Dynamic Modeling

Software Engineering 1
Lecture 10

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

Outline of the Lecture

- Dynamic modeling
  - Sequence diagrams
  - State diagrams
- Using dynamic modeling for the design of user interfaces
- Analysis example
- Requirements analysis model validation

How do you find classes?

- We have established sources for classes:
  - Application domain analysis: We find classes by talking to the client and identify abstractions by observing the end user
  - General world knowledge and intuition
  - Scenarios: Natural language formulation of a concrete usage of the system
  - Use Cases: Natural language formulation of the system functions
  - Textual analysis of problem statement (Abbot)
- Today we identify classes from dynamic models
  - Actions and activities in state chart diagrams are candidates for public operations in classes
  - Activity lines in sequence diagrams are candidates for objects

Dynamic Modeling with UML

- Diagrams for dynamic modeling
  - Interaction diagrams describe the dynamic behavior between objects
  - Statechart diagrams describe the dynamic behavior of a single object

Interaction Diagram

- Two types of interaction diagram:
  - Sequence Diagram:
    - Describes the dynamic behavior of several objects over time
    - Good for real-time specifications
  - Collaboration Diagram:
    - Shows the temporal relationship among objects
    - Position of objects is based on the position of the classes in the UML class diagram.
    - Does not show time

State Chart Diagram

- State Chart Diagram:
  - A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).
- Activity Diagram:
  - A special type of statechart diagram, where all states are action states (Moore Automaton)

Dynamic Modeling

- Definition of 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 important dynamic behavior
  - Sequence diagrams: For the interaction between classes
- Purpose:
  - Detect and supply methods for the object model

How do we detect Methods?

- Purpose:
  - Detect and supply methods for the object model
- How do we do this?
  - We look for objects, who are interacting and extract their “protocol”
  - We look for objects, who have interesting behavior on their own
  - ▷ We start with the flow of events in a use case
  - ▷ From the flow of events we proceed to the sequence diagram

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 can 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 10 shown on Tuesday Dec 5 at 10:30”.
  - Event class “Lecture Given”, Subclass “Slide Shown”
- Attribute of an event class
  - Slide Update(5:30 AM, 12/4/2006)
  - Train_Leaves(4:45pm, Manhattan)
  - Mouse button down(button#, tablet-location)

Sequence Diagram

- A sequence diagram is a graphical description of the objects participating in a use case using a DAG notation
- Heuristic for finding participating objects:
  - A event always has a sender and a receiver.
  - Find them for each event => These are the objects participating in the use case
- Relation to object identification:
  - Several objects/classes have already been identified during object modeling
  - New objects are now identified as a result of dynamic modeling

An Example

- Flow of events in “Get SeatPosition” use case :

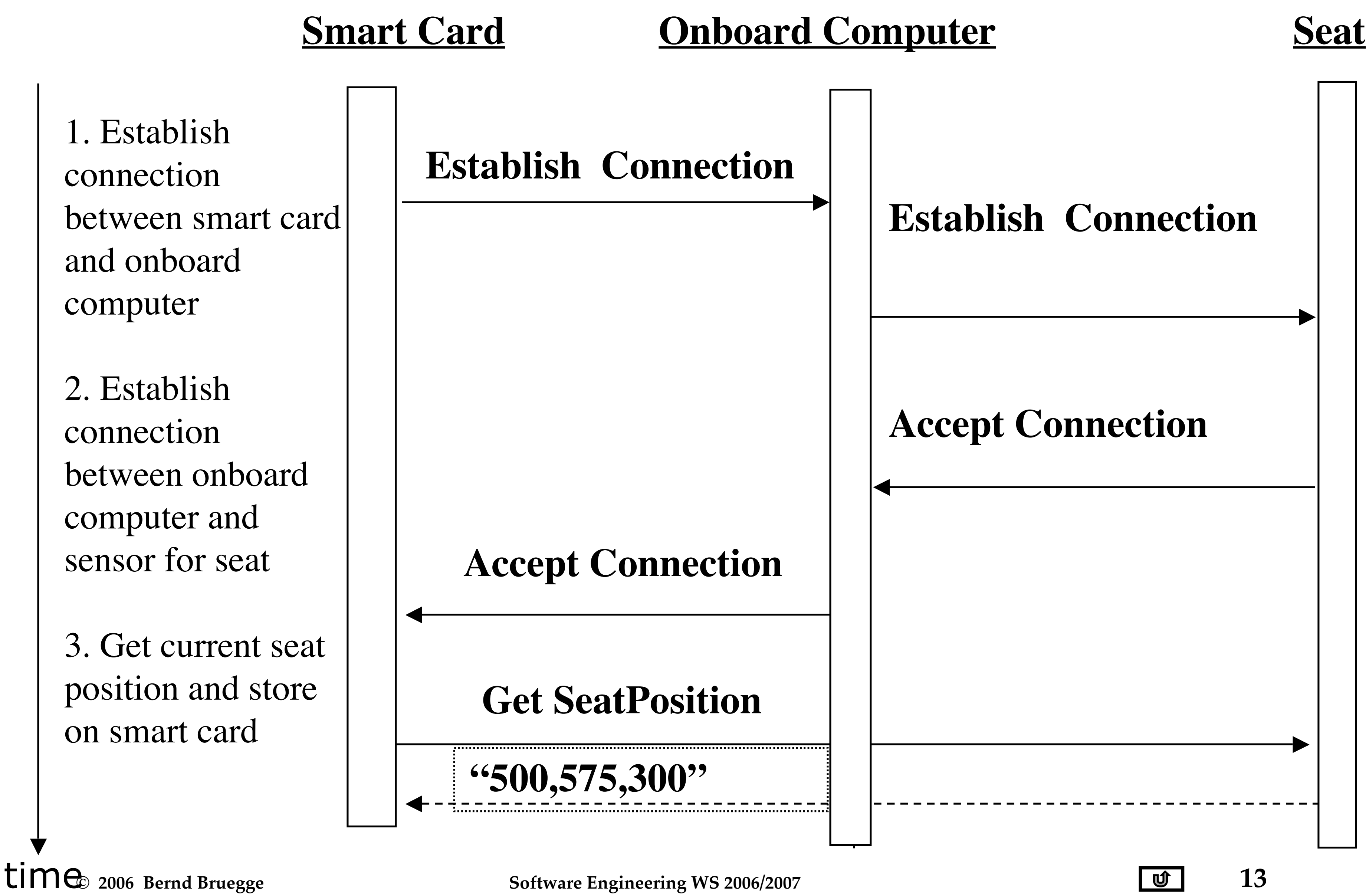
  1. Establish connection between smart card and onboard computer

  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”



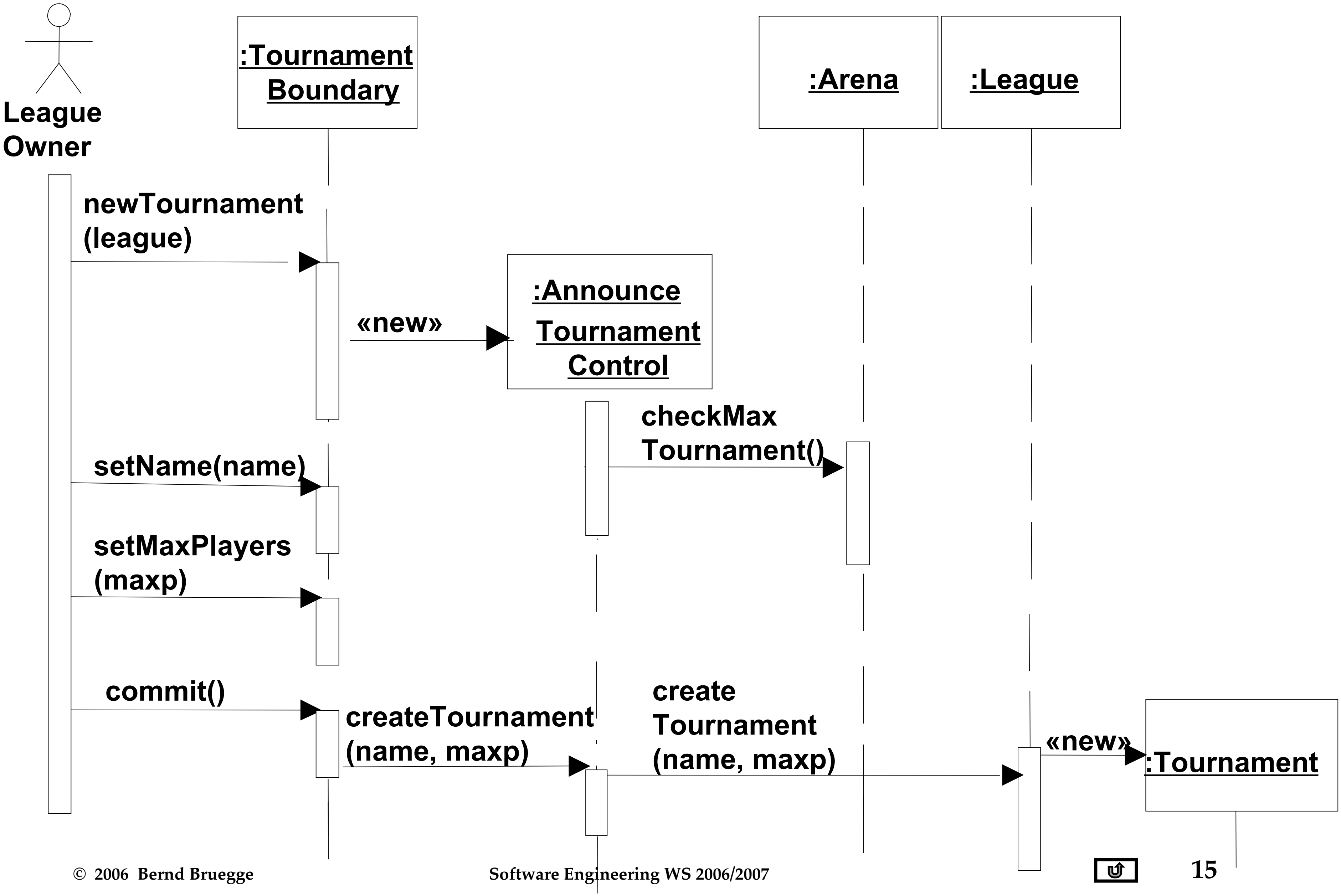
Heuristics for Sequence Diagrams

- Layout:
  1st column: Should correspond to the actor who initiated 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
  - Control objects are created at the initiation of a use case
  - Boundary objects are created by control objects

- Access of objects
  - Entity objects are accessed by control and boundary objects
  - Entity objects should never access boundary or control objects

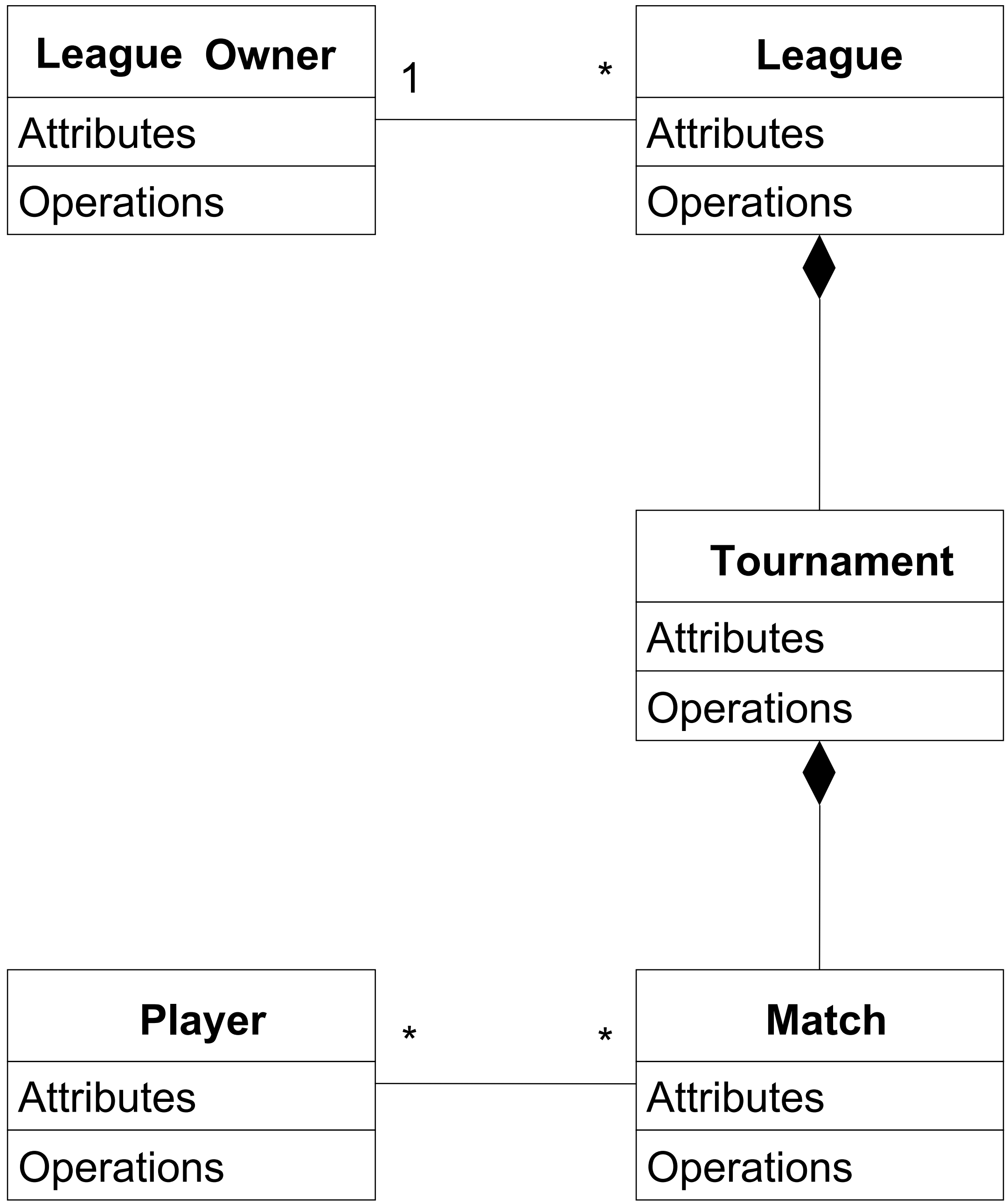
ARENA Sequence Diagram: Create Tournament

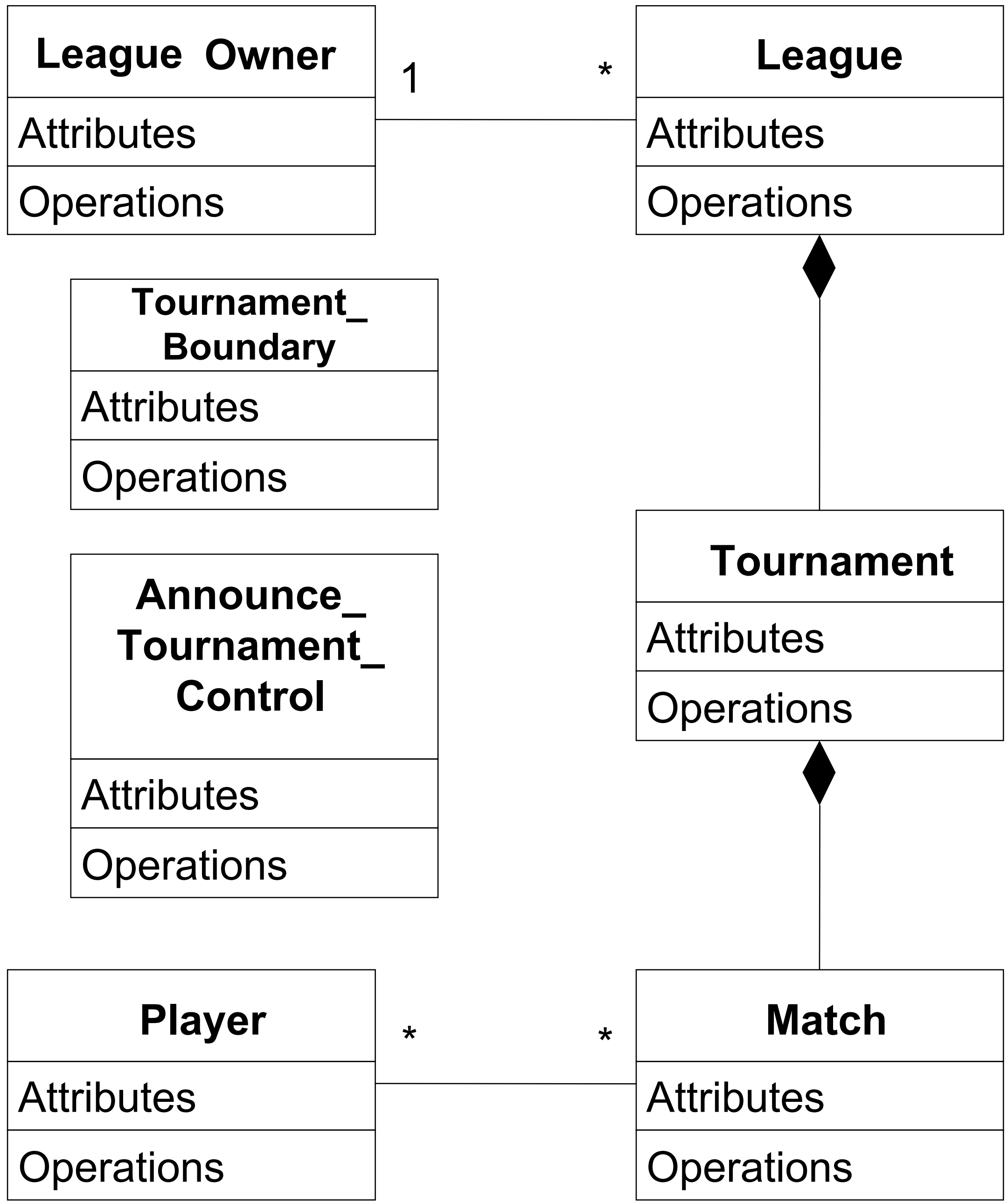


Impact on ARENA's Object Model

- Let's assume ARENA's object model contained the objects
  - League Owner, Arena, League, Tournament, Match and Player

- The Sequence Diagram identified new Classes
  - Tournament Boundary, Announce_Tournament_Control


League Owner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

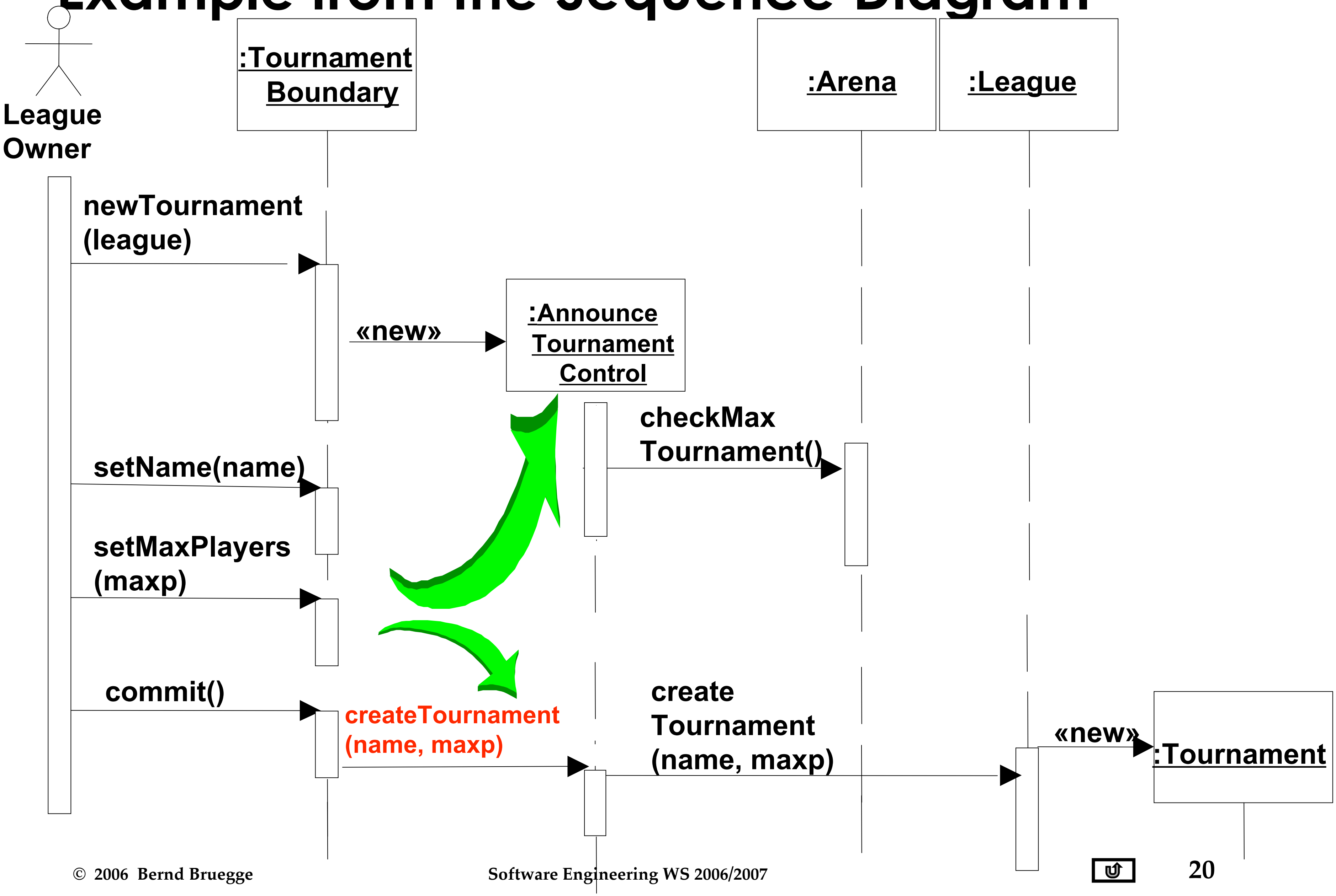

League Owner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament_ Boundary
Attributes
Operations
Announce_ Tournament_ Control
Attributes
Operations
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

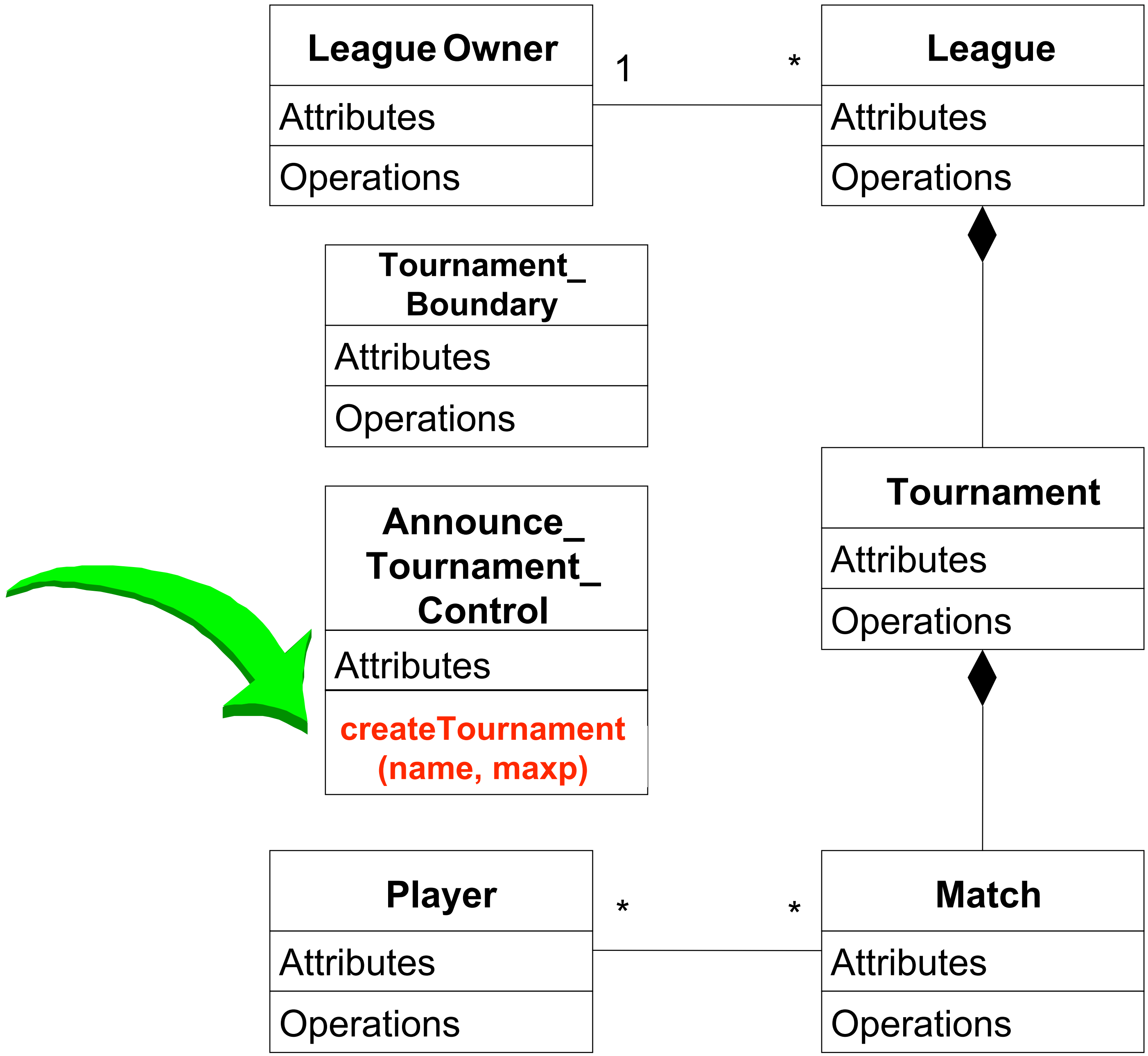
Impact on ARENA's Object Model (2)

- The sequence diagram supplied 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
Announce_
Tournament_
Control
Attributes
createTournament
(name, maxp)
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

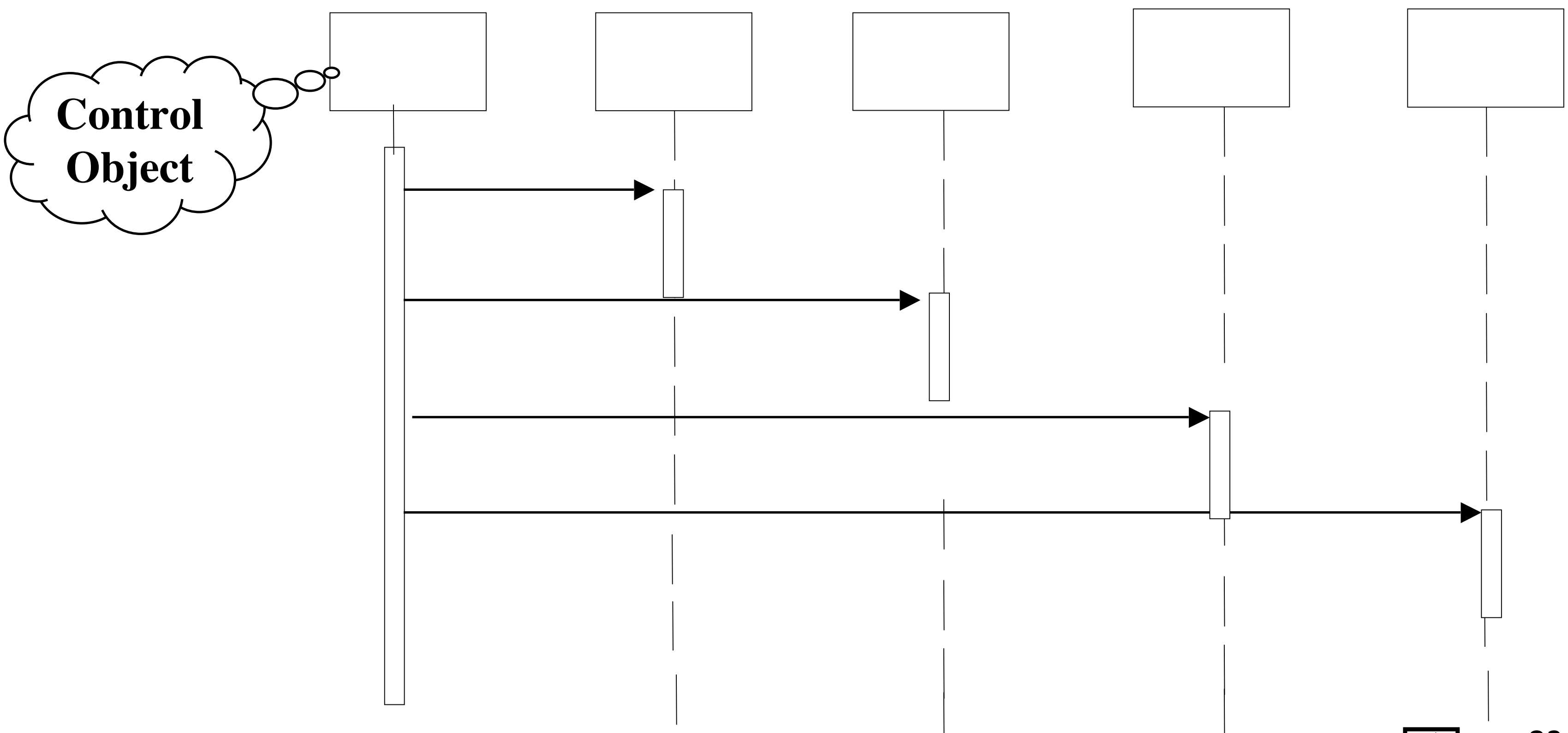
What else can we get out of Sequence Diagrams?

- 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)

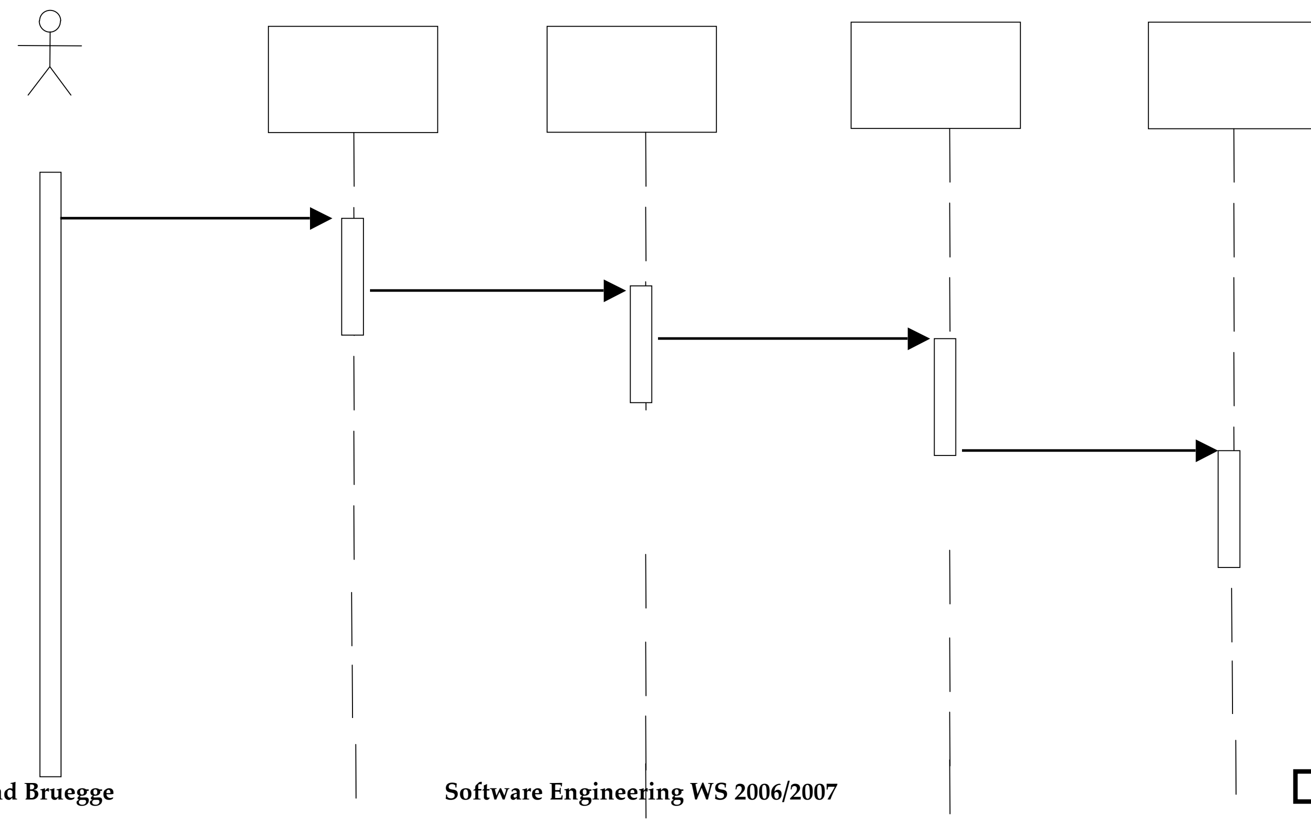
Fork Diagram

- 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.



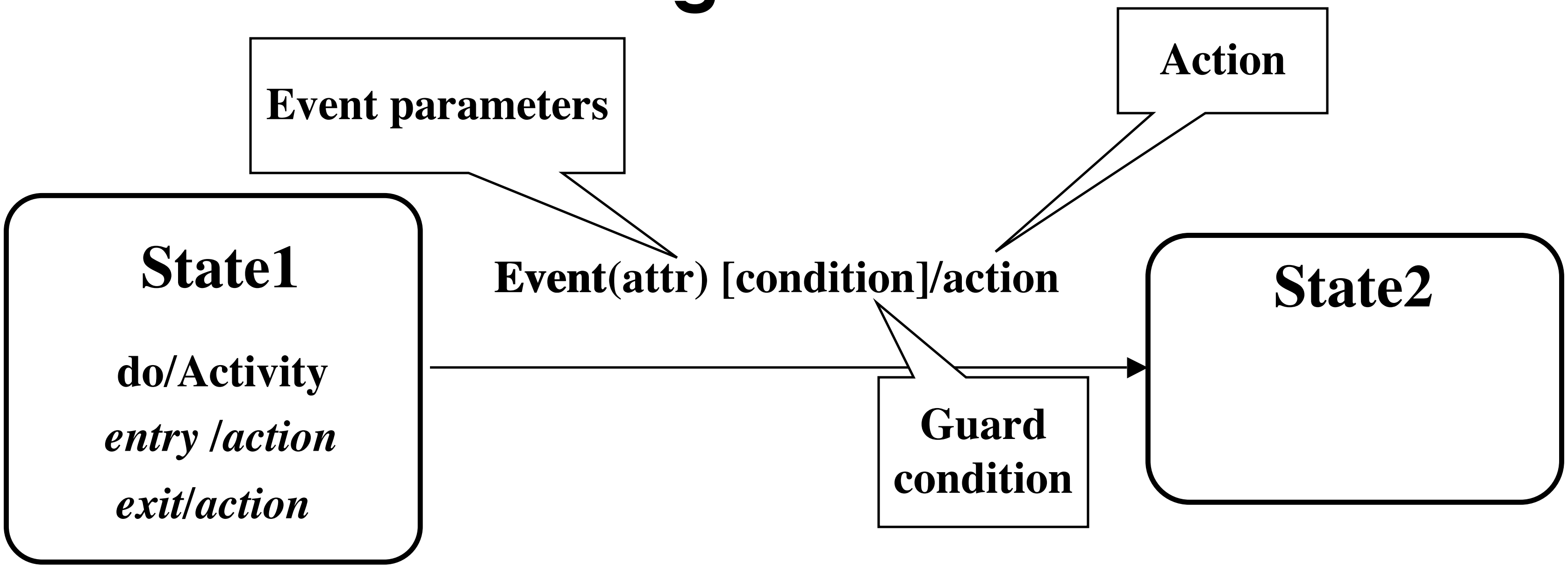
Fork or Stair?

- Object-oriented supporters claim that the stair structure is better
- Better heuristics:
  - 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

Dynamic Modeling

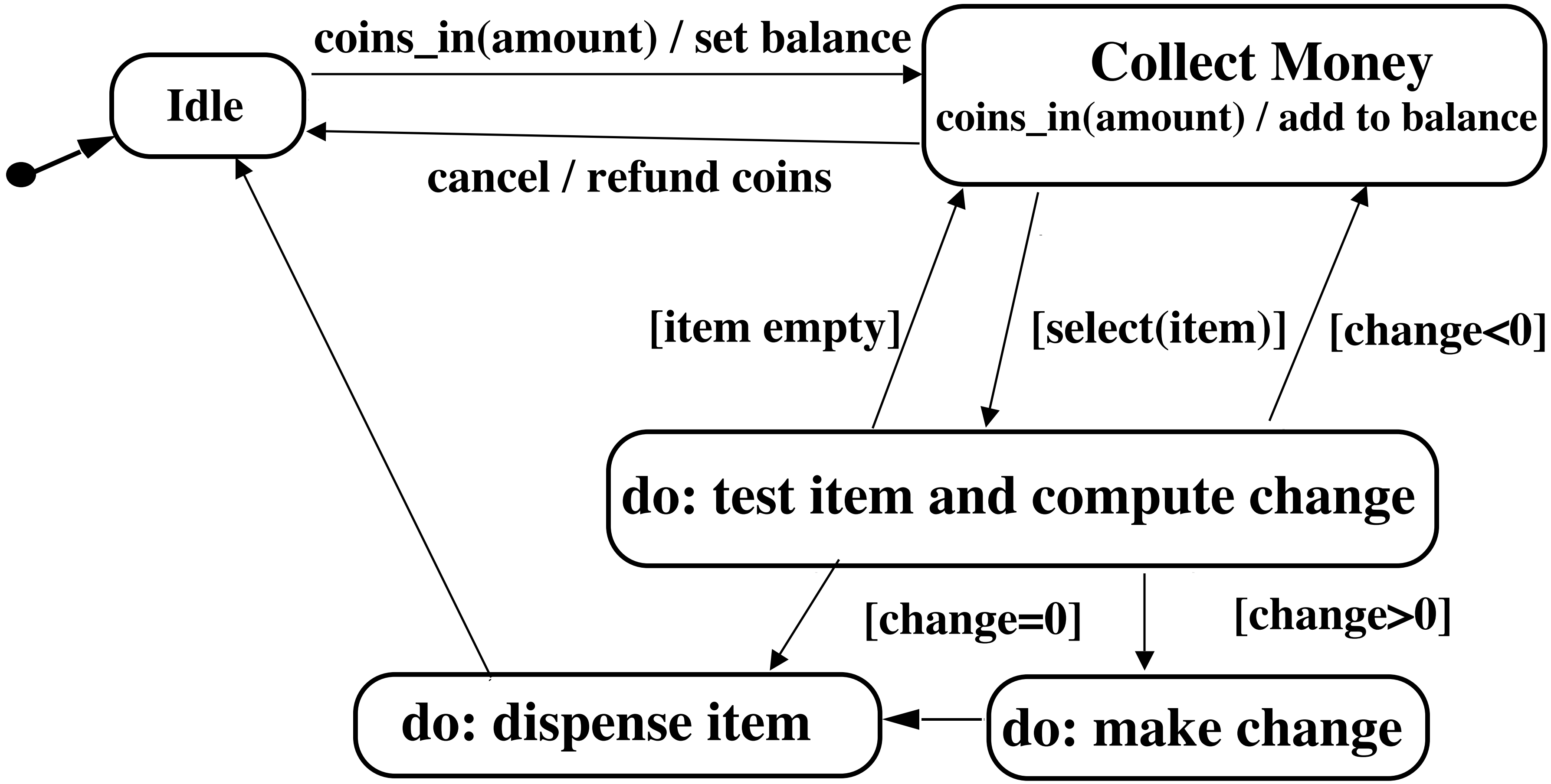
- We distinguish between two types of operations:
  - Activity: Operation that takes time to complete
    - associated with states
  - Action: Instantaneous operation
    - associated with events
- A statechart diagram relates events and states for one class
- An object model with several classes with interesting behavior has *a set* of state diagrams

UML Statechart Diagram Notation



- Notation is based on work by Harel
  - Added are a few object-oriented modifications
- A UML statechart diagram can be mapped into a finite state machine

Example of a StateChart Diagram



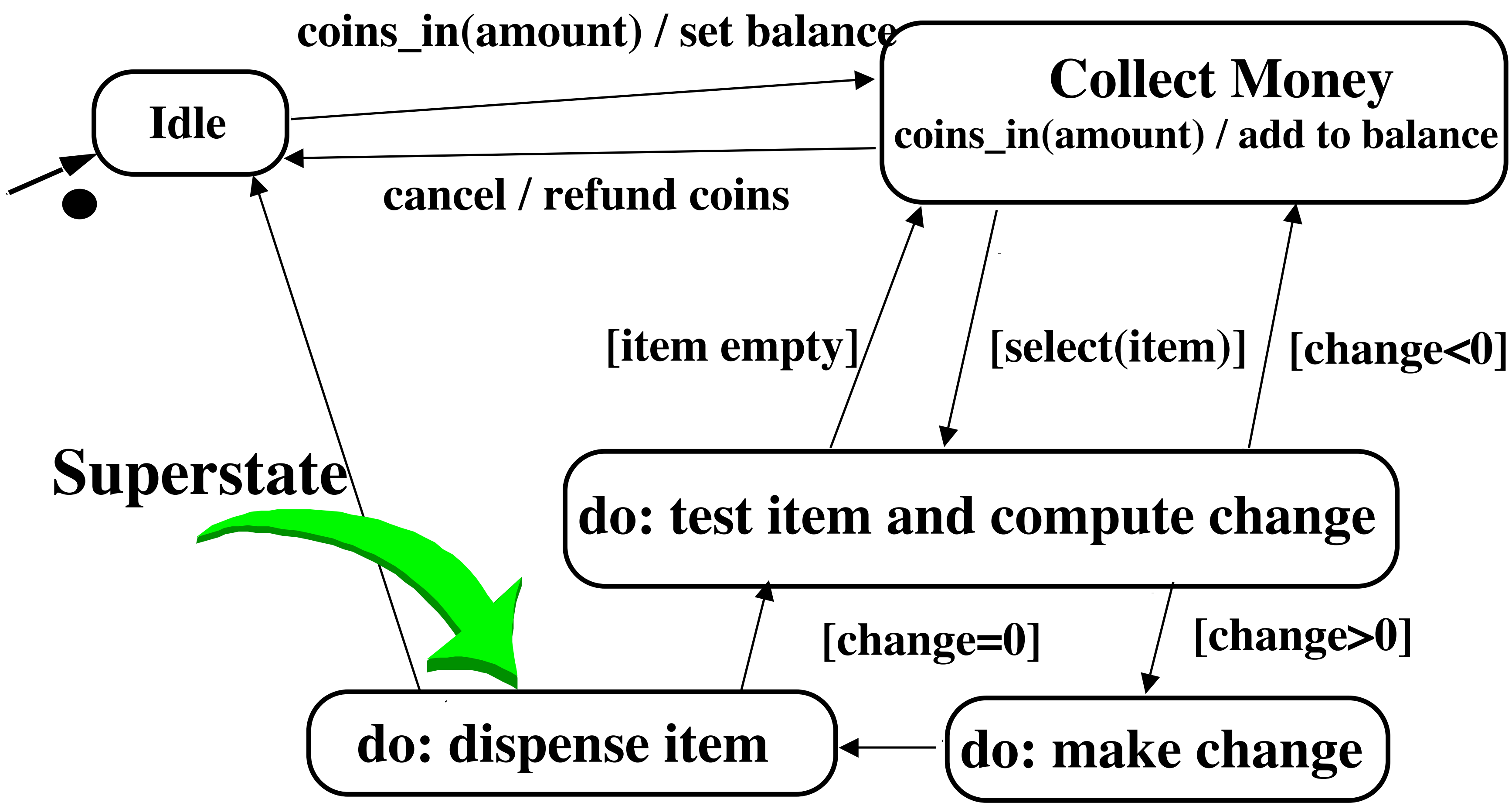
State

- 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

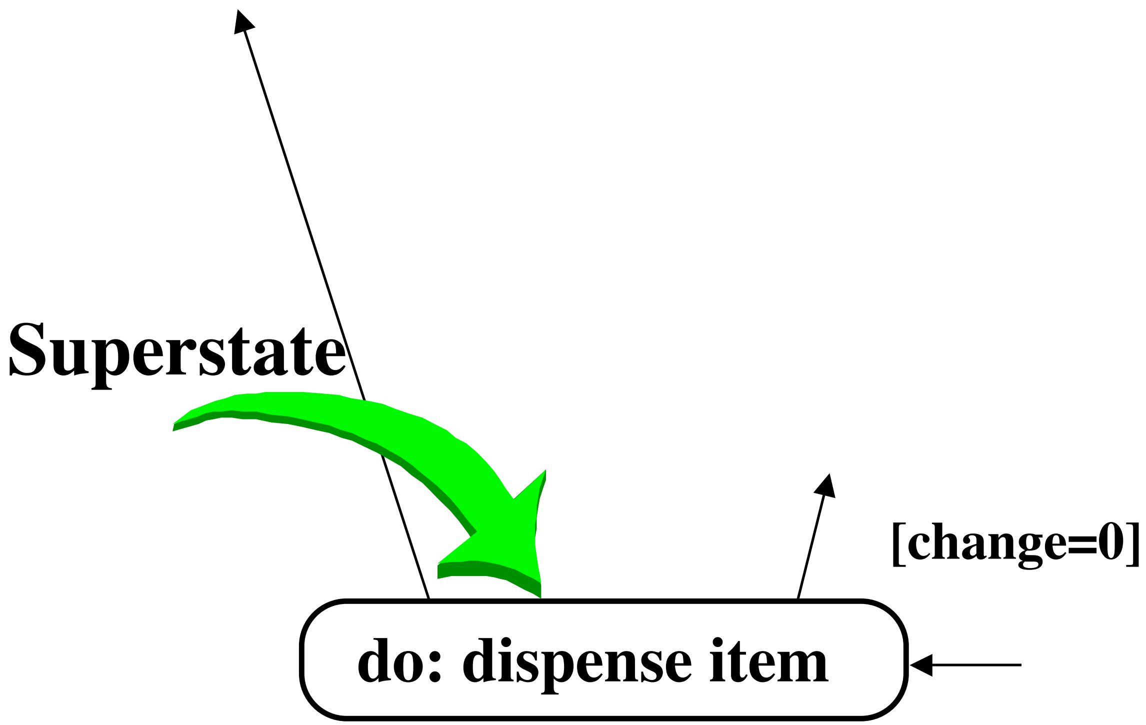
Nested State Diagram

- Activities in states can be composite items that denote other state diagrams
- A lower-level state diagram corresponds to a sequence of lower-level states and events that are invisible in the higher-level diagram.

Example of a Nested Statechart Diagram



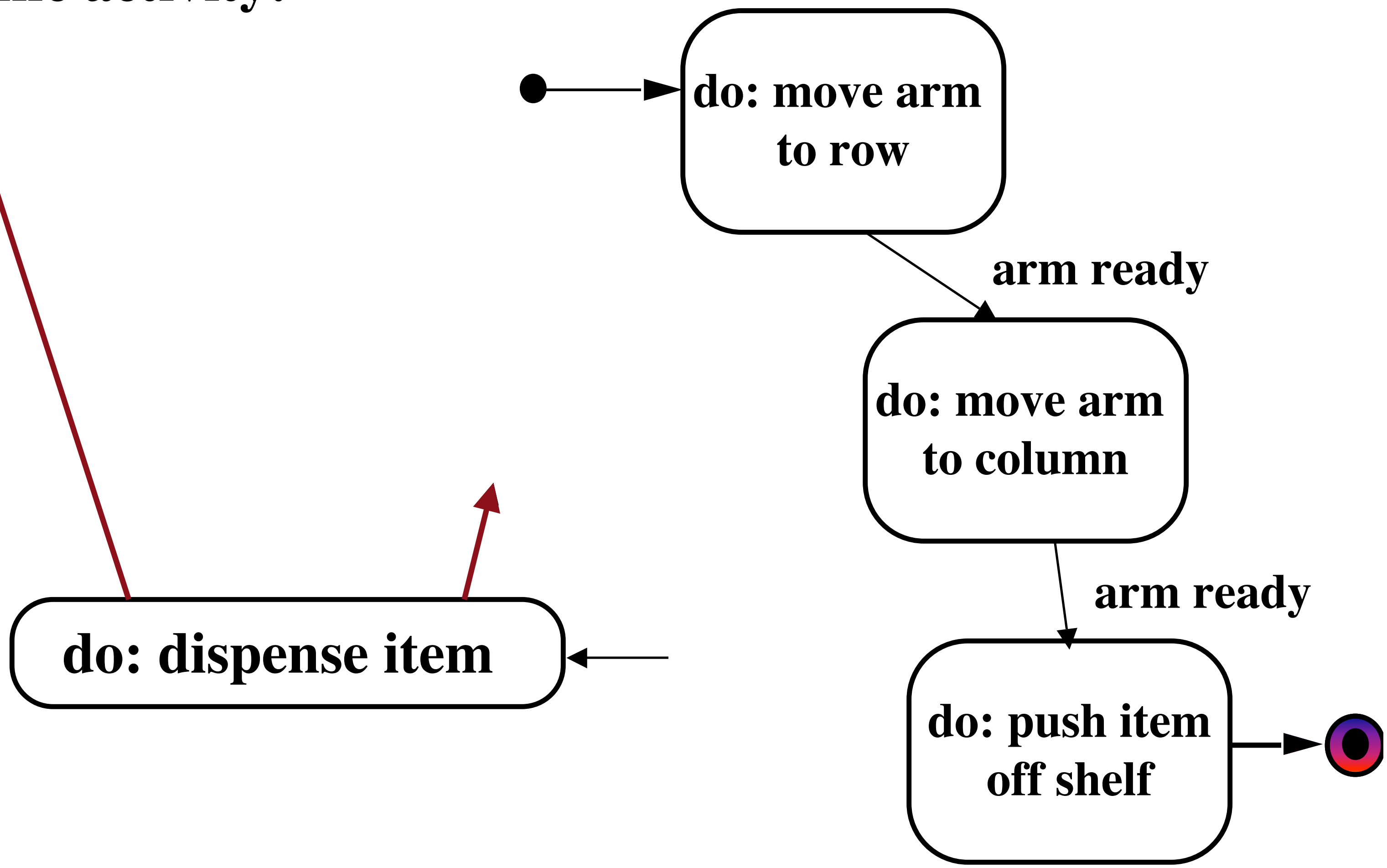
Example of a Nested Statechart Diagram



Example of a Nested Statechart Diagram

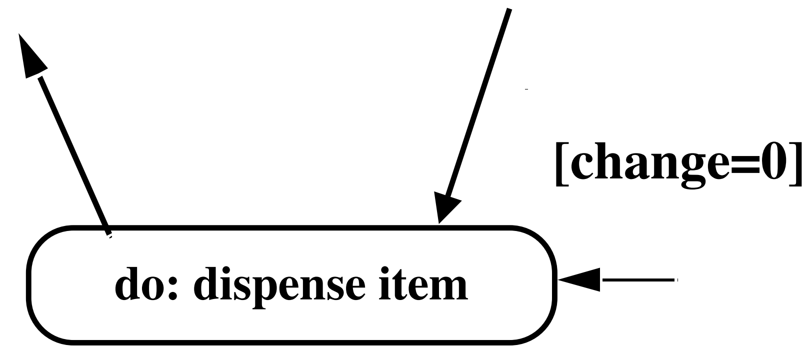
'Dispense item' as an atomic activity:

'Dispense item' as a composite activity:

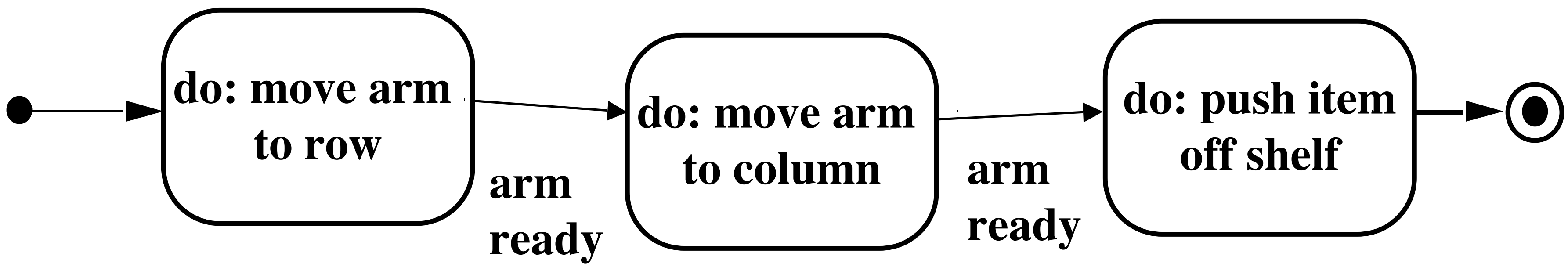


Expanding activity "do:dispense item"

'Dispense item' as an atomic activity:



'Dispense item' as a composite activity:



Superstates

- Sets of substates in a nested state diagram can be denoted with a superstate

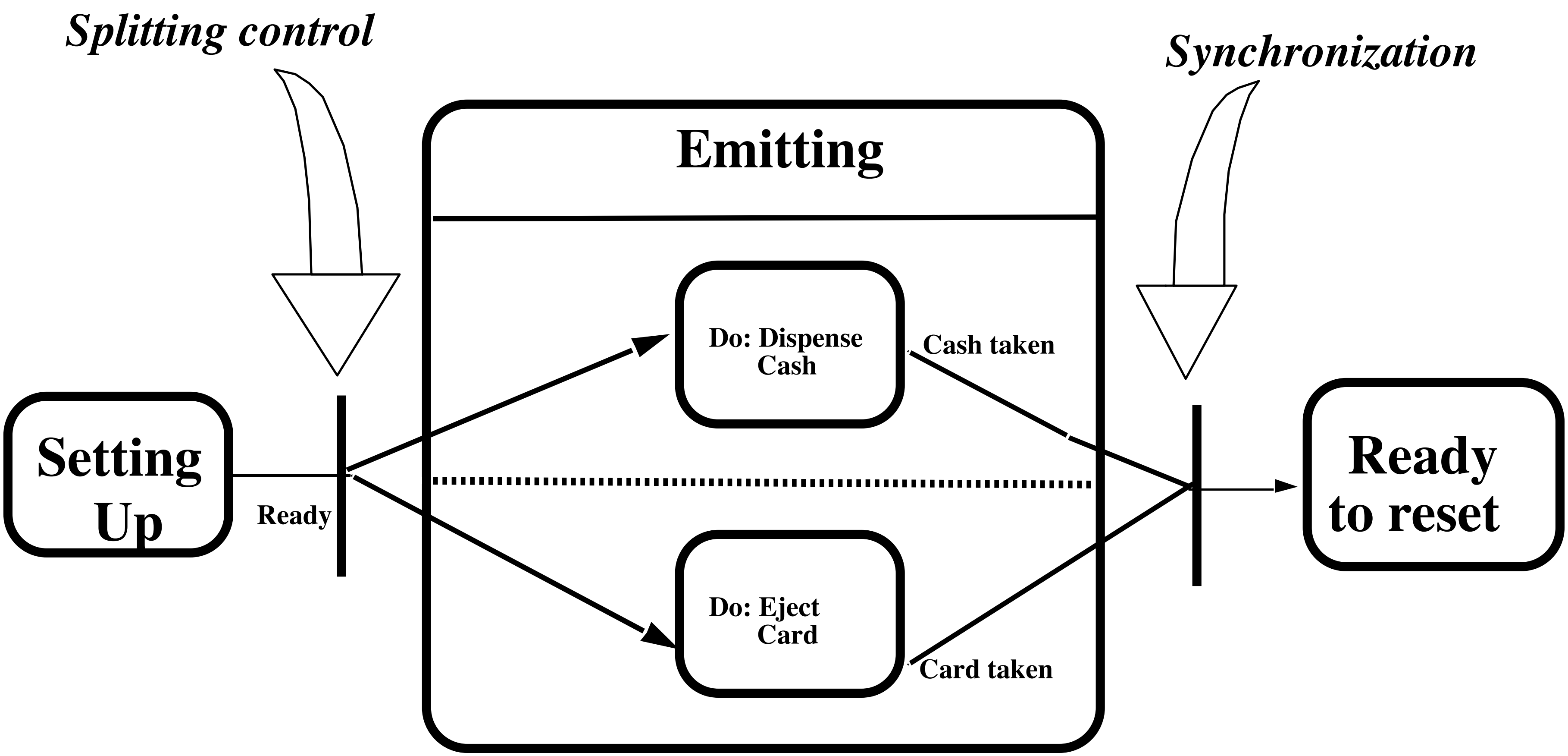
- Superstates:
  - Avoid spaghetti models
  - Reduce the number of lines in a state diagram

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



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 or activities are shown as bullets under the screen name

Navigation Path Example

Screen name

Action or Activity

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)

Summary: Requirements Analysis

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

- 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

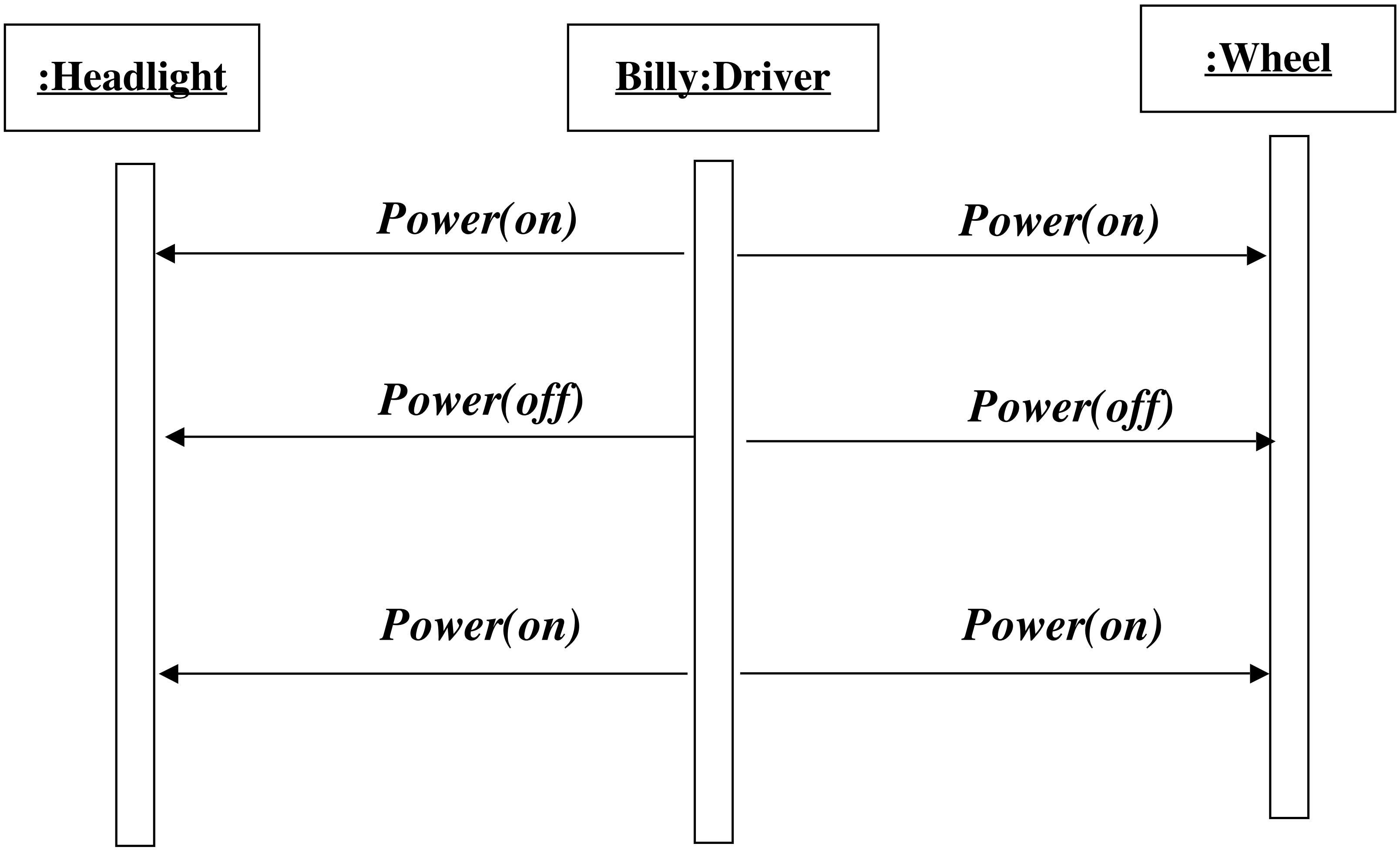
<u>Use case 5: Move car forward</u>

- 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

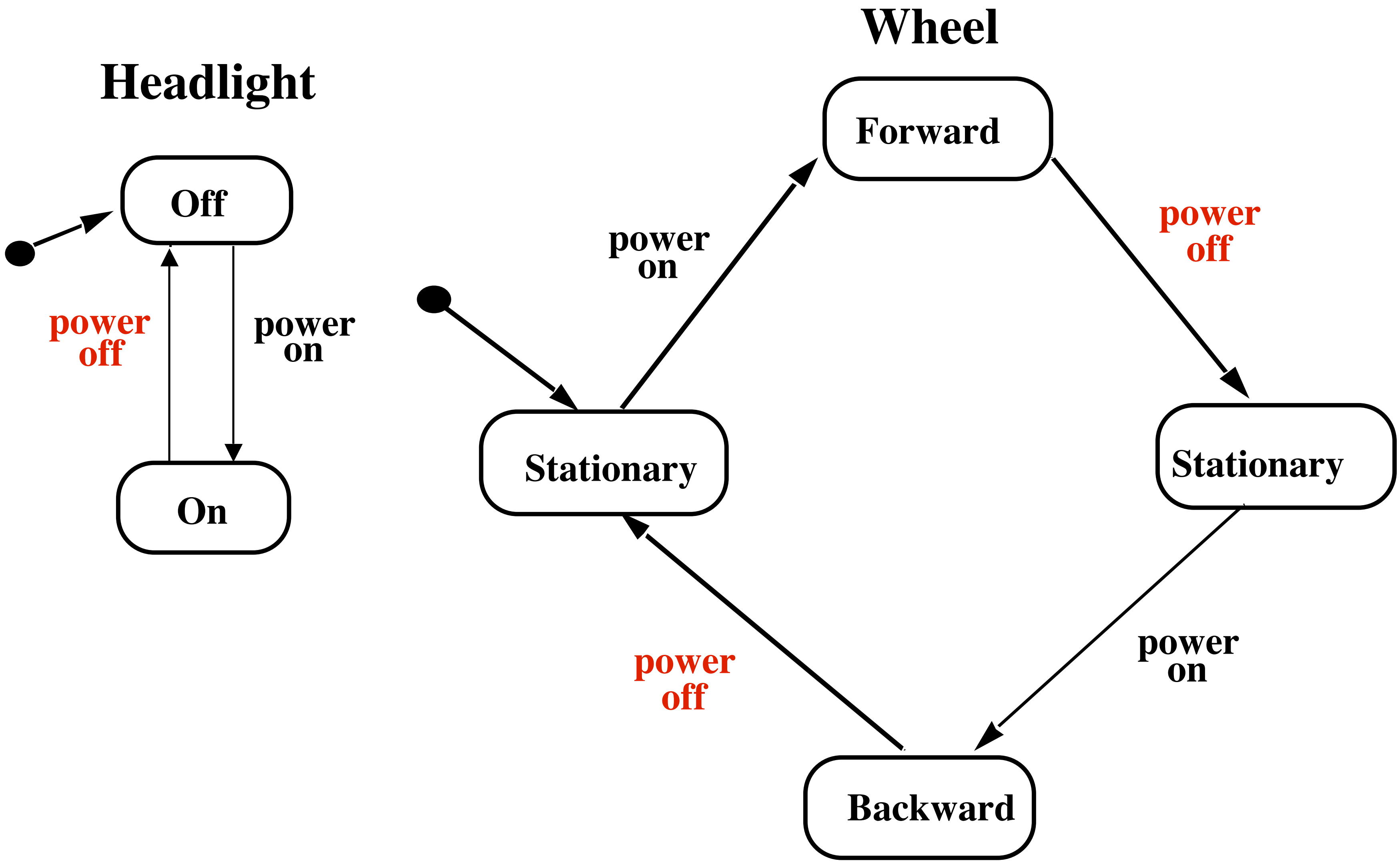
Dynamic Modeling: Create the Sequence Diagram

- 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
  - . . .

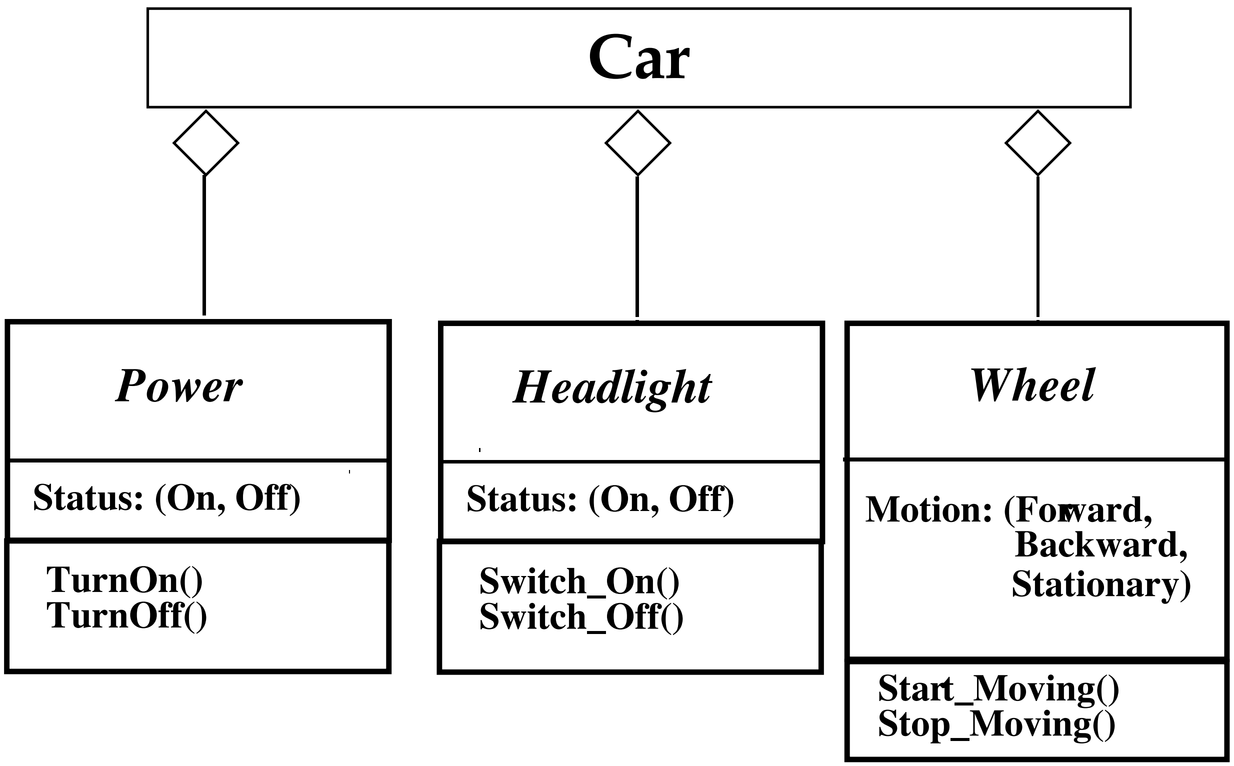
Sequence Diagram for Drive Car Scenario



Toy Car: Dynamic Model



Toy Car: Object Model



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

Dominance of Models

- Compiler:
  - The functional model most important.
  - The dynamic model is trivial because there is only one type input and only a few outputs.
- 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.

Outline of the Lecture

- ✓ Dynamic modeling
  - ✓ Sequence diagrams
  - ✓ State diagrams
- ✓ Using dynamic modeling for the design of user interfaces
- ✓ Analysis example
- → Requirements analysis model validation

Verification vs Validation of models

$$
\begin{tikzcd}
R \arrow[r, "\text{Analysis}"] \arrow[d, "f_R"'] & M_{\text{Analysis}} \arrow[r, "\text{System Design}"] \arrow[d, "f_{MA}"'] & M_{\text{System}} \arrow[r, "\text{Object Design}"] \arrow[d, "f_{MS}"'] & M_{\text{Object}} \arrow[r, "\text{Implementation}"] \arrow[d, "f_{MD}"'] & M_{\text{Impl}} \arrow[d, "f_{Impl}"'] \\
R \arrow[r] & M_{\text{Analysis}} \arrow[r] & M_{\text{System}} \arrow[r] & M_{\text{Object}} \arrow[r] & M_{\text{Impl}}
\end{tikzcd}
$$

$\xleftarrow{\text{Validation}}$ $\xleftarrow{\text{Verification}}$ $\xleftarrow{\text{Verification}}$ $\xleftarrow{\text{Verification}}$

$$
\begin{tikzcd}
M \arrow[r, "f_M"] & M \\
R \arrow[u, "I"] \arrow[r, "f_R"'] & R \arrow[u, "I"']
\end{tikzcd}
$$

Verification and Validation

- 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, Realistism

Checklist for a Requirements Review

- Is the model correct?
  - A model is correct if it represents the client's view of the 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

Checklist for the Requirements Review (2)

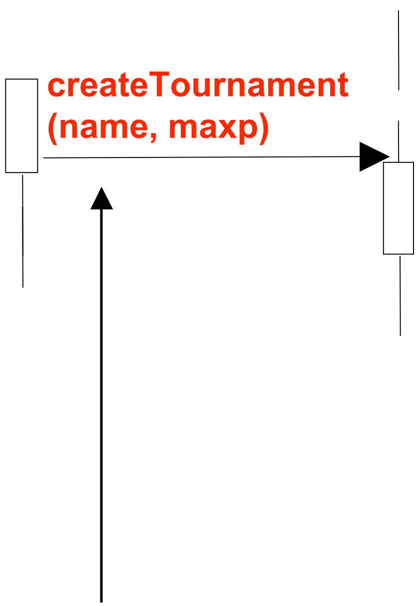
- Syntactical check of the models
  - Check for consistent naming of classes, attributes, methods in different subsystems
  - Identify dangling associations (“pointing to nowhere”)
  - Identify double- defined classes
  - Identify missing classes (mentioned in one model but not defined anywhere)
  -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

Examples for syntactical Problems

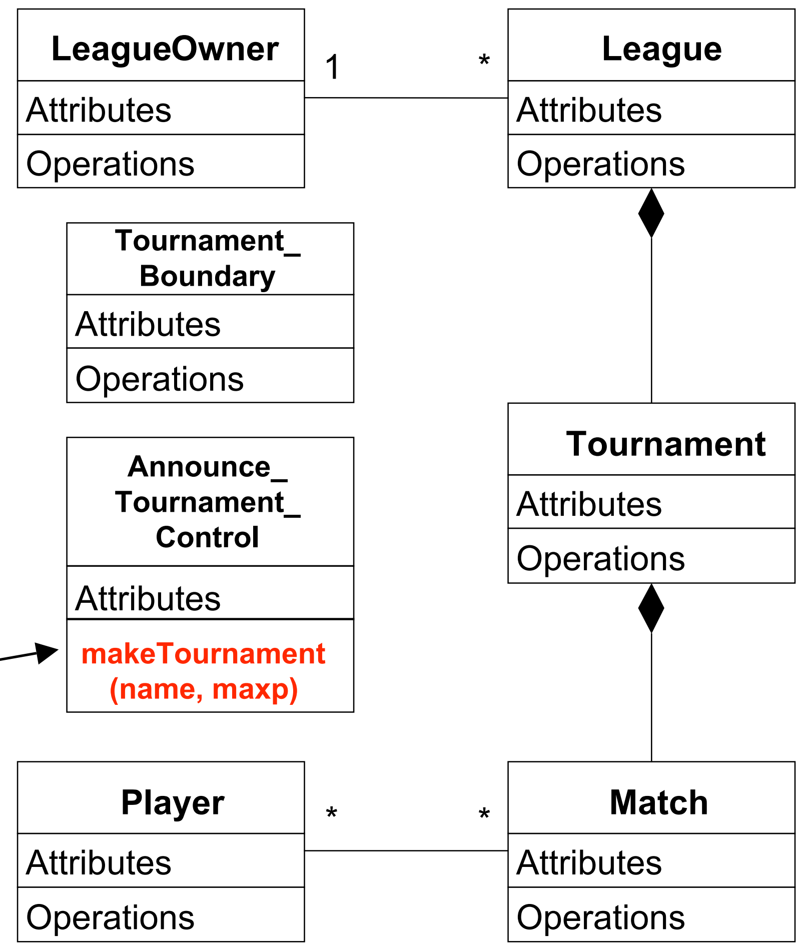
- Different spellings in different UML diagrams
- Omissions in diagrams

Different spellings in different UML diagrams

UML Sequence Diagram



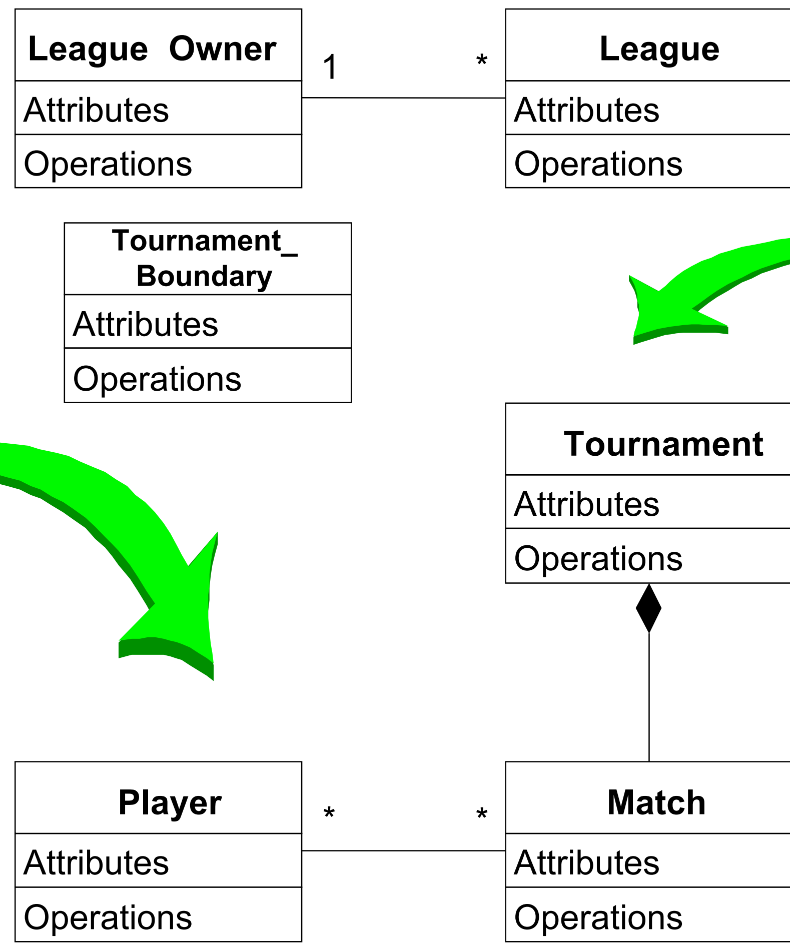
UML Class Diagram



Different spellings
in different models
for the same operation

Omissions in some UML Diagrams

Class Diagram



Missing Association (Incomplete Analysis?)

Missing class (The control object Announce_Tournament is mentioned in the sequence diagram)

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 Model

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

Summary

- 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.

- In addition, we described the requirements analysis document and its components

Backup slides

Is this a good Sequence Diagram?