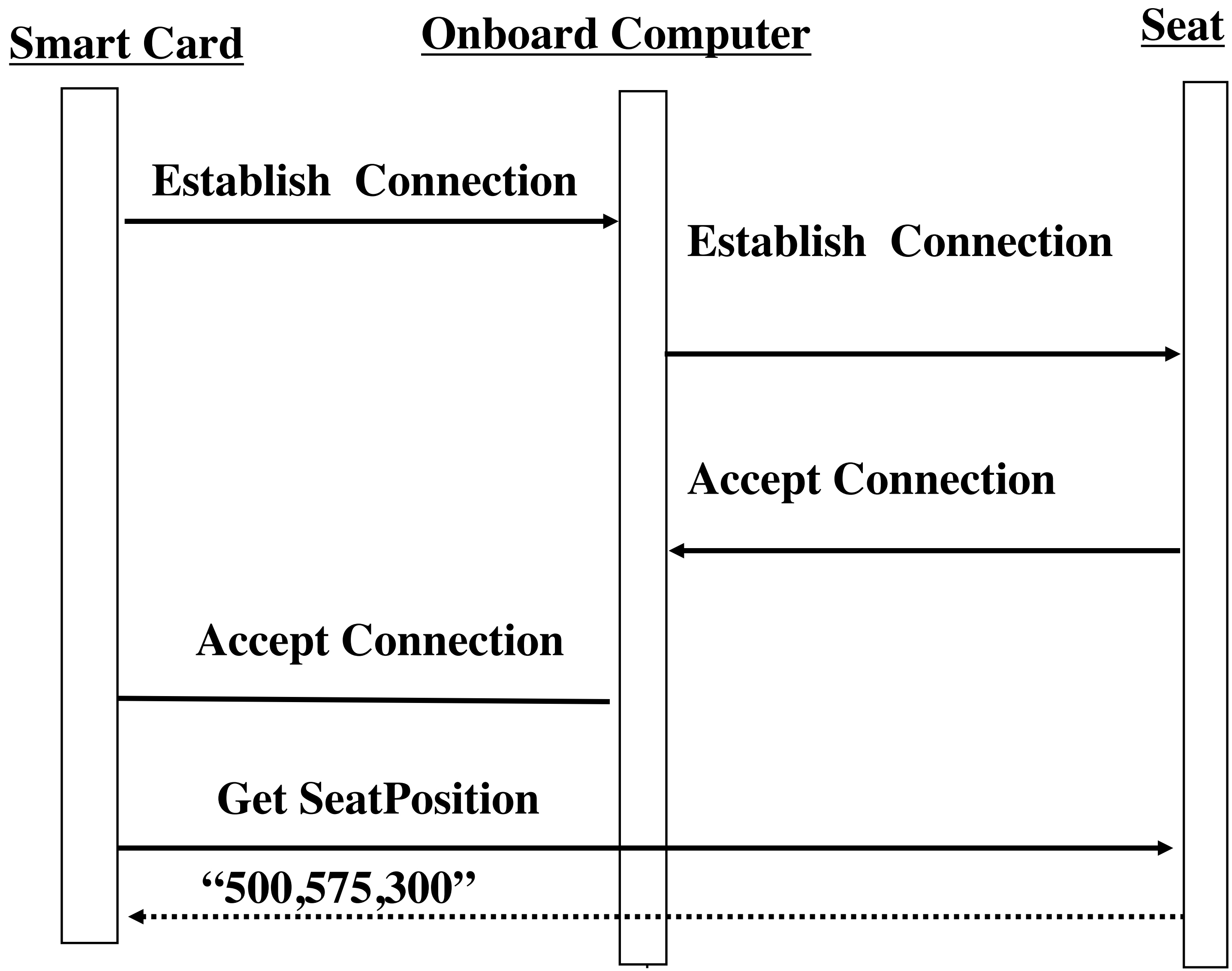
- Creation of objects:
  - Create control objects at beginning of event flow
  - The control objects create the boundary objects
- Access of objects:
  - Entity objects can be accessed by control and boundary objects
  - Entity objects should not access boundary or control objects.

Is this a good Sequence Diagram?

The first column is not an actor

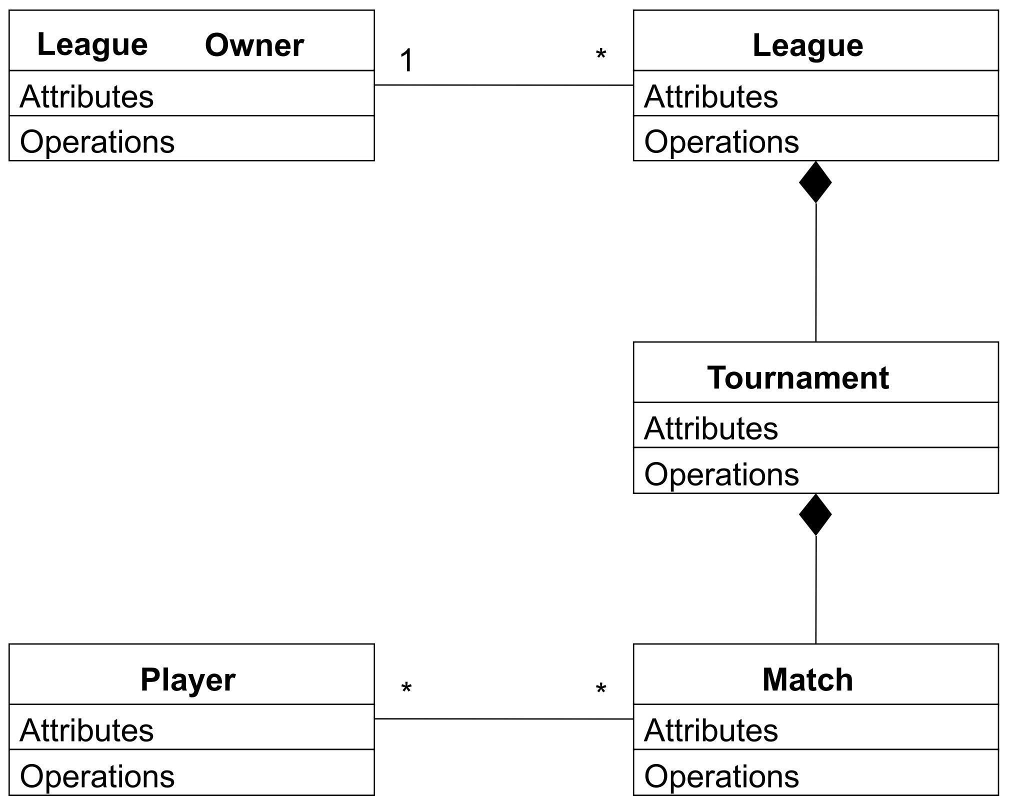
It is not clear where the boundary object is

It is not clear where the control object is



Another Example: Finding Objects from a Sequence Diagram

- Let's assume ARENA's object model contains - at this modeling stage – the following six objects
  - League Owner, League, Tournament, Match and Player

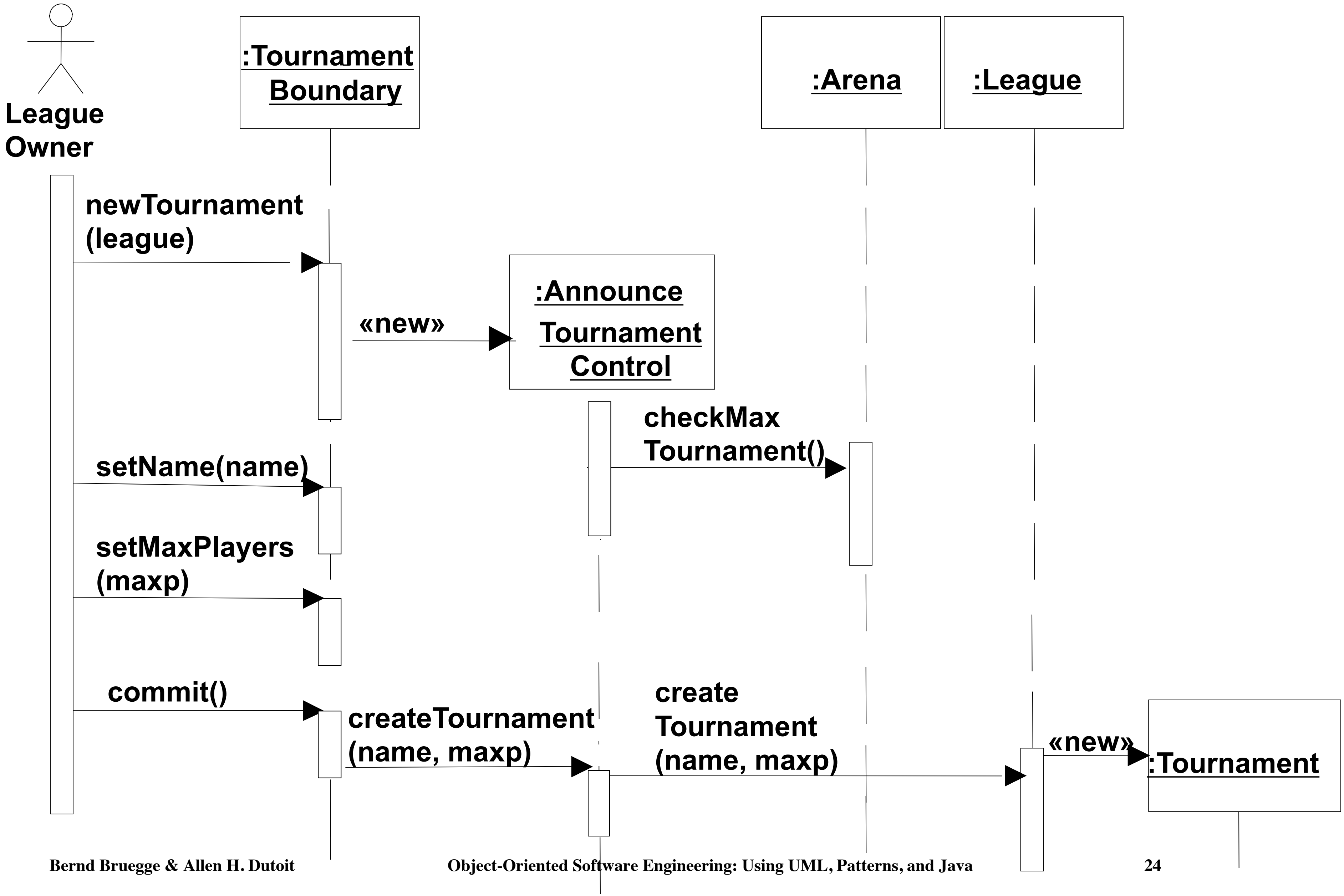


Another Example: Finding Objects from a Sequence Diagram

- Let’s assume ARENA’s object model contains - at this modeling stage – the following six objects
  - League Owner, League, Tournament, Match and Player

- We now model the use case **CreateTournament** with a sequence diagram

ARENA Sequence Diagram: Create Tournament

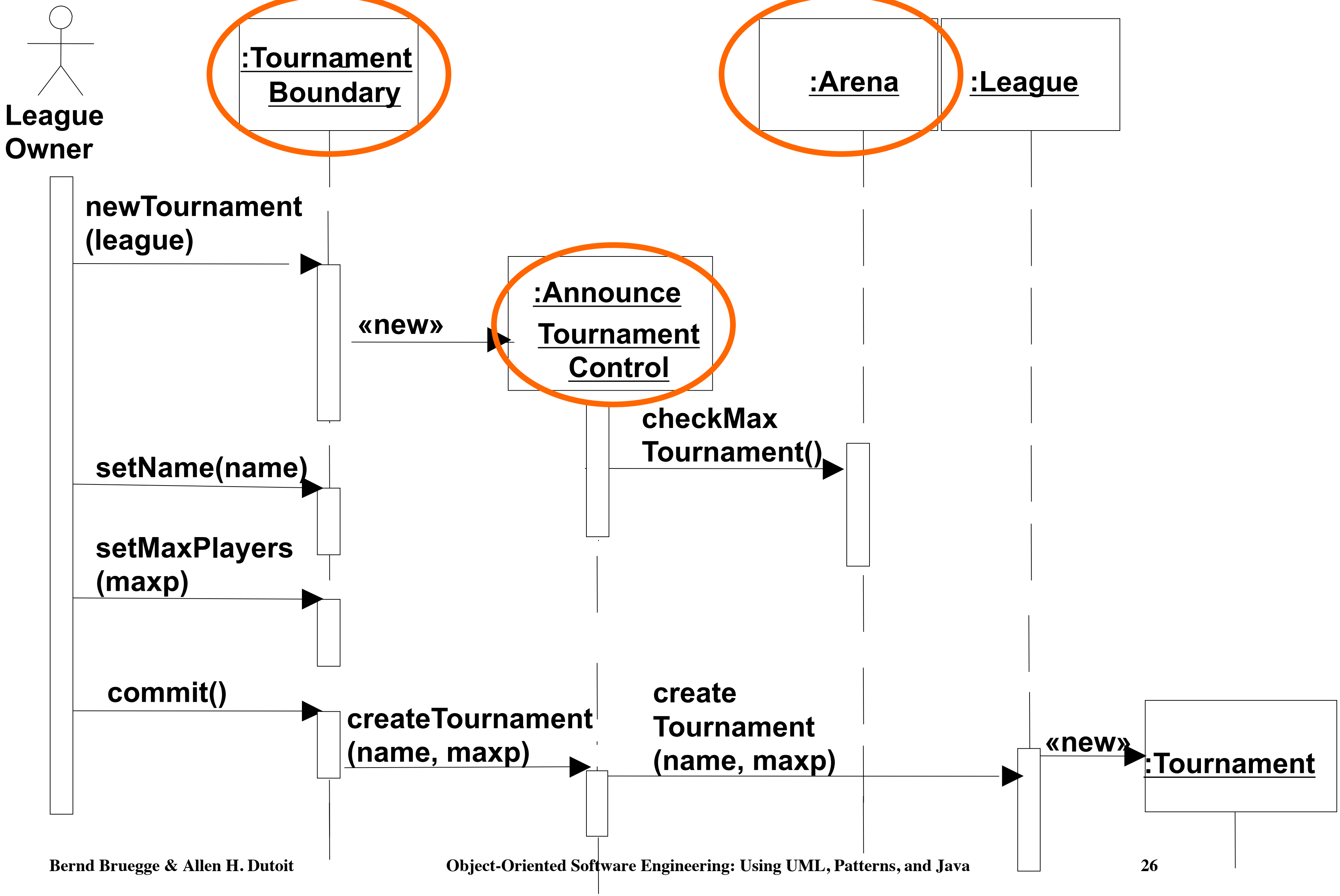


Another Example: Finding Objects from a Sequence Diagram

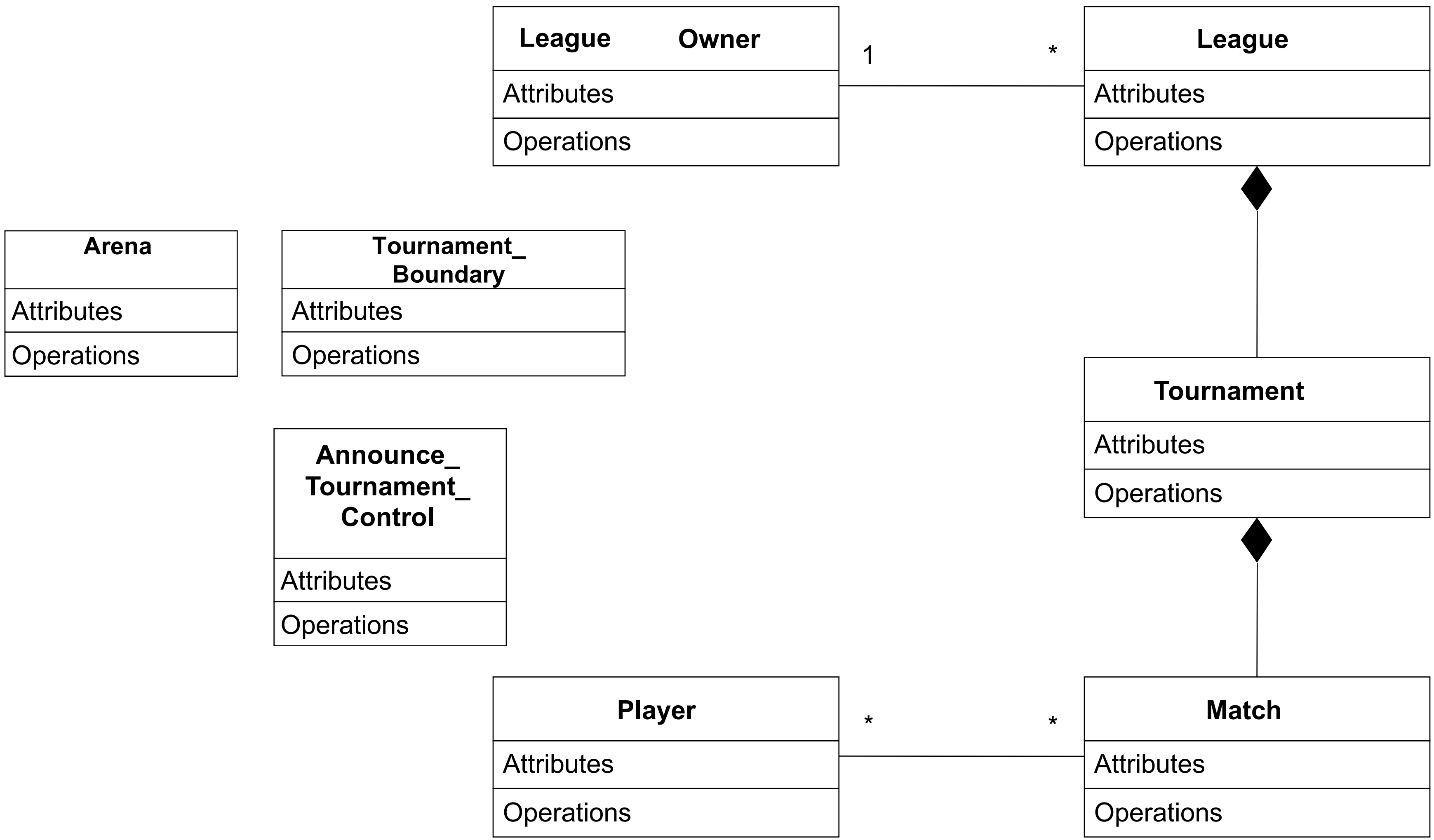
The Sequence Diagram identified 3 new Classes

- Tournament Boundary, Announce_Tournament_Control and Arena

ARENA Sequence Diagram: Create Tournament



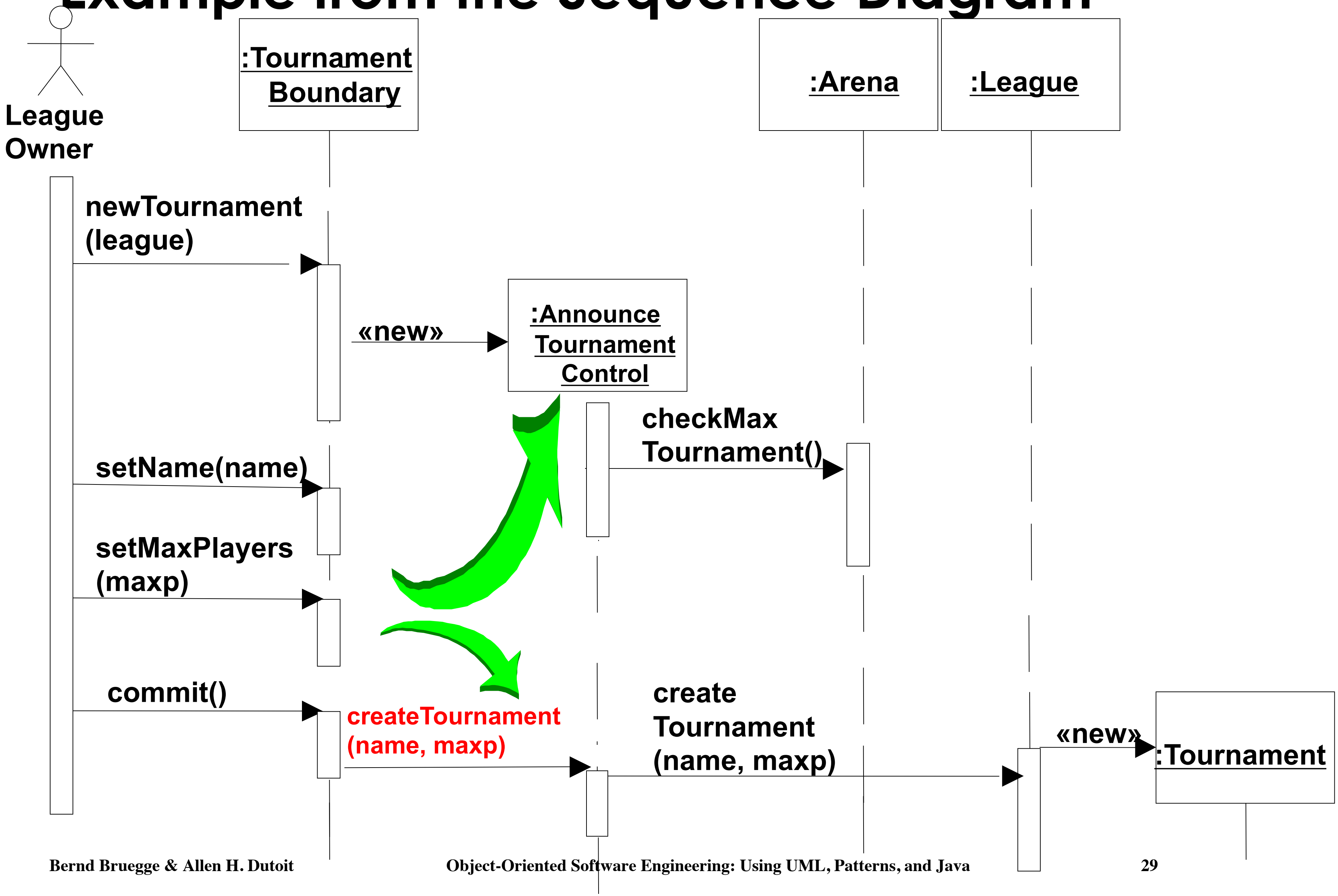
Impact on Arena's Object Model

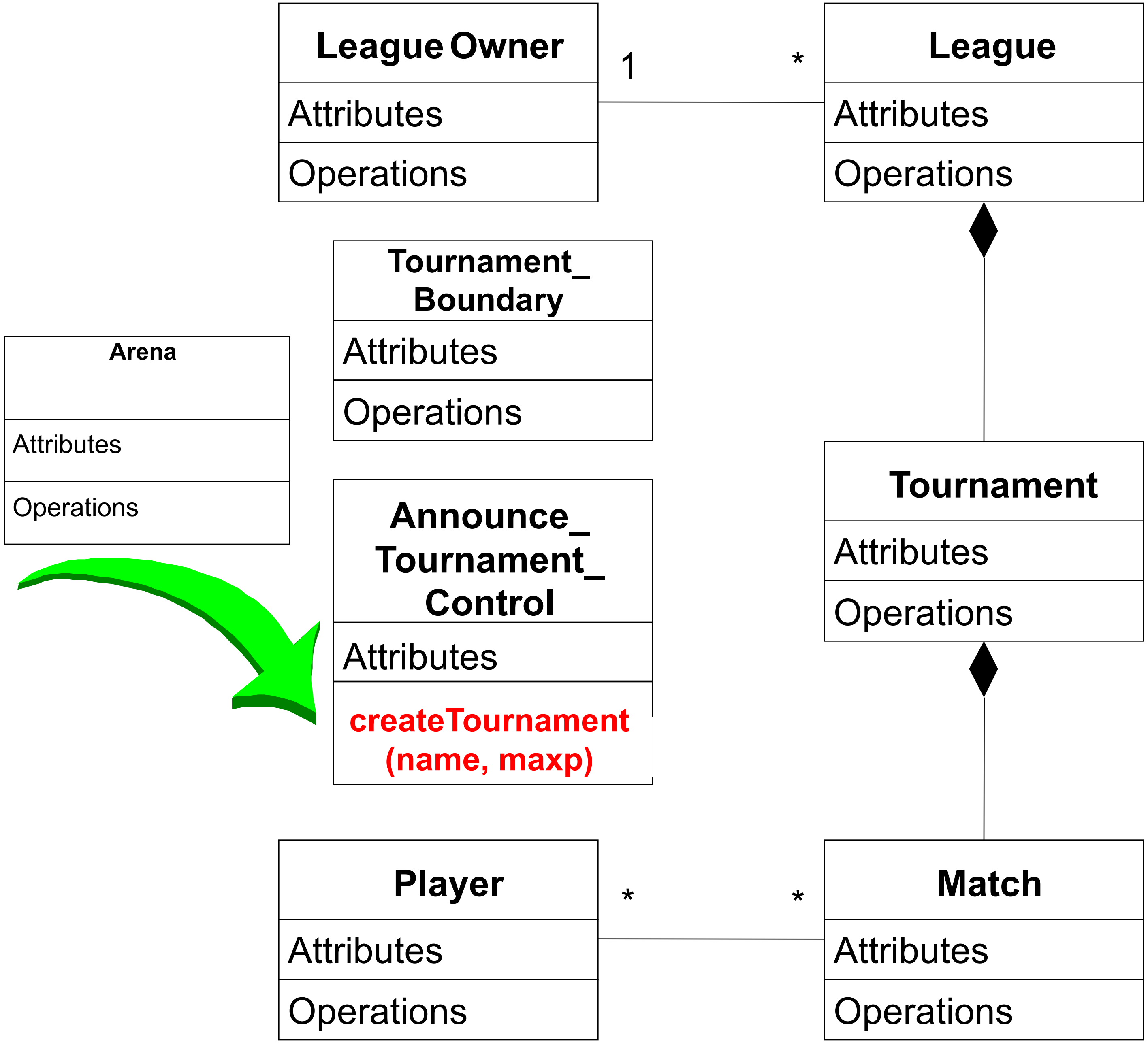


Impact on ARENA's Object Model (2)

- The sequence diagram also supplies us with many new events
  - newTournament(league)
  - setName(name)
  - setMaxPlayers(max)
  - commit
  - checkMaxTournament()
  - createTournament
- Question:
  - Who owns these events?
- Answer:
  - For each object that receives an event there is a public operation in its associated class
  - The name of the operation is usually the name of the event.

Example from the Sequence Diagram




League Owner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament_
Boundary
Attributes
Operations
Arena
Attributes
Operations
Announce_
Tournament_
Control
Attributes
createTournament
(name, maxp)
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

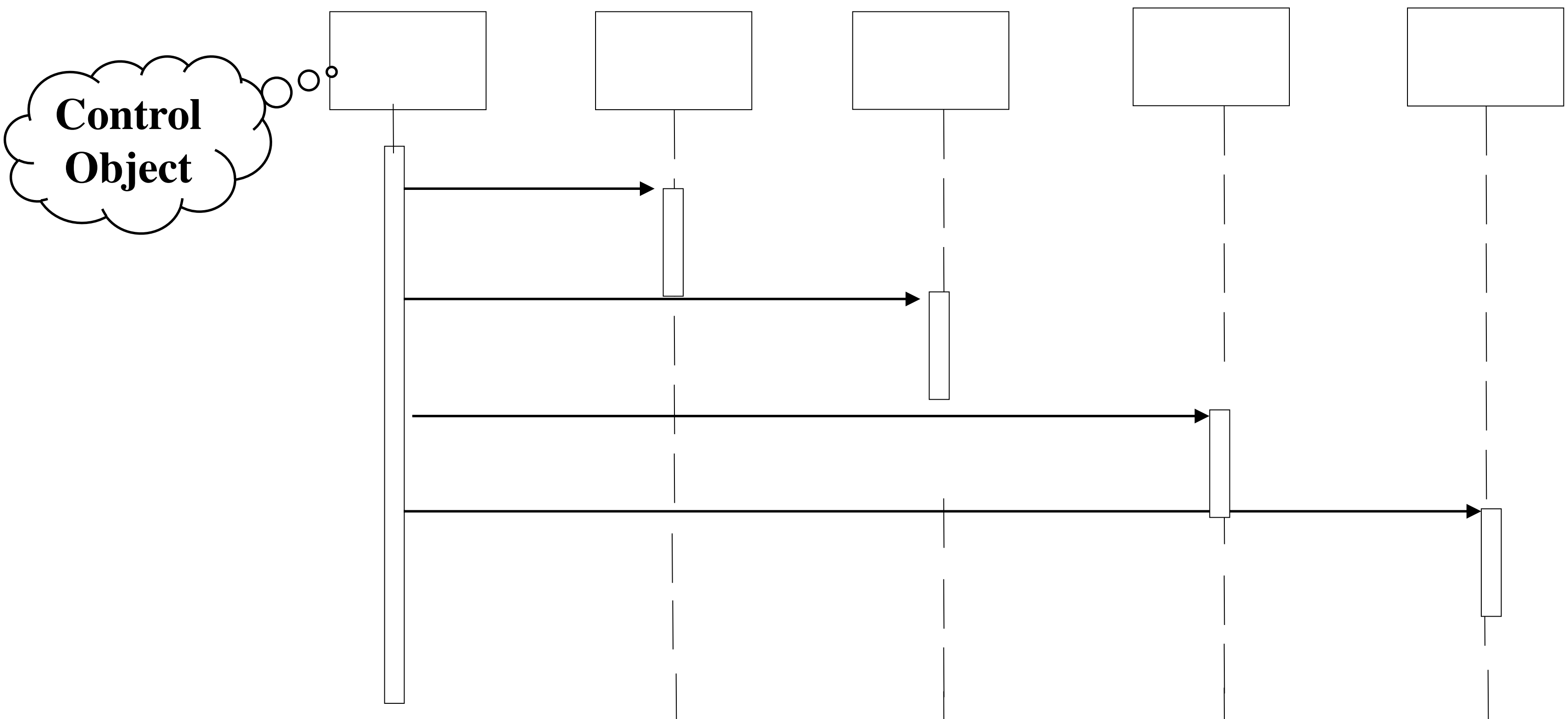
What else can we get out of Sequence Diagrams?

- Sequence diagrams are derived from use cases

- The structure of the sequence diagram helps us to determine how decentralized the system is
- We distinguish two structures for sequence diagrams
  - Fork Diagrams and Stair Diagrams (Ivar Jacobsen)

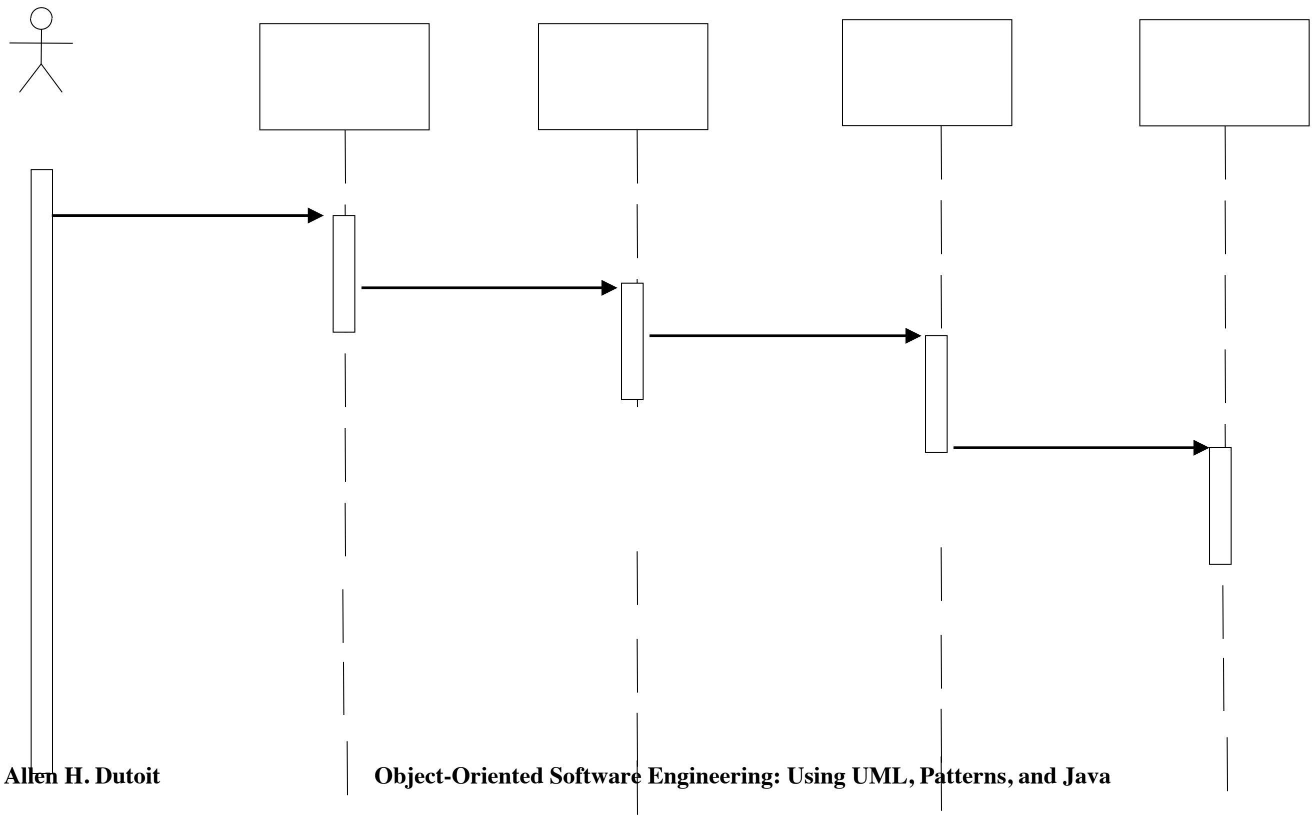
Fork Diagram

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



Stair Diagram

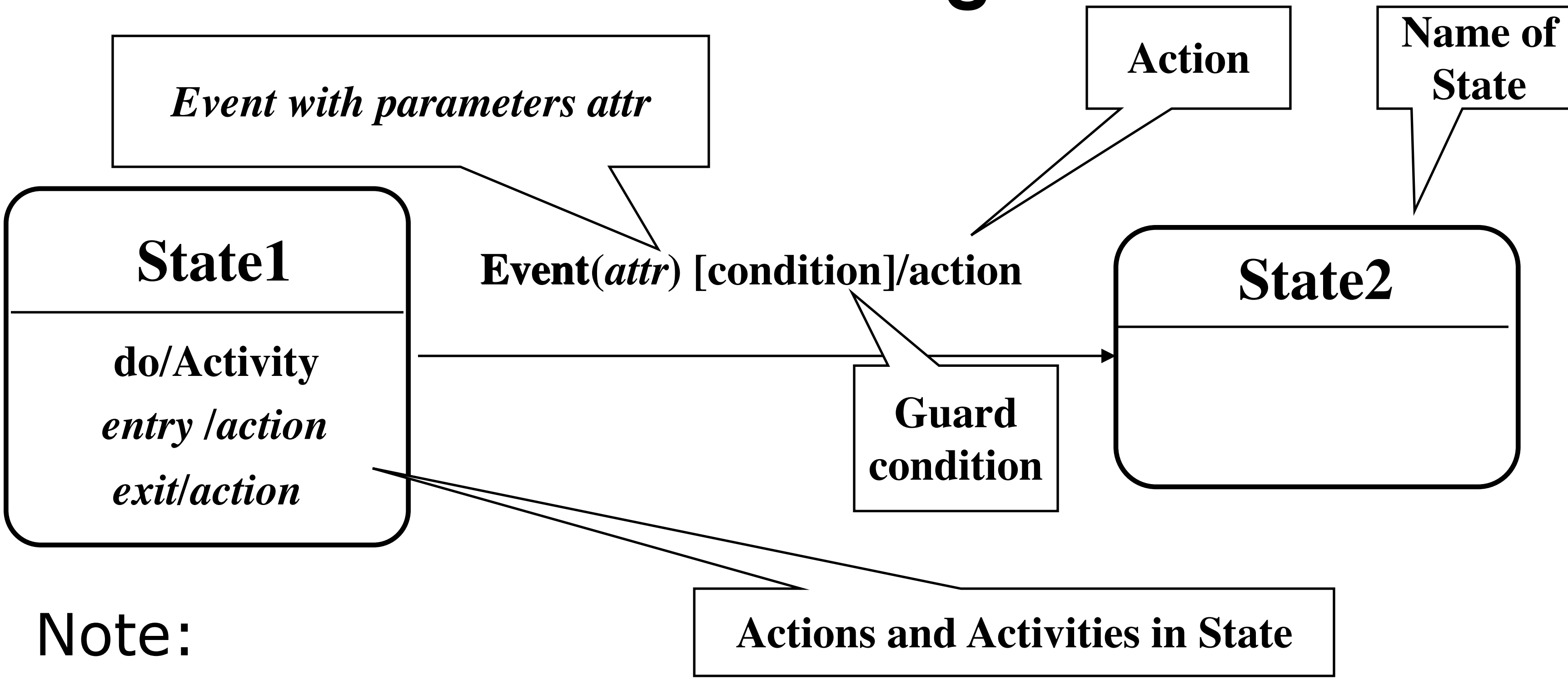
- The dynamic behavior is distributed. Each object delegates responsibility to other objects
  - Each object knows only a few of the other objects and knows which objects can help with a specific behavior



Fork or Stair?

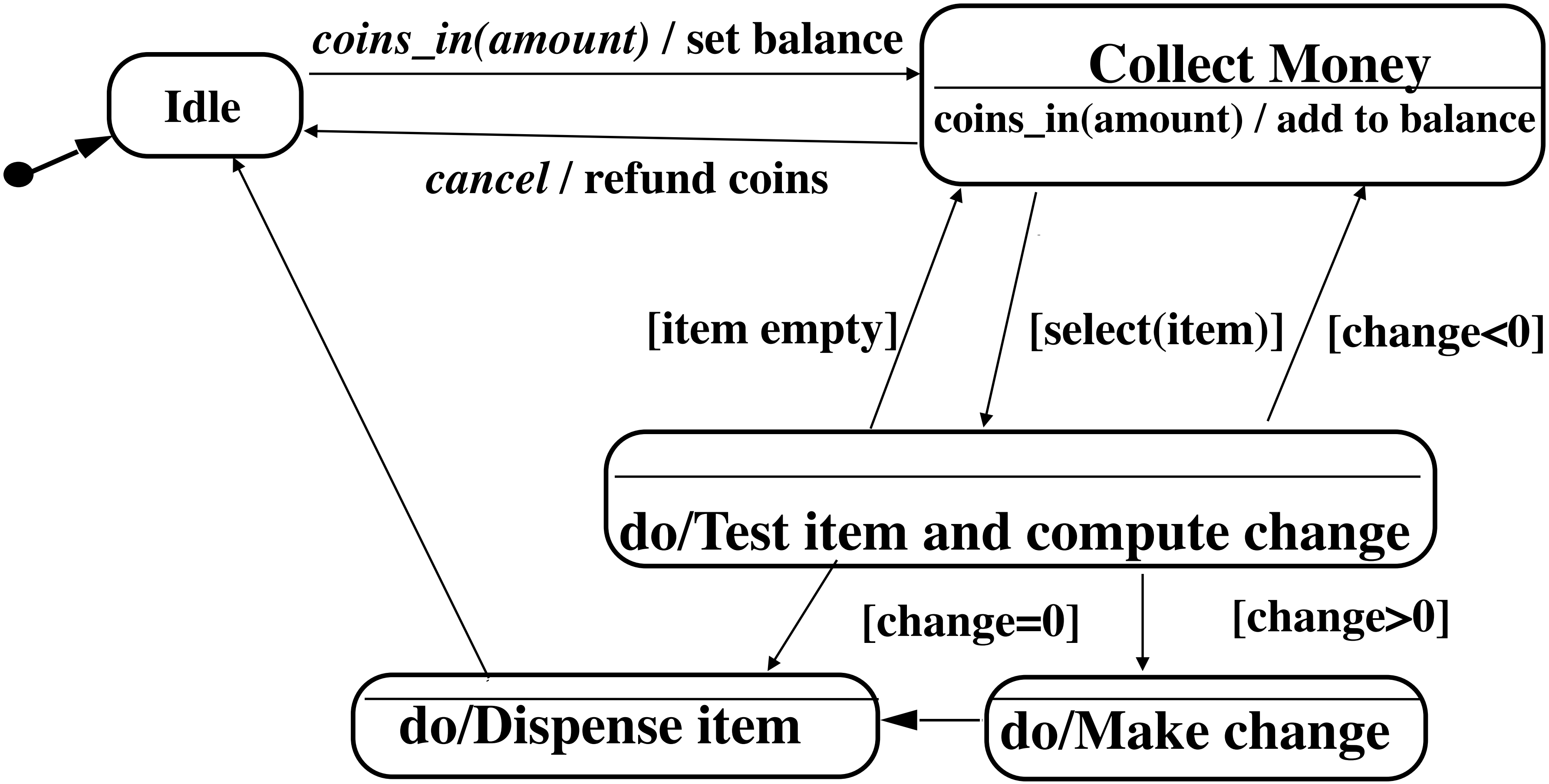
- Object-oriented supporters claim that the stair structure is better
- Modeling Advice:
  - Choose the stair - a decentralized control structure - if
    - The operations have a strong connection
    - The operations will always be performed in the same order
  - Choose the fork - a centralized control structure - if
    - The operations can change order
    - New operations are expected to be added as a result of new requirements.

Review: UML Statechart Diagram Notation



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

Example of a StateChart Diagram



State

- 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.

State Chart Diagram vs Sequence Diagram

- State chart diagrams help to identify:
  - Changes to an individual object over time

- Sequence diagrams help to identify:
  - The temporal relationship of between objects over time
  - Sequence of operations as a response to one ore more events.

Dynamic Modeling of User Interfaces

- Statechart diagrams can be used for the design of user interfaces
- States: Name of screens
- Actions are shown as bullets under the screen name

Navigation Path Example

Screen name

Action

Diagnostics Menu
- User moves cursor to Control Panel or Graph

Control panel
- User selects functionality of sensors

Define
- User defines a sensor event from a list of events

Enable
- User can enable a sensor event from a list of sensor events

Disable
- User can disable a sensor event from a list of sensor events

Graph
- User selects data group and type of graph

Selection
- User selects data group
  - Field site
  - Car
  - Sensor group
  - Time range

Practical Tips for Dynamic Modeling

- Construct dynamic models only for classes with significant dynamic behavior
  - Avoid “analysis paralysis”
- Consider only relevant attributes
  - Use abstraction if necessary
- Look at the granularity of the application when deciding on actions and activities
- Reduce notational clutter
  - Try to put actions into superstate boxes (look for identical actions on events leading to the same state).

Outline of the Lecture

- Dynamic modeling
  - Interaction Diagrams
    - Sequence diagrams
    - Communication diagrams
  - State diagrams
- Requirements analysis model validation
- Analysis Example

Model Validation and Verification

- Verification is an equivalence check between the transformation of two models
- Validation is the comparison of the model with reality
  - Validation is a critical step in the development process Requirements should be validated with the client and the user.
  - Techniques: Formal and informal reviews (Meetings, requirements review)
- Requirements validation involves several checks
  - Correctness, Completeness, Ambiguity, Realism

Checklist for a Requirements Review

- Is the model correct?
  - A model is correct if it represents the client's view of the system
- Is the model complete?
  - Every scenario is described
- Is the model consistent?
  - The model does not have components that contradict each other
- Is the model unambiguous?
  - The model describes one system, not many
- Is the model realistic?
  - The model can be implemented

Verification vs Validation of models

$$
\begin{tikzcd}[column sep=huge, row sep=large]
R \arrow[r, "\text{Analysis}"] \arrow[d, "f_R"'] & M_{\text{Analysis}} \arrow[r, "\text{System Design}"] \arrow[d, "f_{MA}"'] & M_{\text{System}} \arrow[r, "\text{Object Design}"] \arrow[d, "f_{MS}"'] & M_{\text{Object}} \arrow[r, "\text{Implementation}"] \arrow[d, "f_{MD}"'] & M_{\text{Impl}} \arrow[d, "f_{Impl}"'] \\
R \arrow[r] & M_{\text{Analysis}} \arrow[r] & M_{\text{System}} \arrow[r] & M_{\text{Object}} \arrow[r] & M_{\text{Impl}} \\
{} & {} \arrow[l, "\text{Validation}"] & {} \arrow[l, "\text{Verification}"] & {} \arrow[l, "\text{Verification}"] & {} \arrow[l, "\text{Verification}"]
\end{tikzcd}
$$

$$
\begin{tikzcd}[column sep=huge, row sep=large]
M \arrow[r, "f_M"'] & M \\
R \arrow[u, "\mathrm{I}"] \arrow[r, "f_R"'] & R \arrow[u, "\mathrm{I}"']
\end{tikzcd}
$$

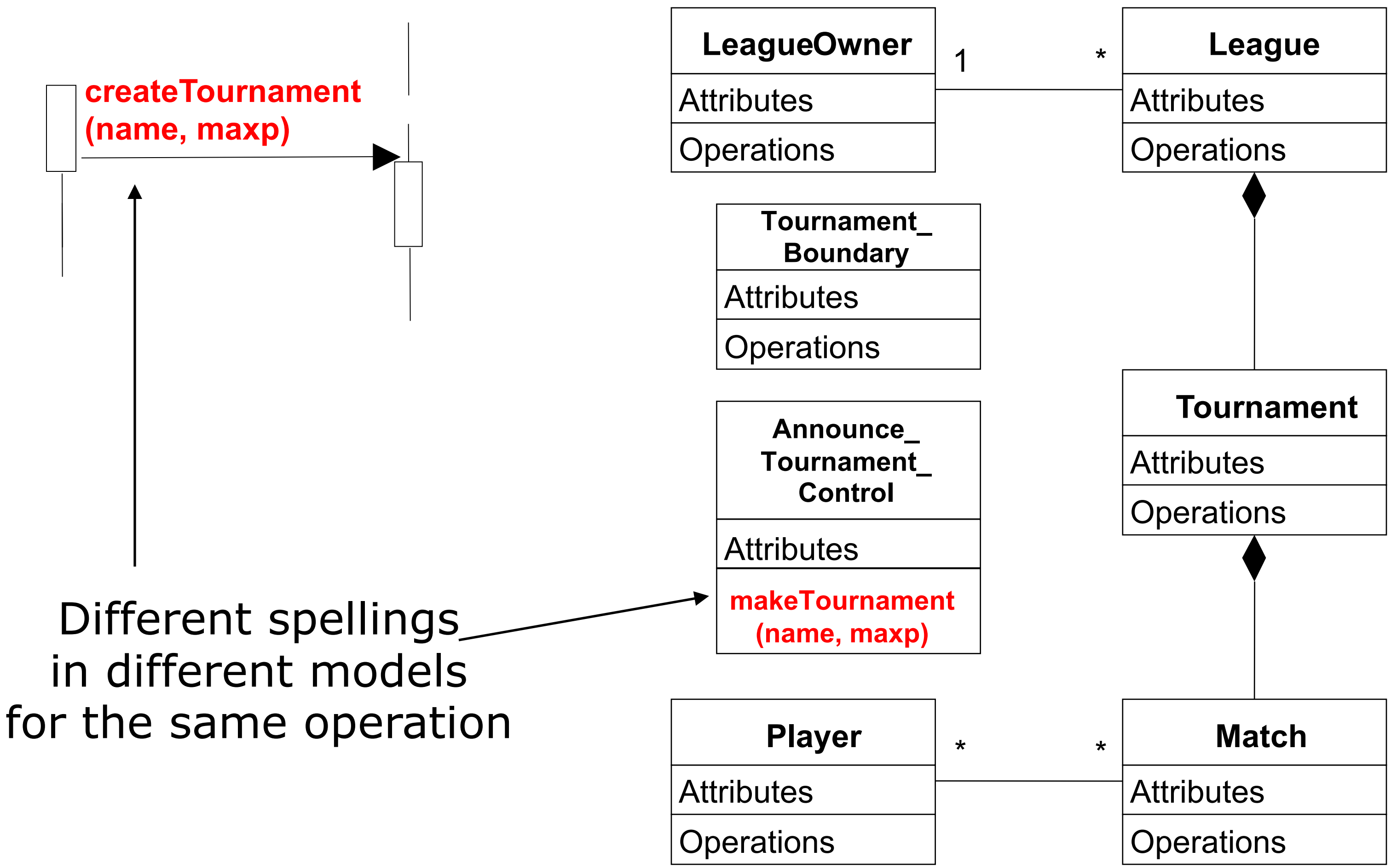
Examples for Inconsistency and Completeness Problems

- Different spellings in different UML diagrams

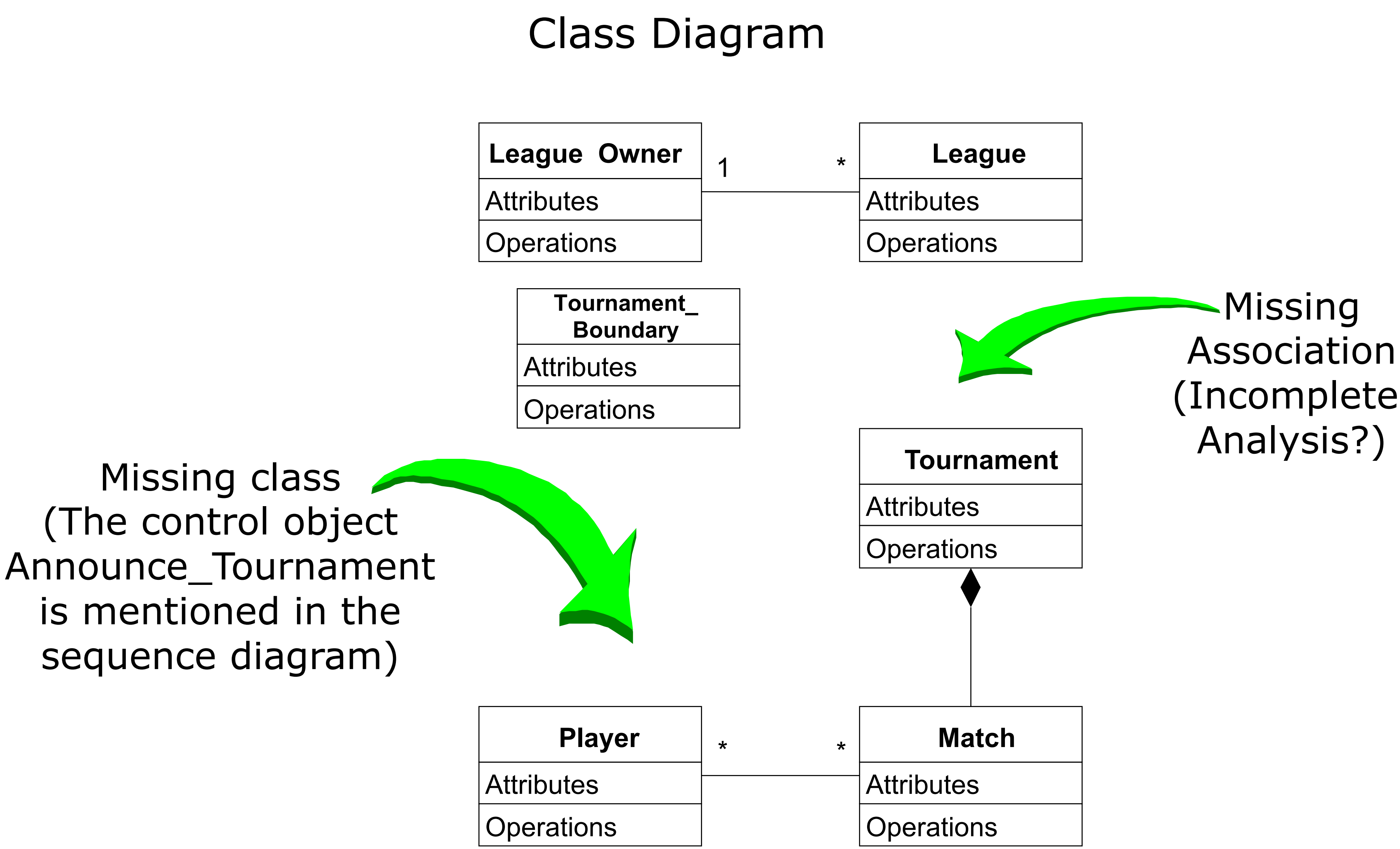
- Omissions in diagrams

Different spellings in different UML diagrams

UML Sequence Diagram UML Class Diagram



Omissions in some UML Diagrams



Checklist for a Requirements Review (2)

- Syntactical check of the models
- Check for consistent naming of classes, attributes, methods in different subsystems
- Identify dangling associations ("pointing to nowhere")
- Identify double- defined classes
- Identify missing classes (mentioned in one model but not defined anywhere)
- Check for classes with the same name but different meanings

When is a Model Dominant?

- Object model:
  - The system has classes with nontrivial states and many relationships between the classes
- Dynamic model:
  - The model has many different types of events: Input, output, exceptions, errors, etc.
- Functional model:
  - The model performs complicated transformations (eg. computations consisting of many steps)
- Which model is dominant in these applications?
  - Compiler
  - Database system
  - Spreadsheet program

Examples of Dominant Models

- Compiler:
  - The functional model is most important
  - The dynamic model is trivial because there is only one type input and only a few outputs
    - Is that true for development environments (e.g. Eclipse)?
- Database systems:
  - The object model most important
  - The functional model is trivial, because the purpose of the functions is to store, organize and retrieve data
- Spreadsheet program:
  - The functional model most important
  - The dynamic model is interesting if the program allows computations on a cell
  - The object model is trivial.

Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Nonfunctional requirements
   - 3.4 Constraints (“Pseudo requirements”)
   - 3.5 System models
     - 3.5.1 Scenarios
     - 3.5.2 Use case model
     - 3.5.3 Object model
       - 3.5.3.1 Data dictionary
       - 3.5.3.2 Class diagrams
     - 3.5.4 Dynamic models
     - 3.5.5 User interfae
4. Glossary

Section 3.5 System Models

3.5.1 Scenarios

- As-is scenarios, visionary scenarios

3.5.2 Use case model

- Actors and use cases

3.5.3 Object model

- Data dictionary
- Class diagrams (classes, associations, attributes and operations)

3.5.4 Dynamic model

- State diagrams for classes with significant dynamic behavior
- Sequence diagrams for collaborating objects (protocol)

3.5.5 User Interface

- Navigational Paths, Screen mockups

Requirements Analysis Questions

1. What are the transformations? **Functional Modeling**

   Create *scenarios and use case diagrams*

   - Talk to client, observe, get historical records

2. What is the structure of the system? **Object Modeling**

   Create *class diagrams*

   - Identify objects.
   - What are the associations between them?
   - What is their multiplicity?
   - What are the attributes of the objects?
   - What operations are defined on the objects?

3. What is its behavior? **Dynamic Modeling**

   Create *sequence diagrams*

   - Identify senders and receivers
   - Show sequence of events exchanged between objects.
   - Identify event dependencies and event concurrency.

   Create *state diagrams*

   - Only for the dynamically interesting objects.

Let's Do Analysis: A Toy Example

- Analyze the problem statement
  - Identify functional requirements
  - Identify nonfunctional requirements
  - Identify constraints (pseudo requirements)
- Build the functional model:
  - Develop use cases to illustrate functional requirements
- Build the dynamic model:
  - Develop sequence diagrams to illustrate the interaction between objects
  - Develop state diagrams for objects with interesting behavior
- Build the object model:
  - Develop class diagrams for the structure of the system

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: Use Cases

- Use case 1: System Initialization
  - Entry condition: Power is off, car is not moving
  - Flow of events:
    1. 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:
    1. Driver turns power off, car stops and headlight goes out.
    2. Driver turns power on, headlight shines and car does not move.
    3. 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:
    1. 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:
    1. Driver turns power off, car stops, headlight goes out.
    2. Power is turned on, headlight shines and car does not move.
    3. Power is turned off, headlight goes out.
  - Exit condition: Car does not move, headlight is out

Use Cases Continued

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

Use Case Pruning

- Do we need use case 5?
- Let us compare use case 1 and use case 5:

<u>Use case 1: System Initialization</u>

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

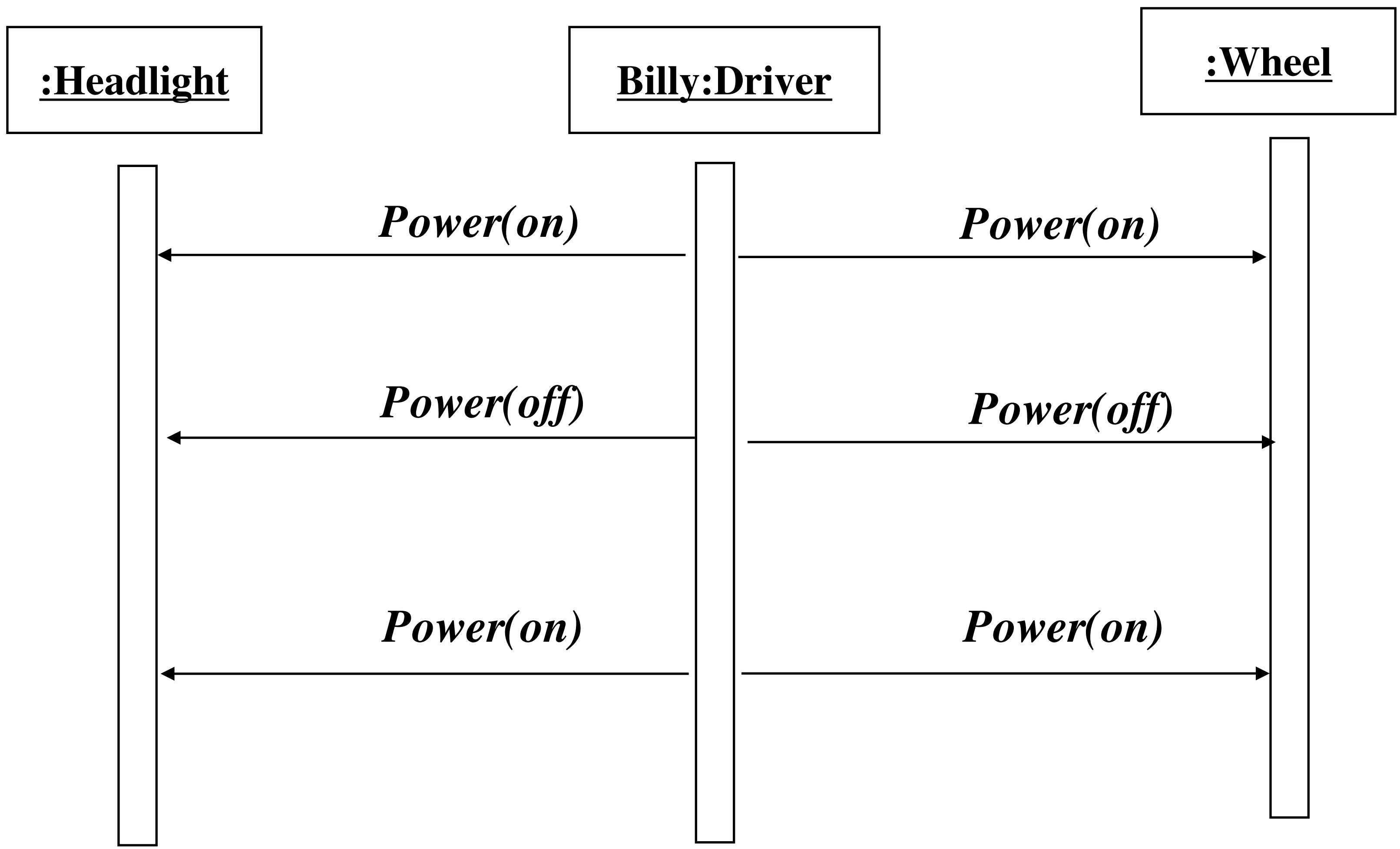
<u>Use case 5: Move car forward</u>

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

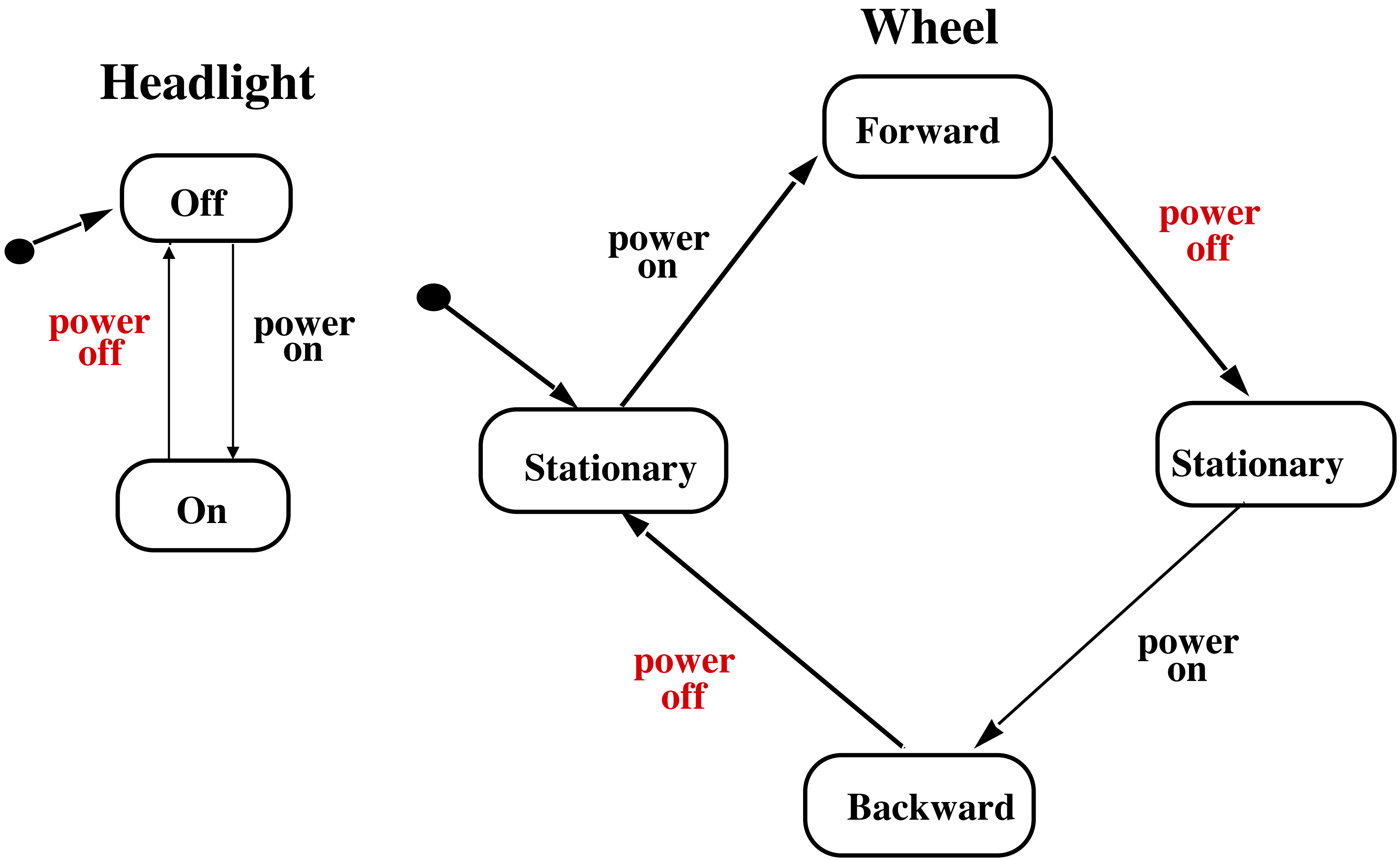
Dynamic Modeling: Create the Sequence Diagram

- 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
  - . . .

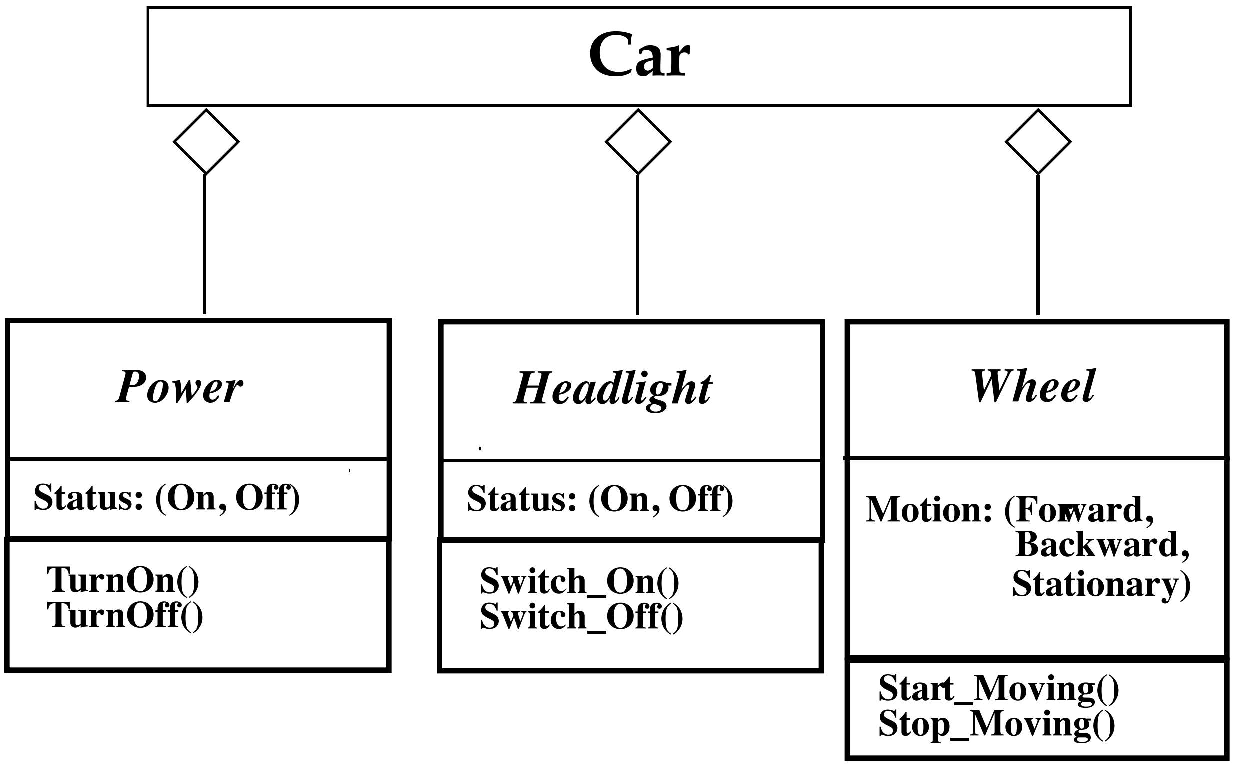
Sequence Diagram for Drive Car Scenario



Toy Car: Dynamic Model



Toy Car: Object Model



Modeling Concurrency of Events

Two types of concurrency:

1. System concurrency
   - The overall system is modeled as the aggregation of state diagrams
   - Each state diagram is executing concurrently with the others.
2. Concurrency within an object
   - An object can issue concurrent events
   - Two problems:
     - Show how control is split
     - Show how to synchronize when moving to a state without object concurrency

Example of Concurrency within an Object