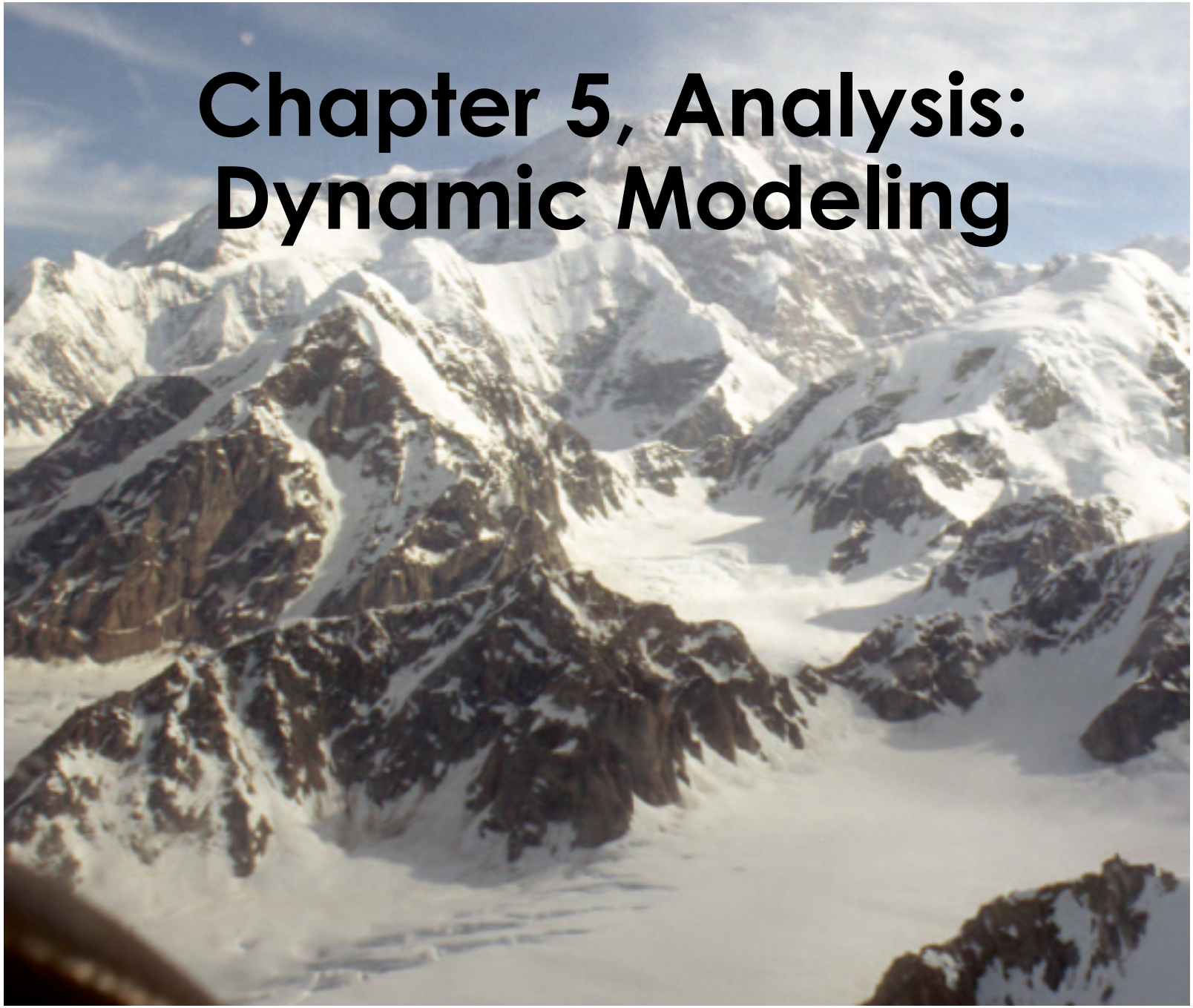
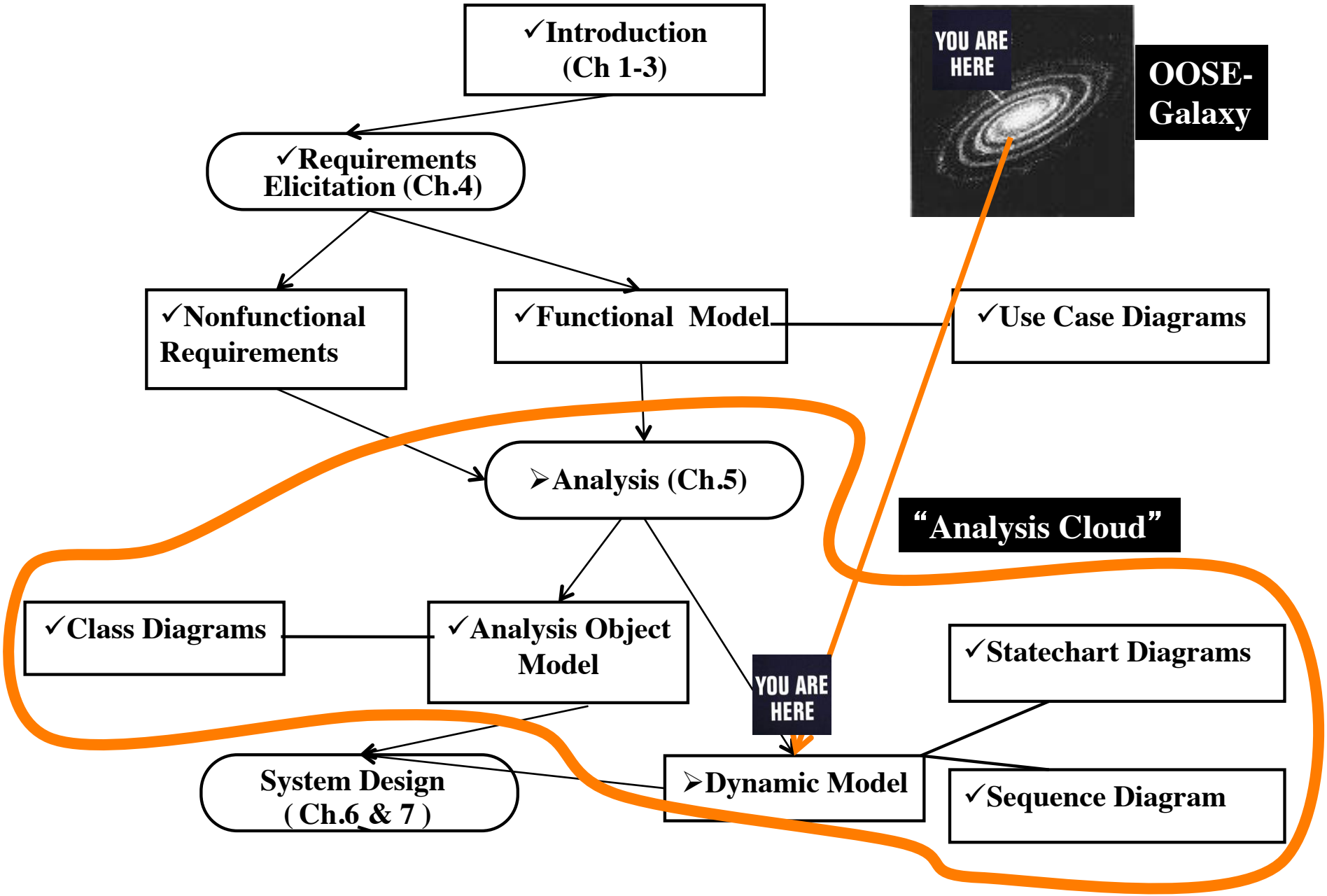
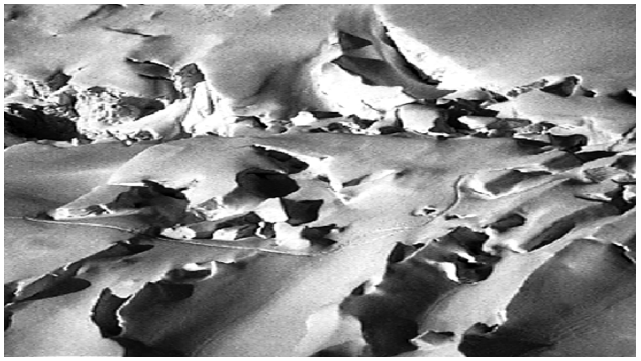
Object-Oriented Software Engineering
Using UML, Patterns, and Java



Chapter 5, Analysis: Dynamic Modeling


✓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

Source Code

Implementation
(Ch. 10)

Testing (Ch. 11)

Deliverable System

Outline of the Lecture

- Dynamic modeling
  - Interaction Diagrams
    - Sequence diagrams
    - Communication diagrams
  - State diagrams
- 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*
  - Interaction diagrams describe the dynamic behavior *between objects.*

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.

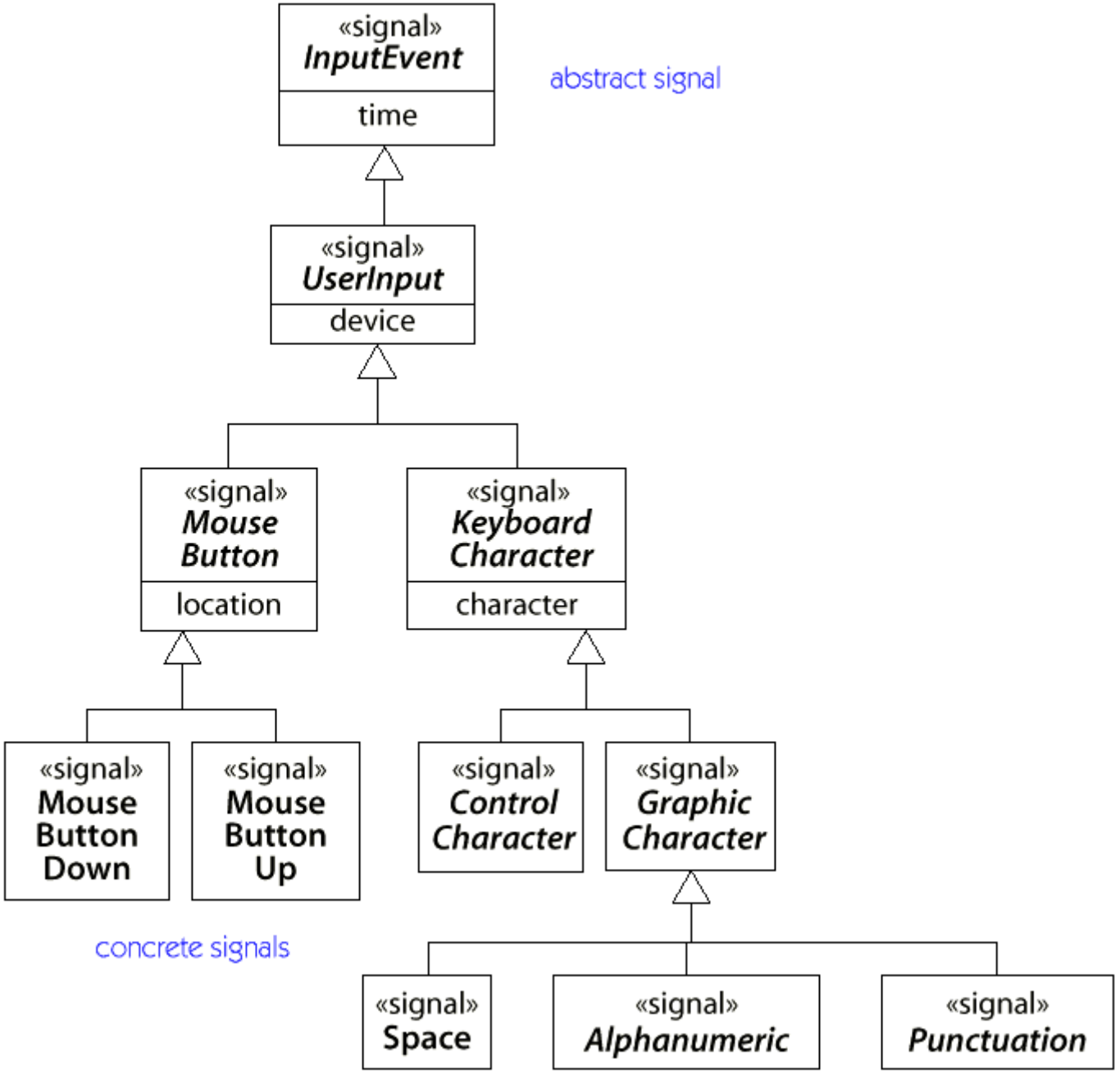
How do we detect Operations?

- Good starting point: Flow of events in a use case description
- We look for objects,
  - who are interacting and extract their “protocol”
  - who have interesting behavior on their own
- 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.

Events hierarchy

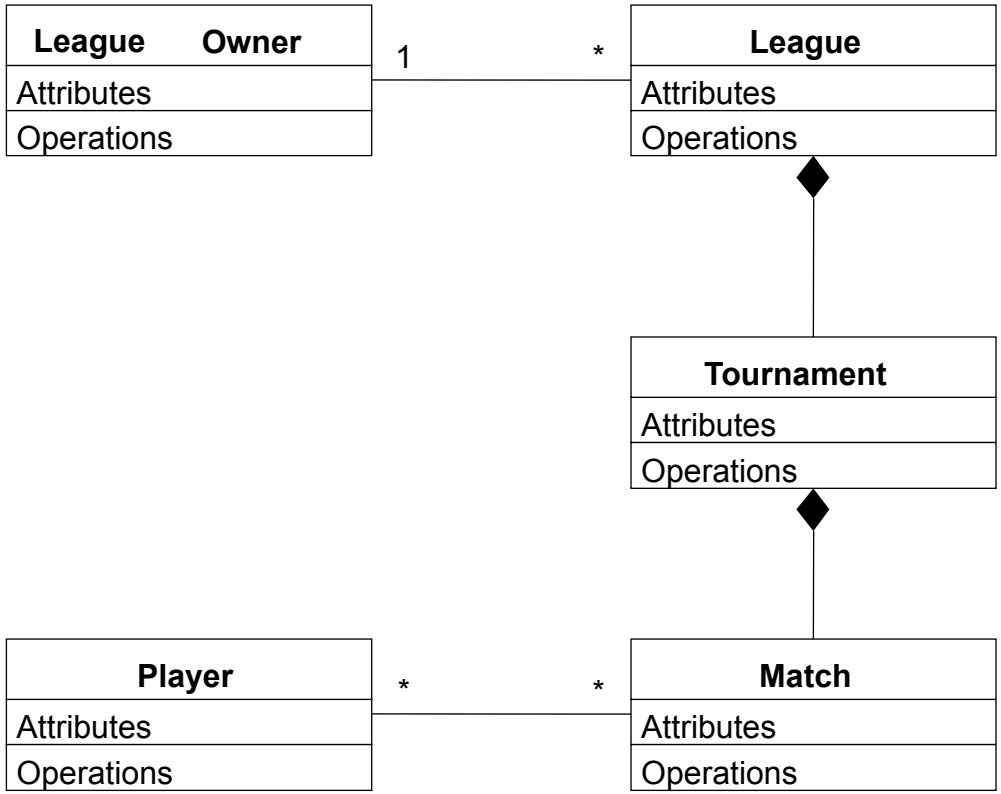


Finding Participating Objects

- 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.

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

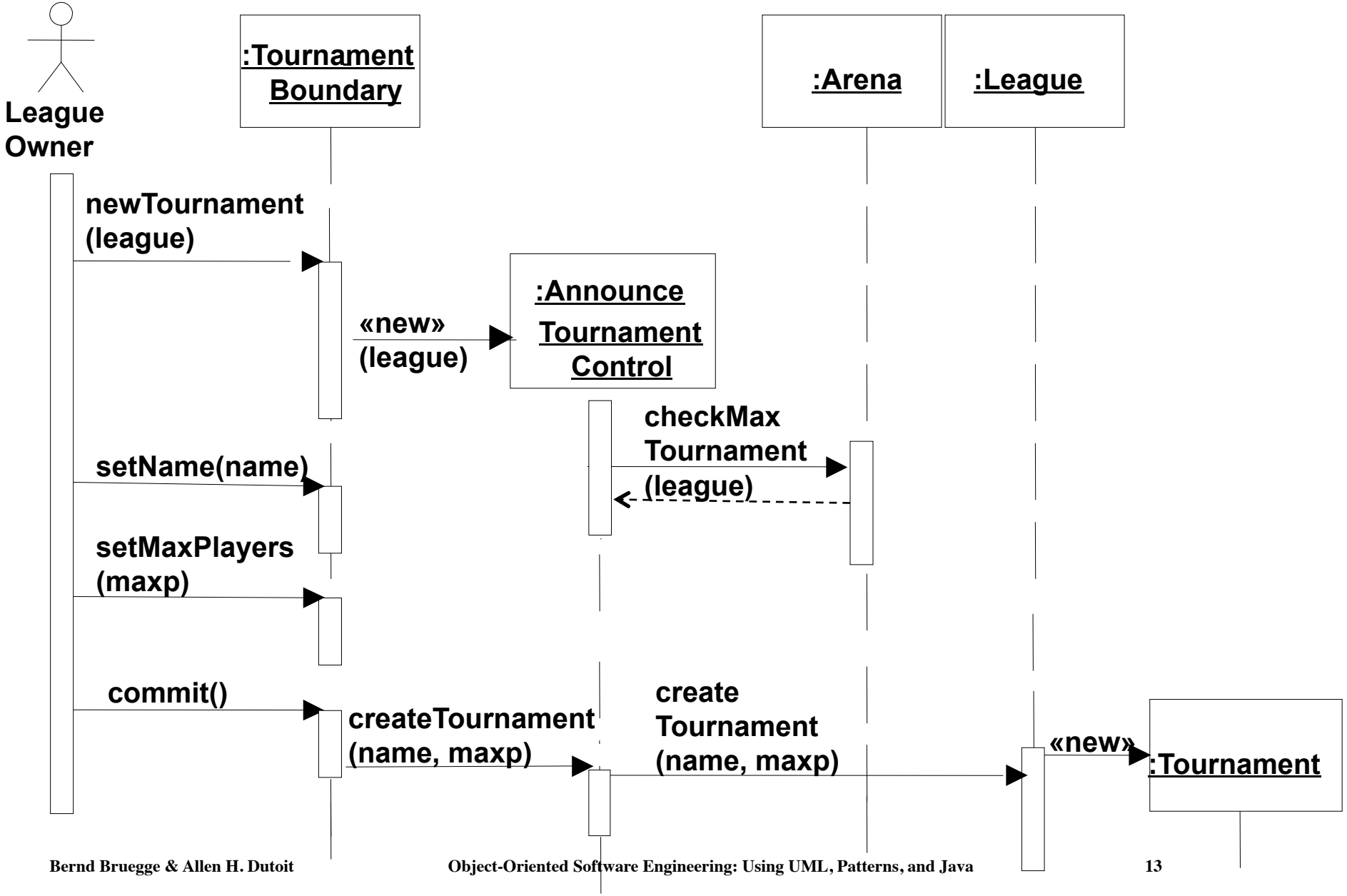


Example: Finding Objects from a Sequence Diagram

- 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



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

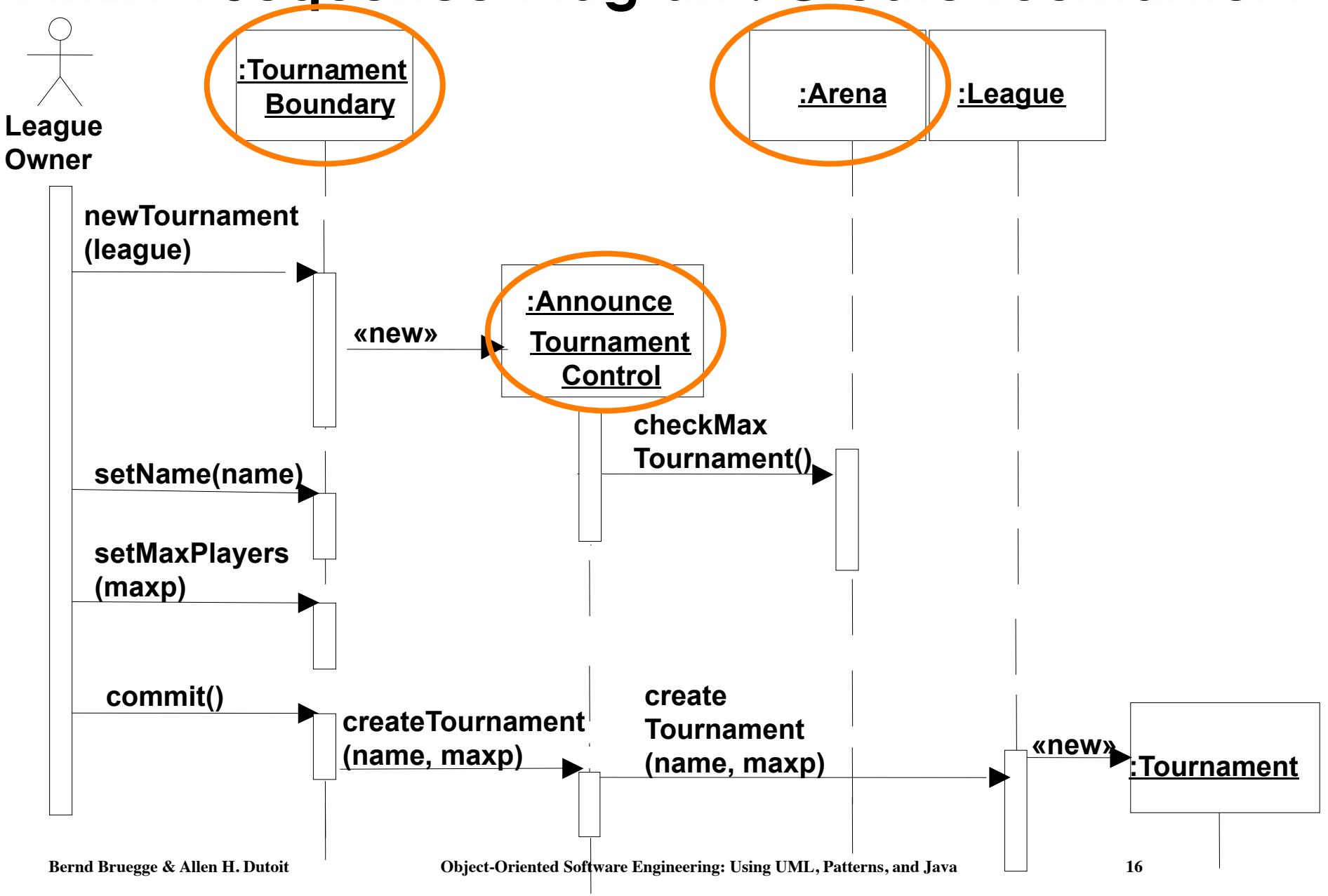
- Creation of objects:
  - Create control objects at beginning of event flow
  - The control objects create the boundary objects
- Access of objects:
  - Entity objects can be accessed by control and boundary objects
  - Entity objects should not access boundary or control objects.

Another Example: Finding Objects from a Sequence Diagram

The Sequence Diagram identified 3 new Classes

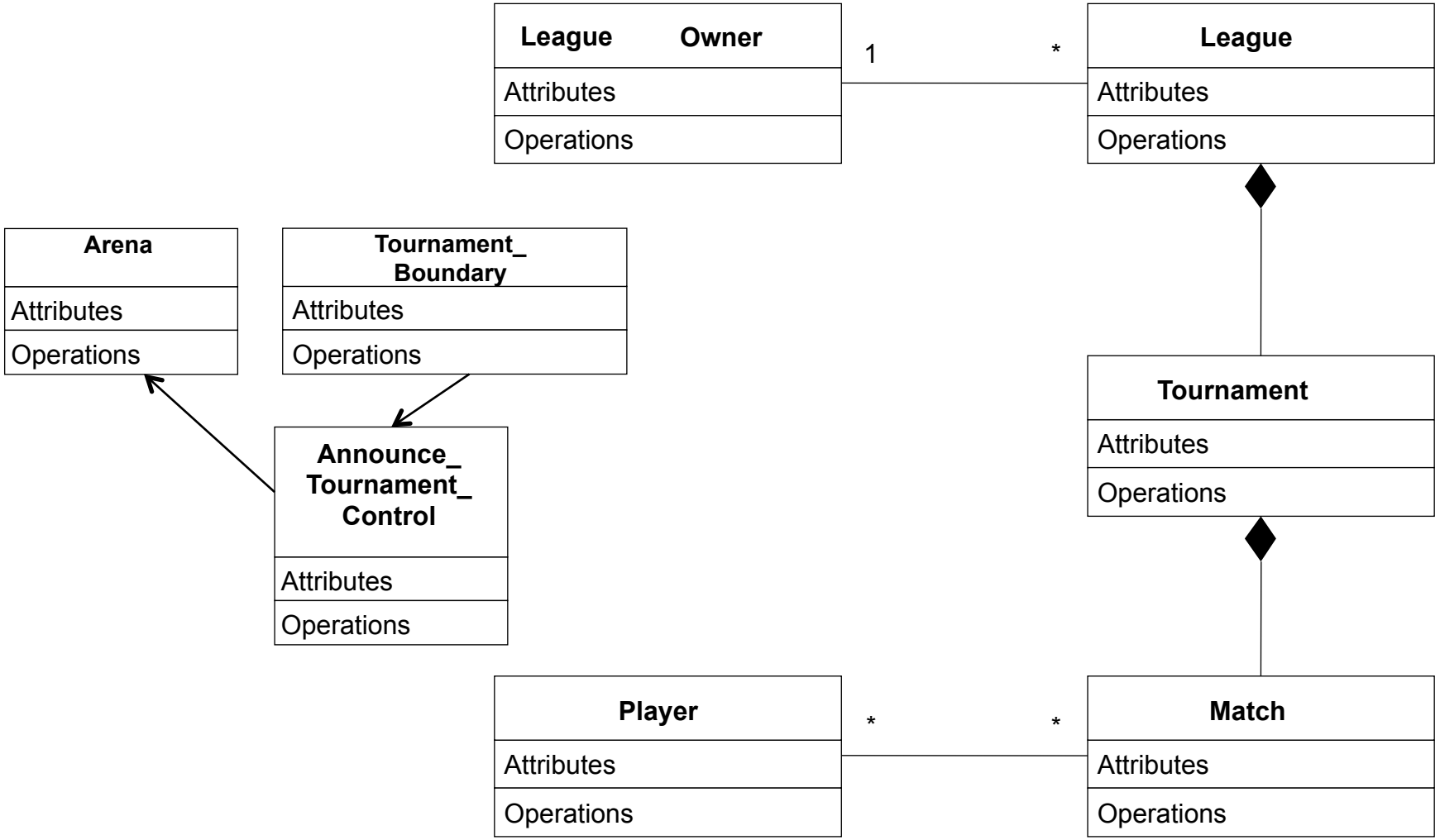
- Tournament Boundary, Announce_Tournament_Control and Arena

ARENA Sequence Diagram: Create Tournament



Impact on Arena's Object Model

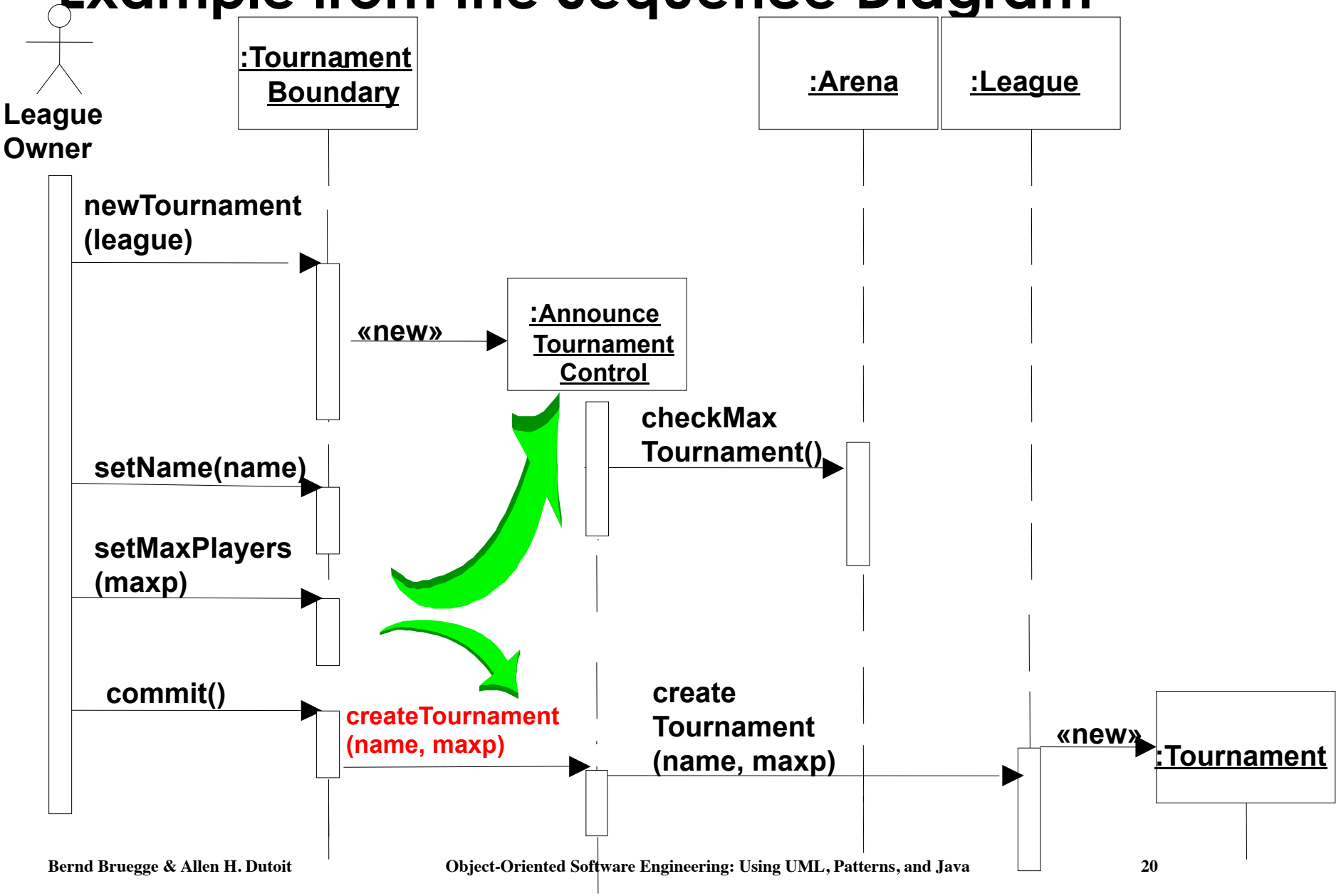
Impact on Arena's Object Model

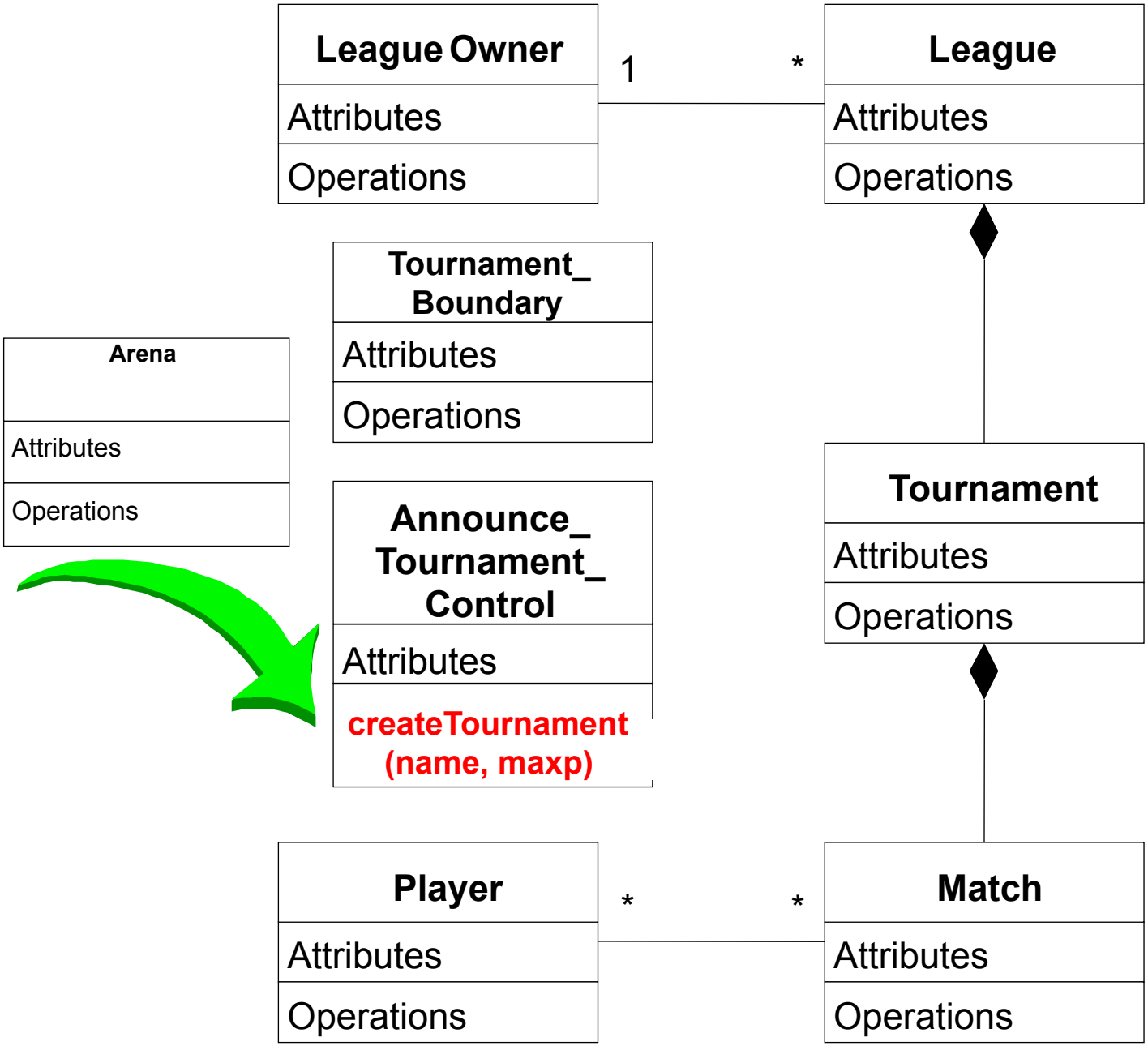


Impact on ARENA's Object Model (2)

- The sequence diagram also supplies us with many new events *(messages exchanged between objects in the diagram)*
  - 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

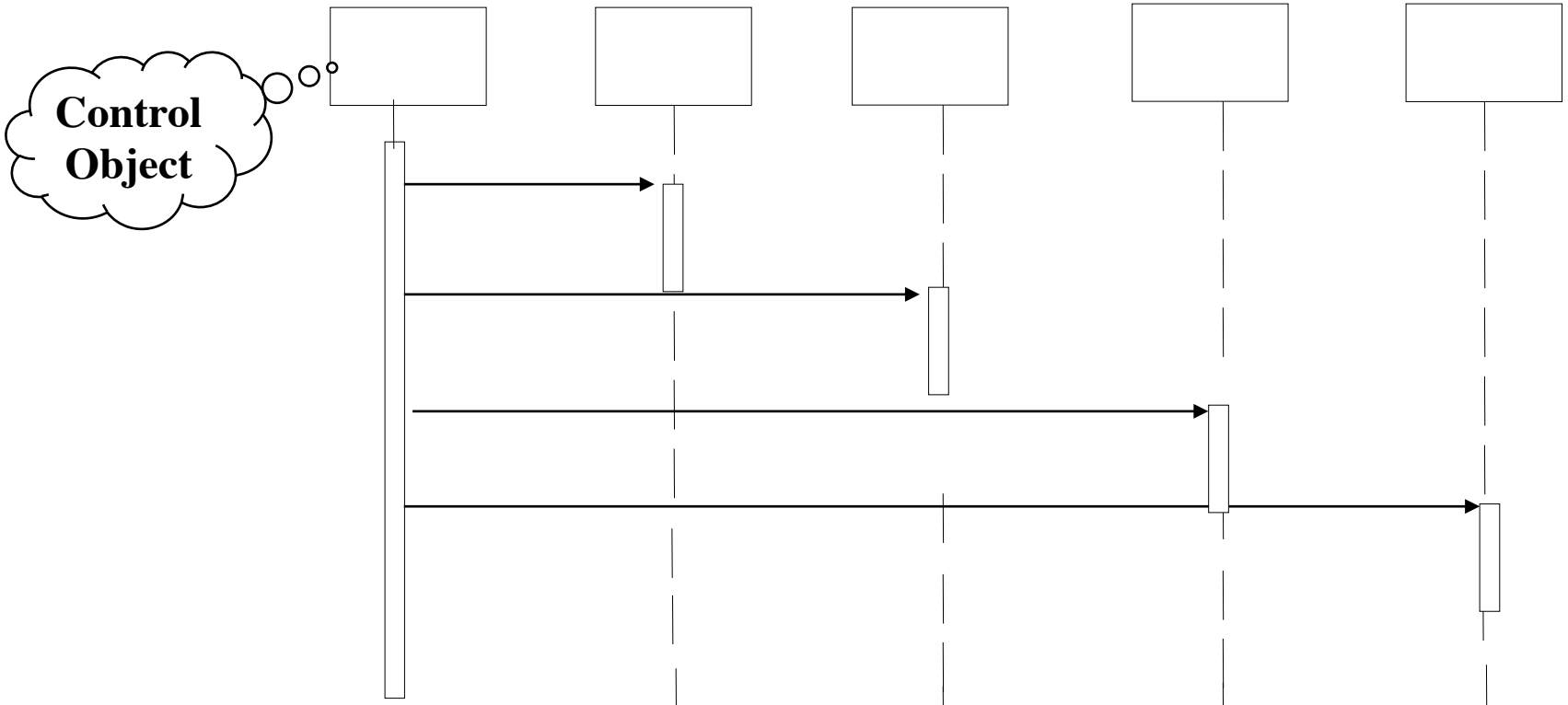
What else can we get out of Sequence Diagrams?

- 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)

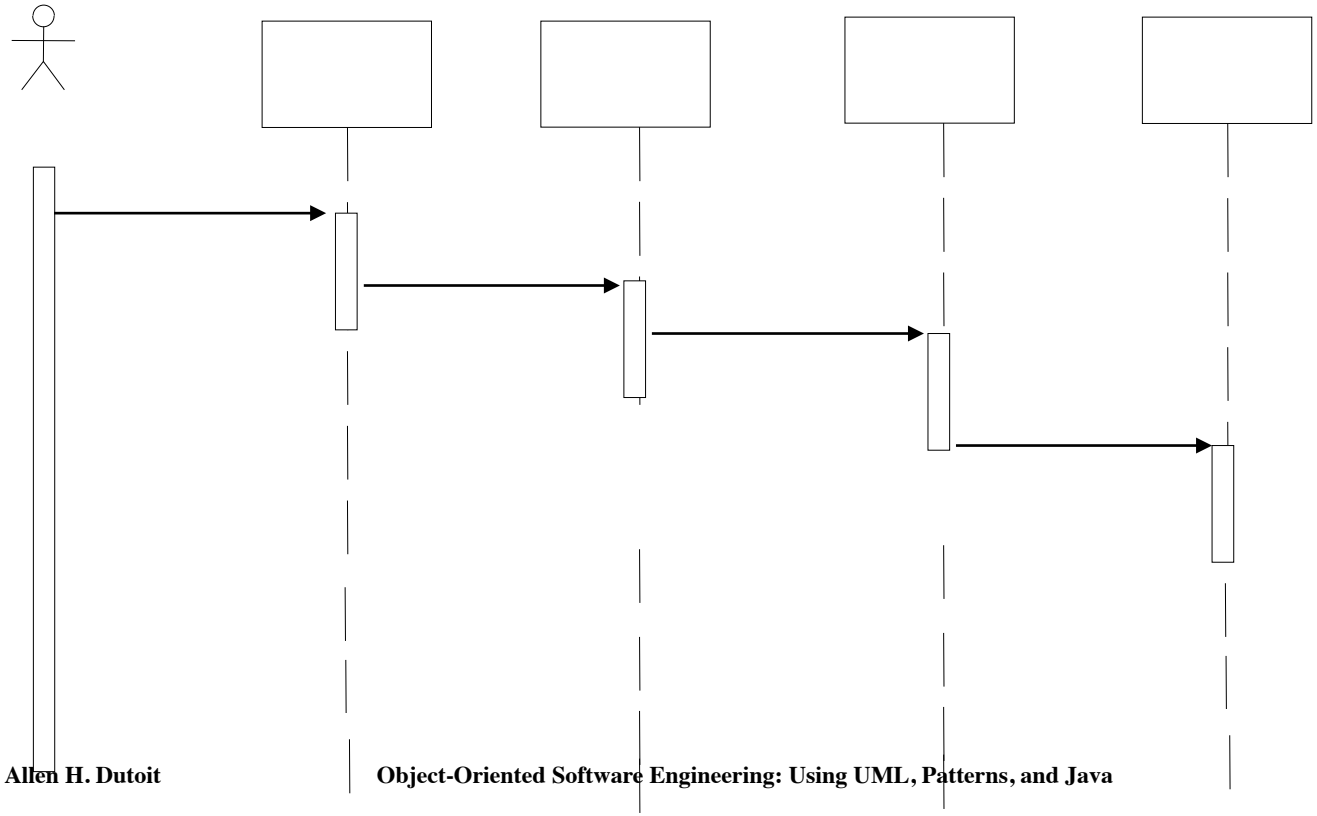
Fork Diagram

- 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.

State

- State: An abstraction of the attributes of a class
  - State is the aggregation of several attributes a class
- State has duration.

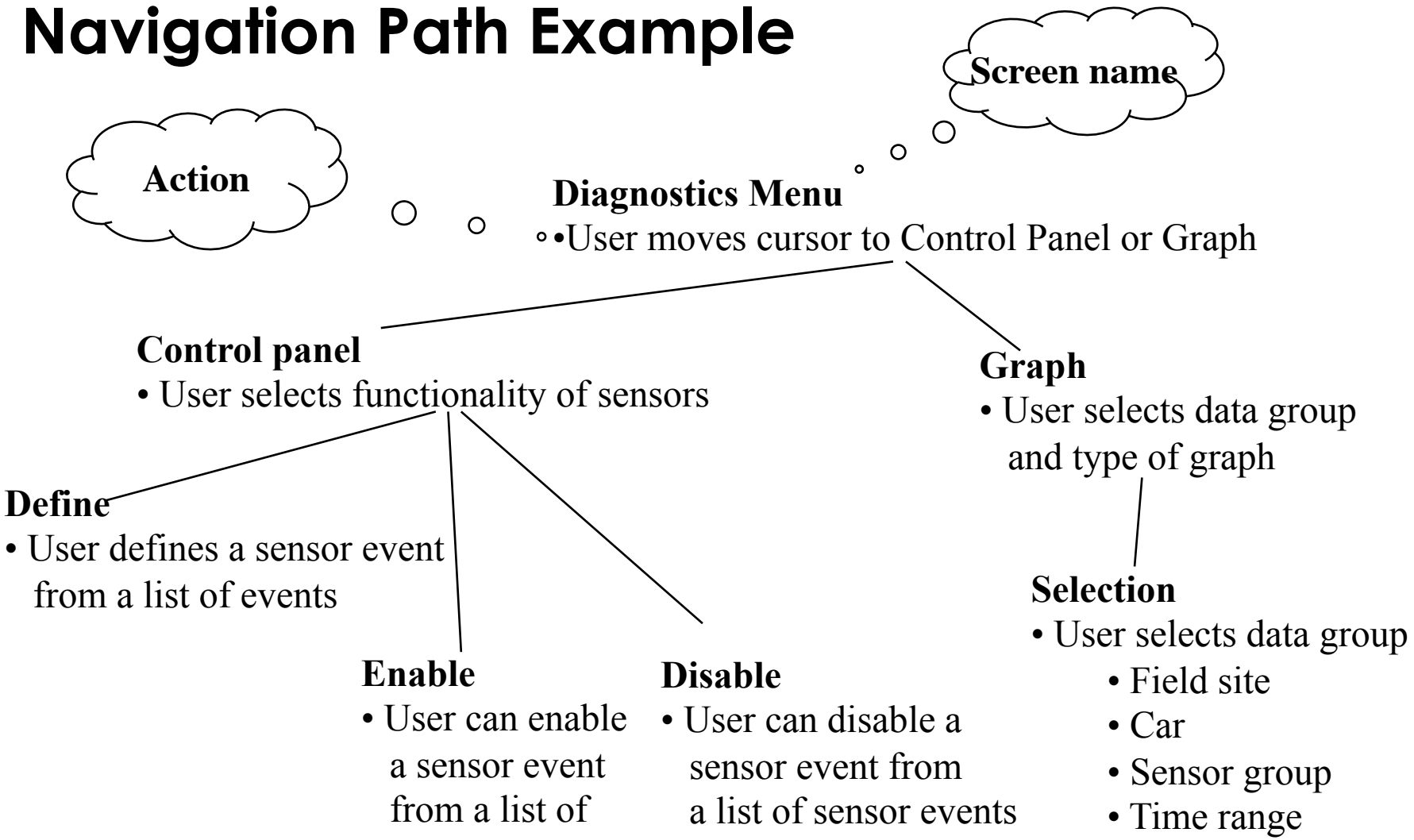
State Chart Diagram vs Sequence Diagram

- State chart diagrams help to identify:
  - Changes to an individual object over time
- Sequence diagrams help to identify:
  - The temporal relationship of communications 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



NOT a good UML diagram! Syntax not respected

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 (e.g. use cases) 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

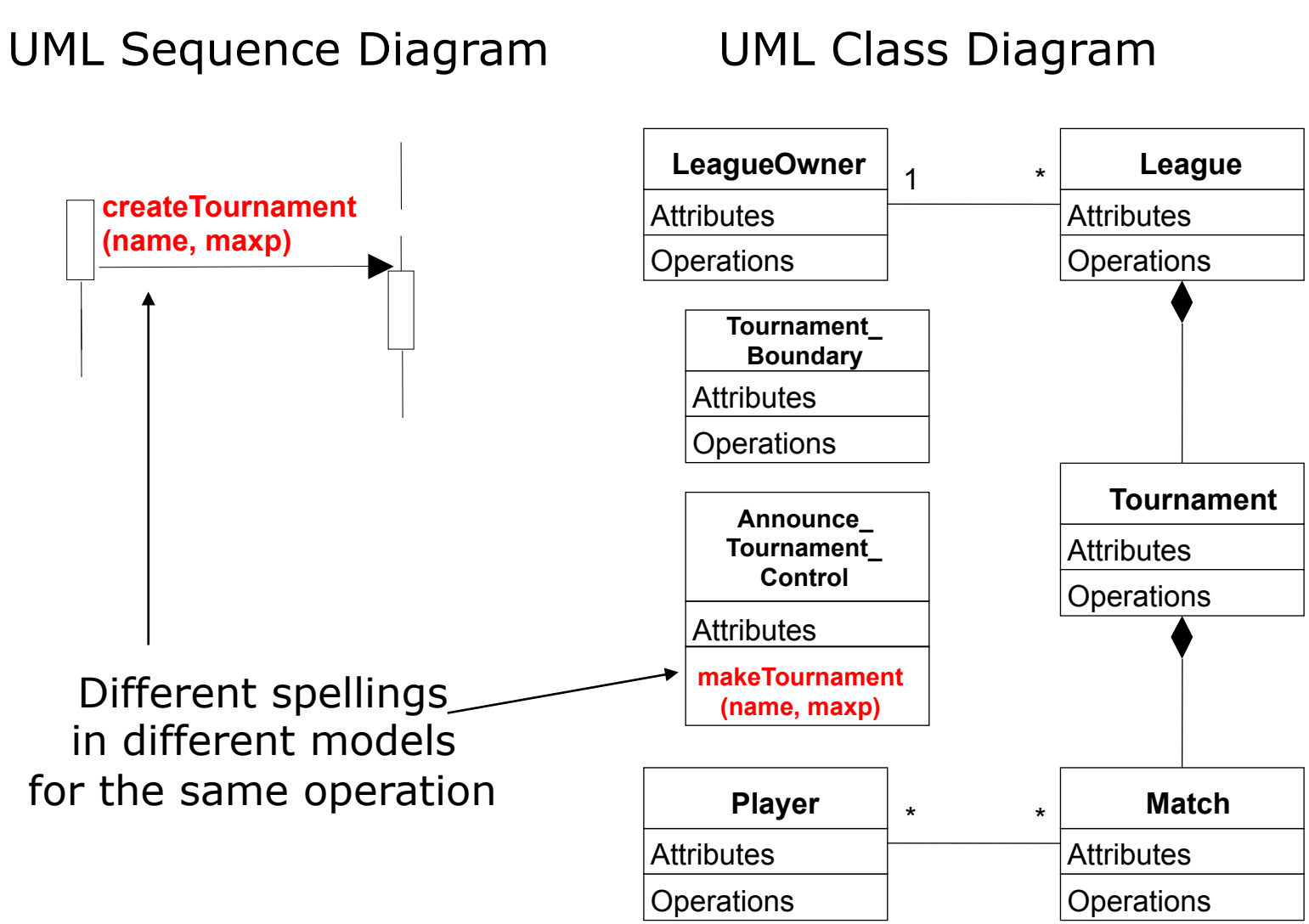
Checklist for a Requirements Review

- 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

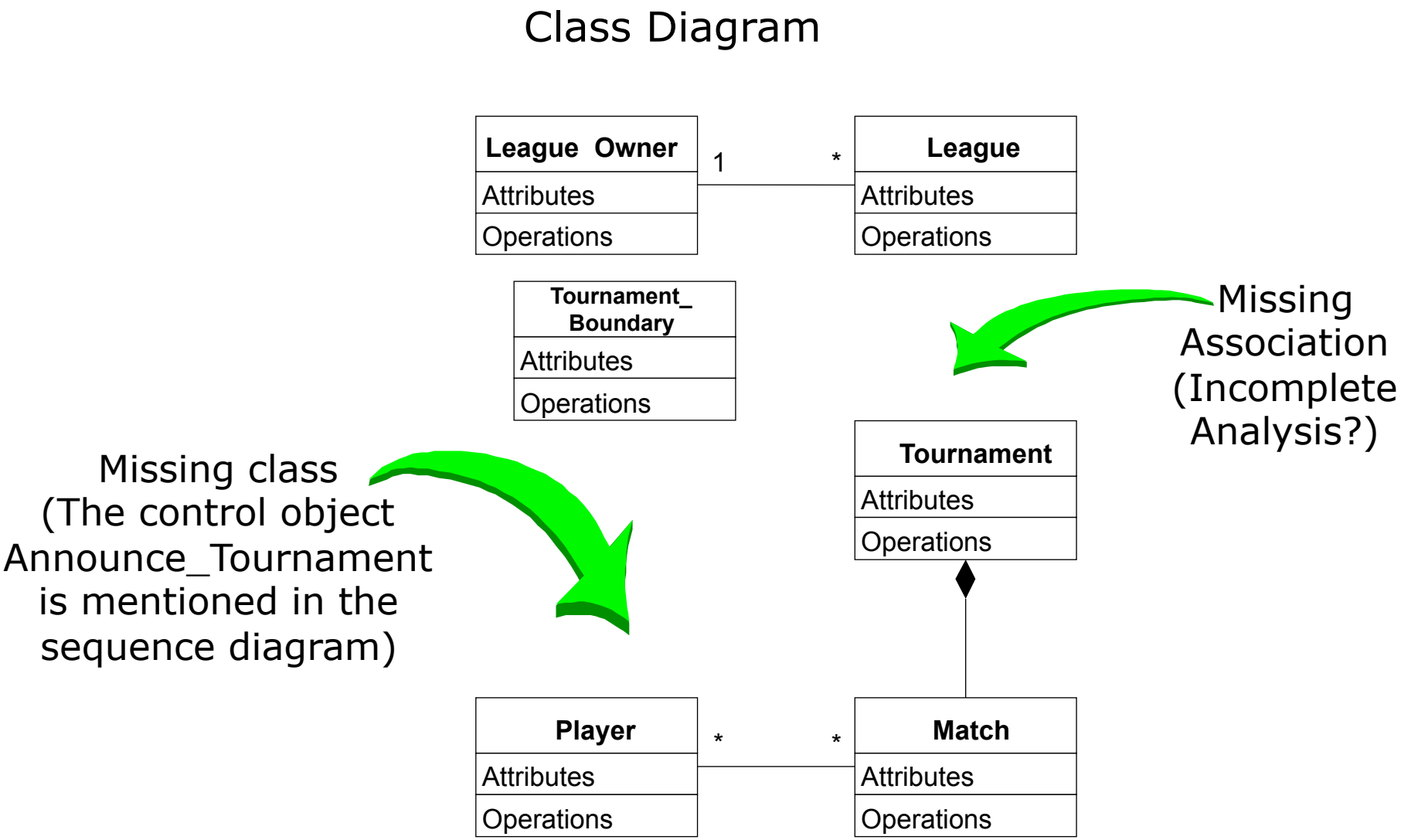
Examples for Inconsistency and Completeness Problems

- Different spellings in different UML diagrams
  - The correct use of a good UML modeling tool may help a lot on this
- Omissions in diagrams

Different spellings in different UML diagrams



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 double- defined classes
- Identify missing classes (mentioned in one model but not defined anywhere)
  - Avoid this problem by using this guideline:
    1. Define new classes/methods in class diagrams
    2. In sequence diagrams: introduce objects/ messages by selecting their names from lists proposed by the case tool
- Check for classes with the same name but different meanings

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 interface
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
  - Description of data (entity classes) and their meaning in the system
- 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/interactions?  **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 object model:
  - Develop class diagrams for the structure of the system
- Build the dynamic model:
  - Develop sequence diagrams to illustrate the interaction between objects
  - Develop state diagrams for objects with interesting behavior

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:

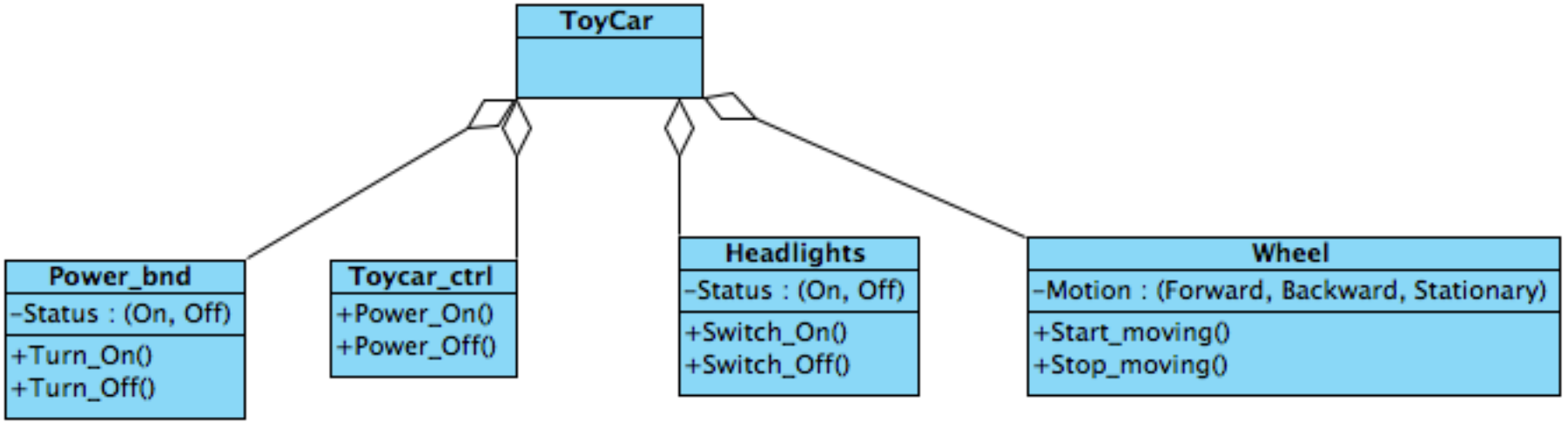
Use case 1: System Initialization

- 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 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

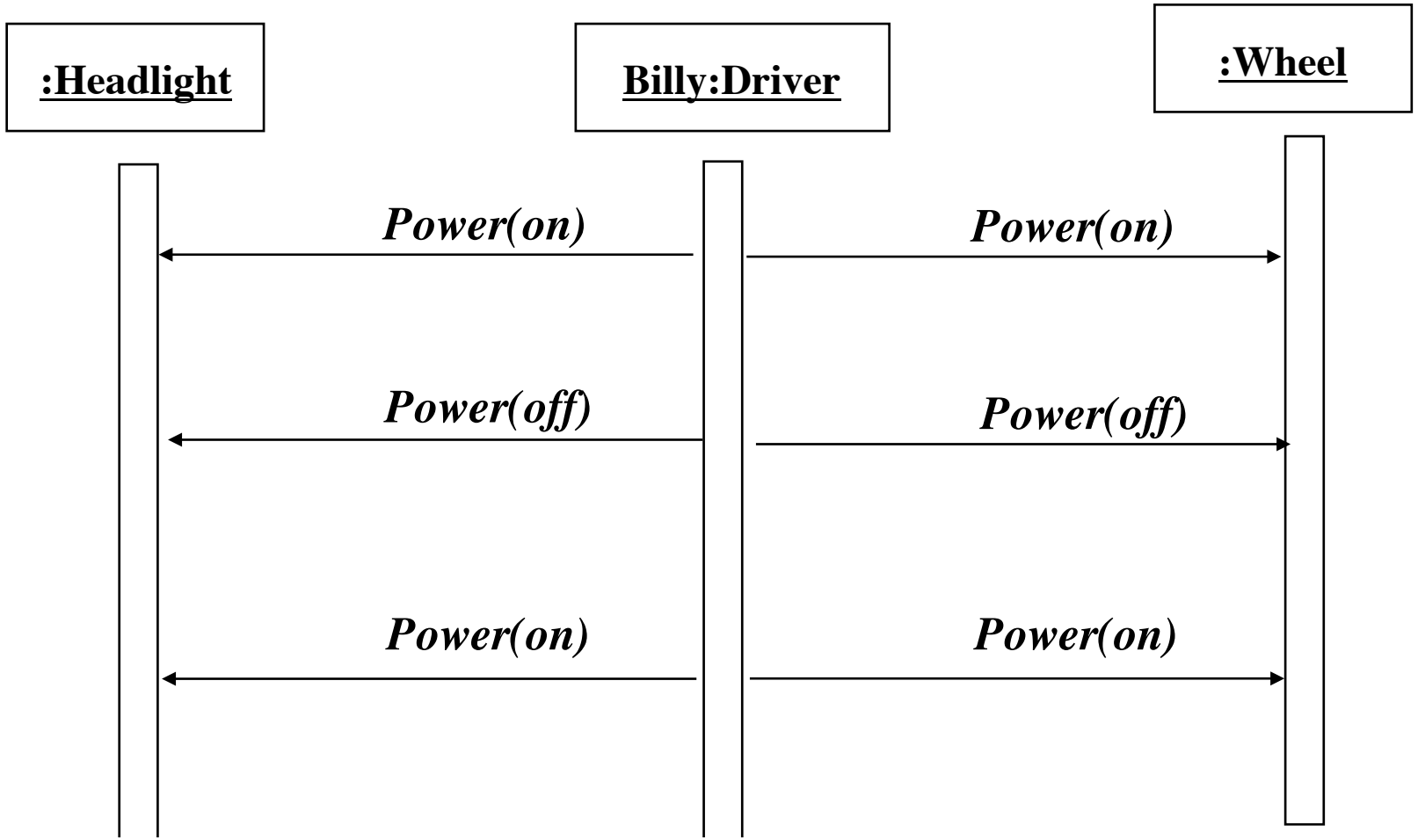
Toy Car: Object Model



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
  - . . .

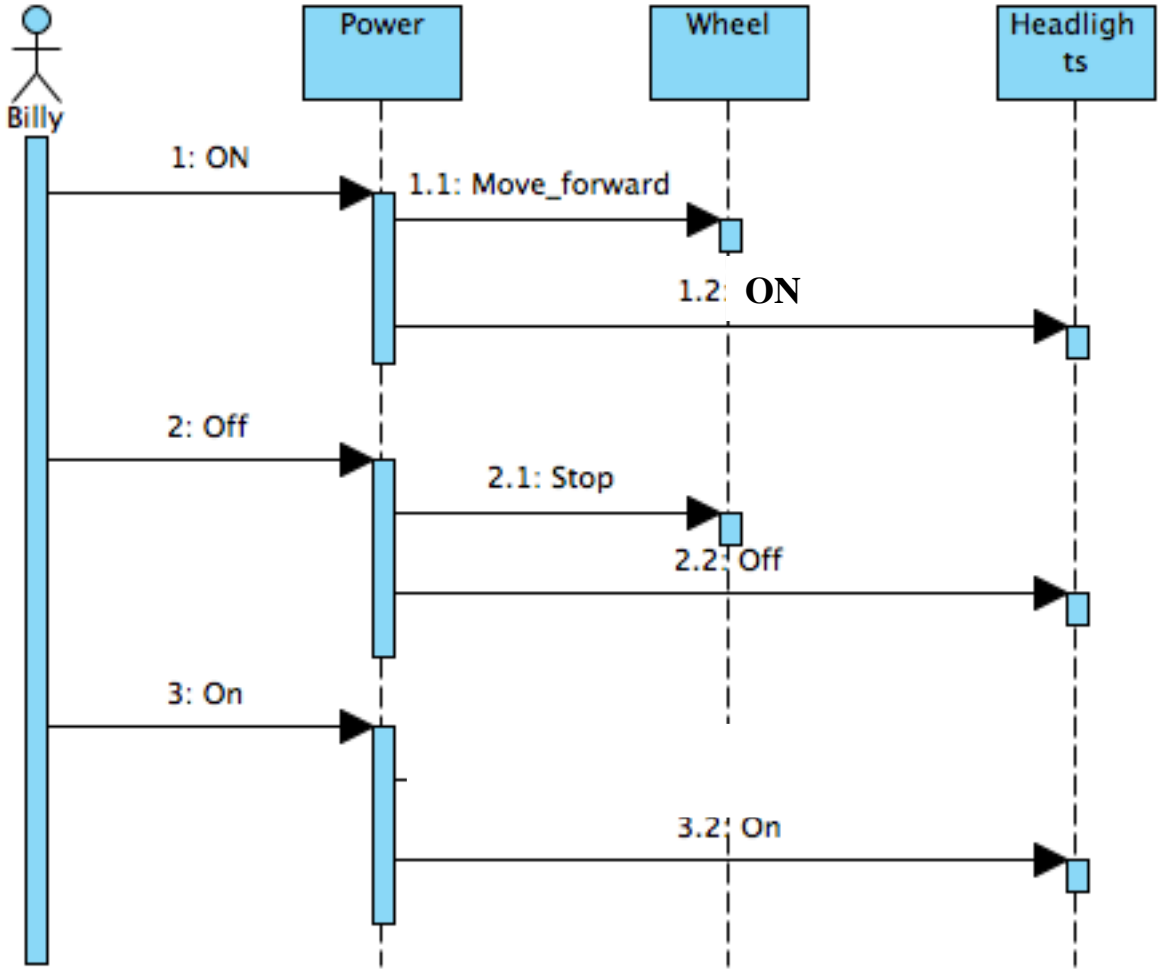
Sequence Diagram for Drive Car Scenario



WRONG ORDER OF OBJECTS!!!

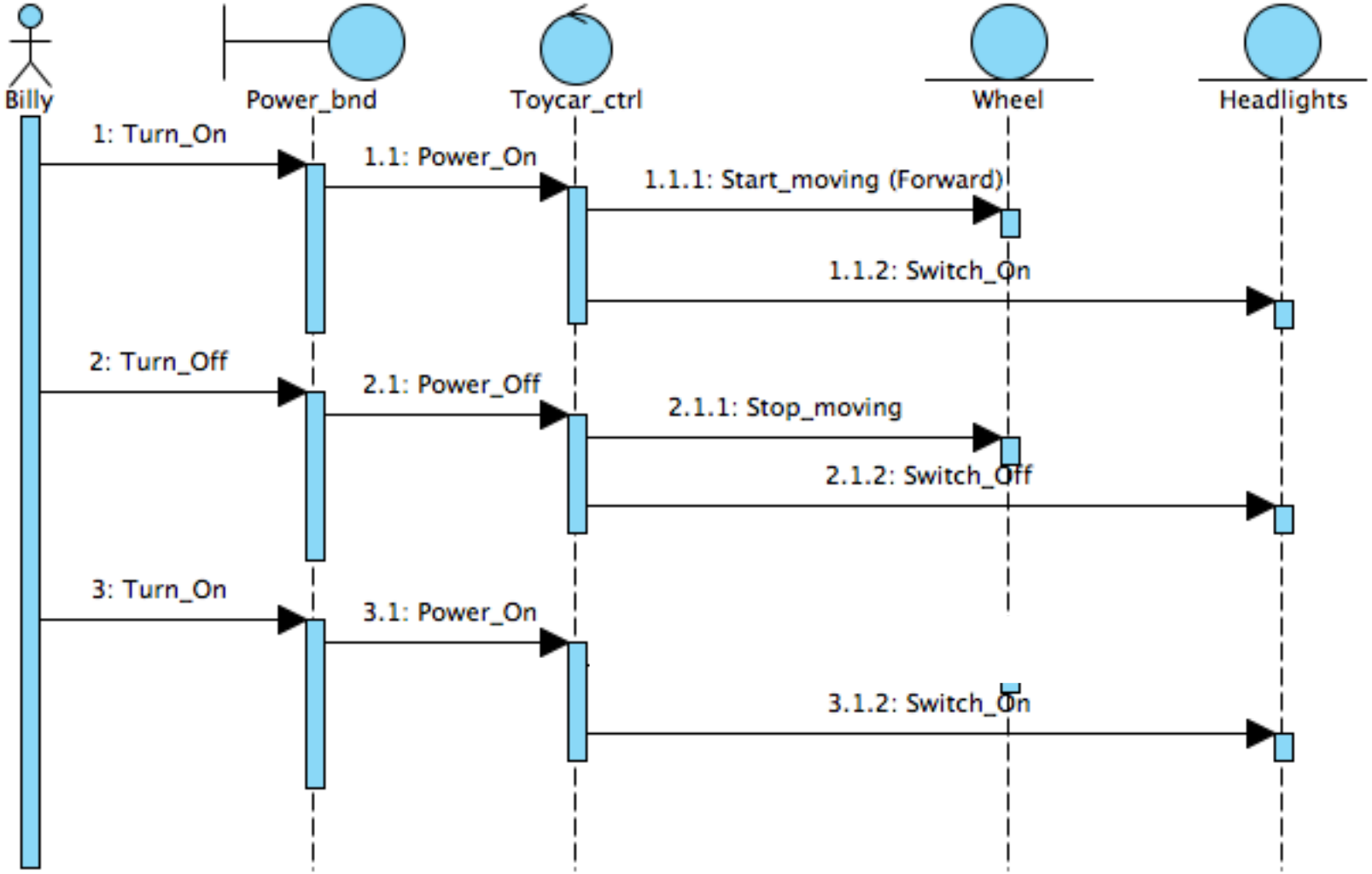
Wrong sequence of events (two messages starting at the same time)

Sequence Diagram for Drive Car Scenario



Not good yet: no boundaries, who is controlling the flow? What about data objects?

Sequence Diagram for Drive Car Scenario



Good!

Toy Car: Dynamic Model