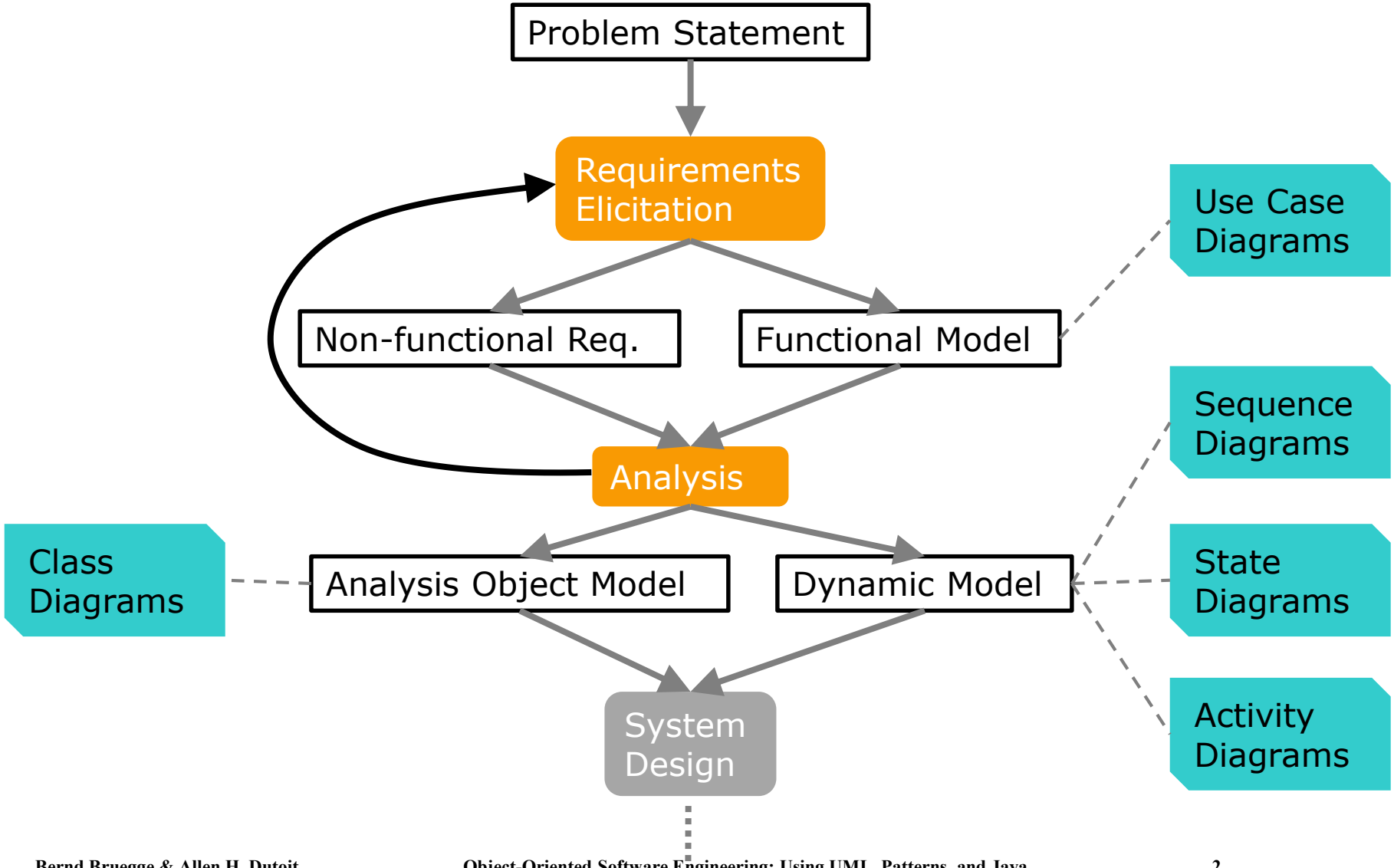
Chapter 5, Analysis: Dynamic Modeling

An overview of OOSE development activities and their products



Dynamic Modeling

- Describe components of the system that have **interesting** dynamic behavior, using
  - State diagrams: One state diagram per class with interesting dynamic behavior
  - Sequence diagrams: For interaction between classes
  - Activity diagrams: Model (complex) logic (business rules) captured by a use case
- Purpose:
  - Detect and supply operations for the object model.

How do we detect Operations?

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

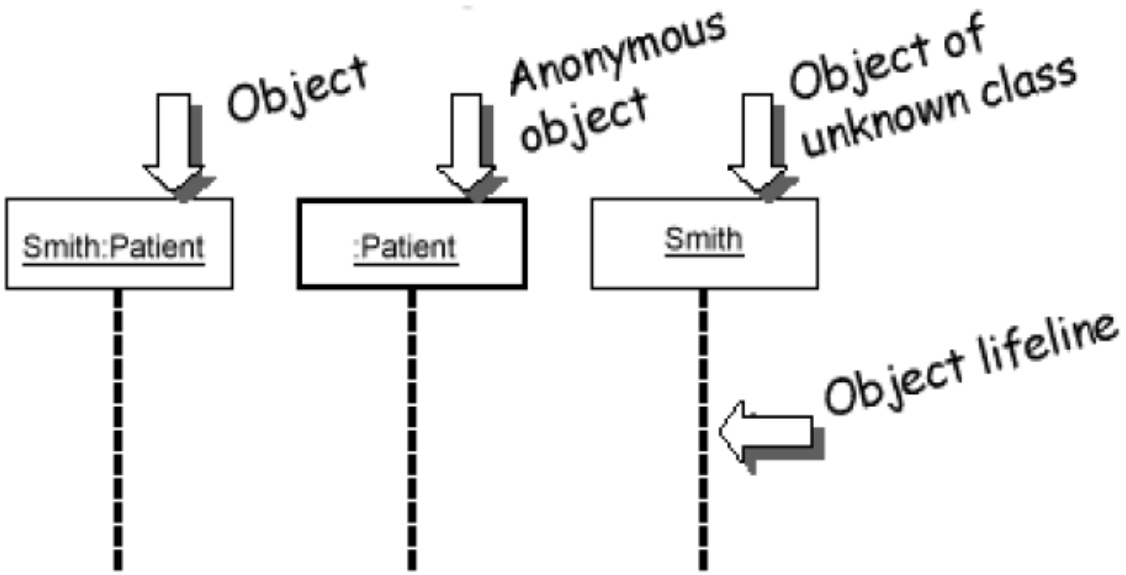
Sequence Diagram

- A Sequence diagram ties uses cases with objects
- Shows how the behavior of a use case is distributed among its objects
- Use cases (ends users) vs Sequence diagrams (developers)
- A graphical description of objects participating in a use case using a DAG notation
- Heuristic for finding participating objects:
  - An event always has a sender and a receiver
  - Find them for each event => These are the objects participating in the use case.

Representing Objects

squares with object type, optionally preceded by object name and colon

- write object's name if it clarifies the diagram

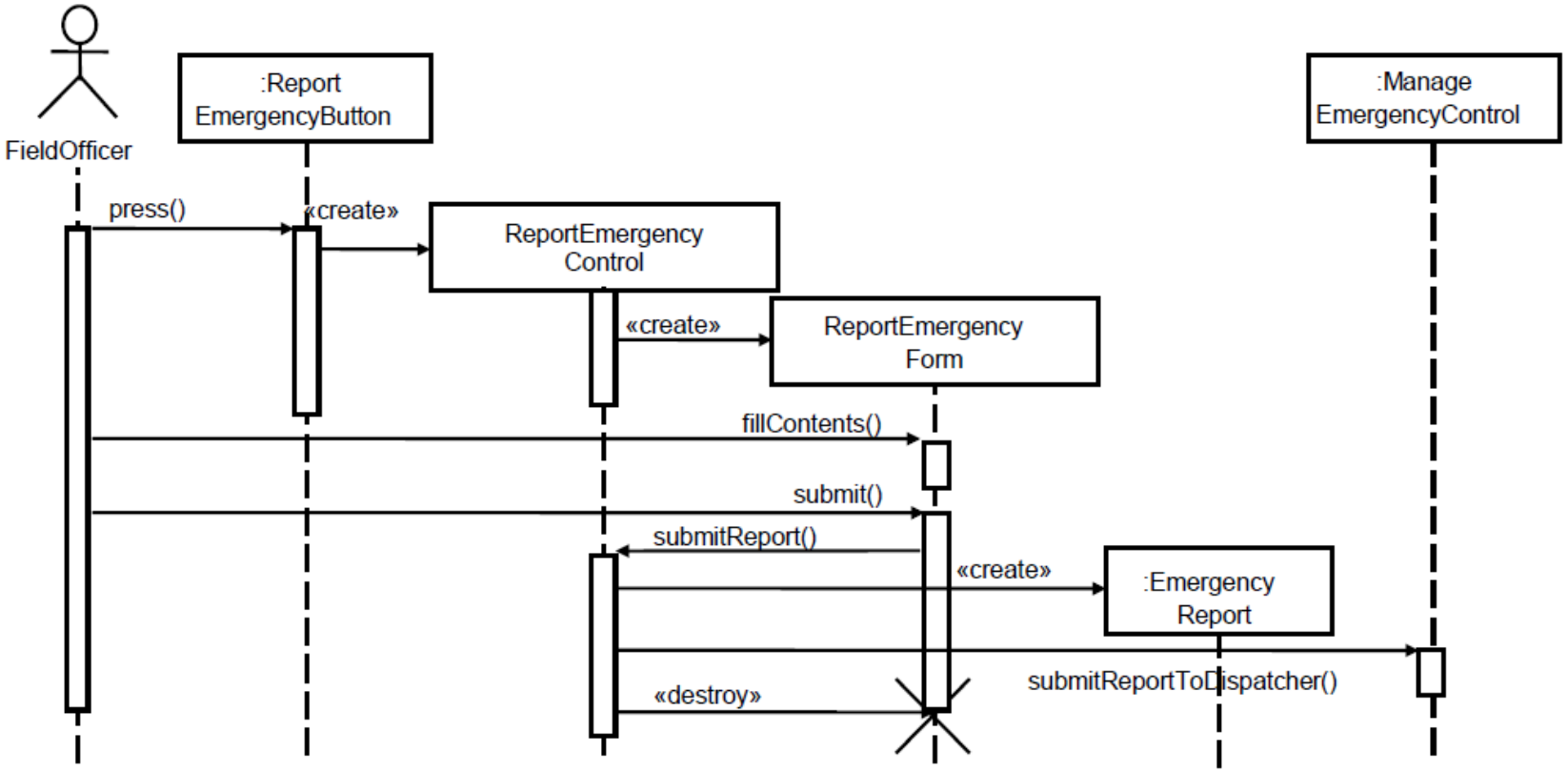


Name syntax: <objectname>:<classname>

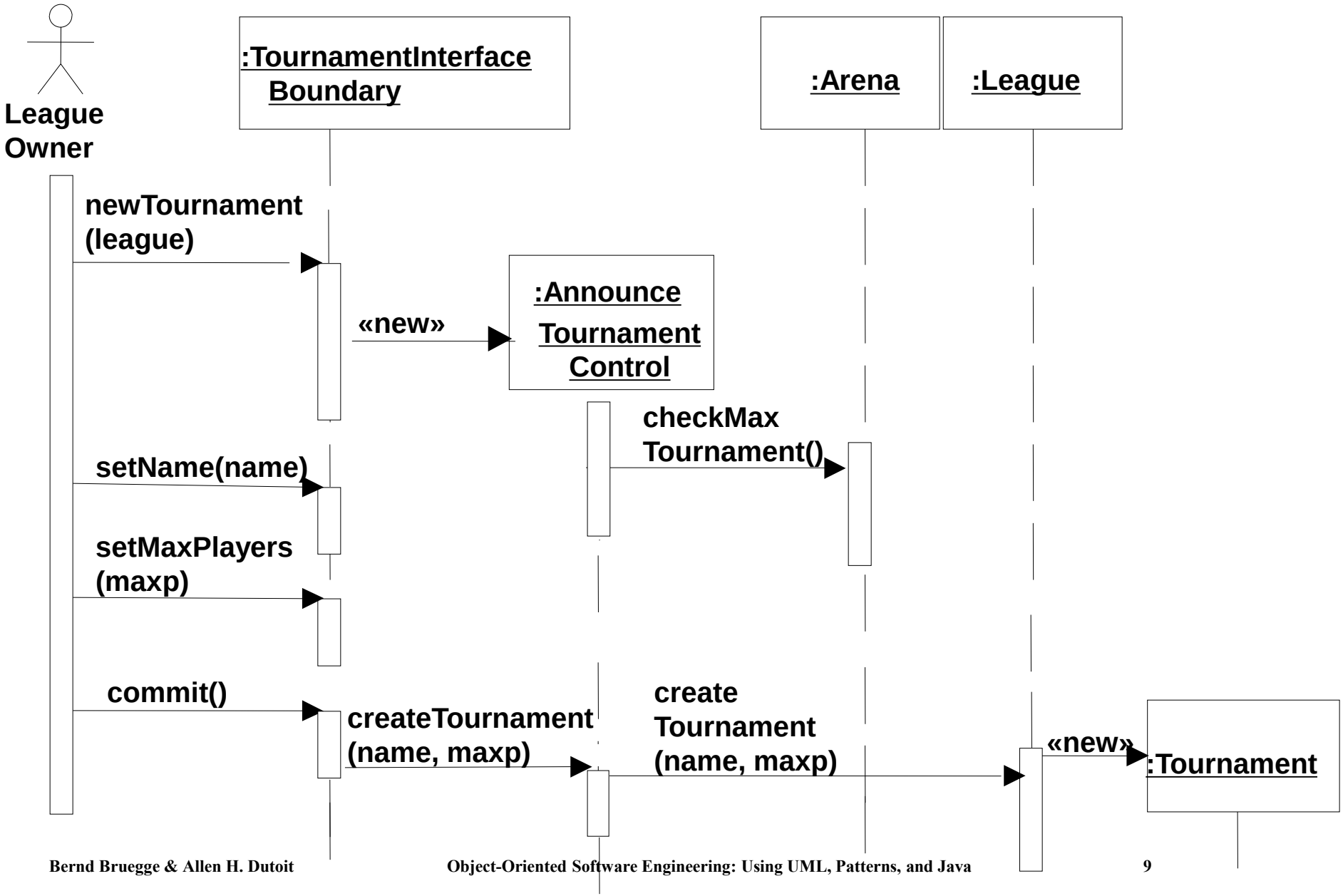
Heuristics for Sequence Diagram

- Layout:
  - 1st column: actor of use case
  - 2nd column: a boundary object
  - 3rd column: control object managing rest of use case
- Creation of objects:
  - Create control objects at beginning of event flow
  - Control objects create boundary and entity objects
- Access of objects:
  - Entity objects can be accessed by control objects
  - Entity objects should not access boundary or control objects.

Report Emergency



ARENA Sequence Diagram: Create Tournament



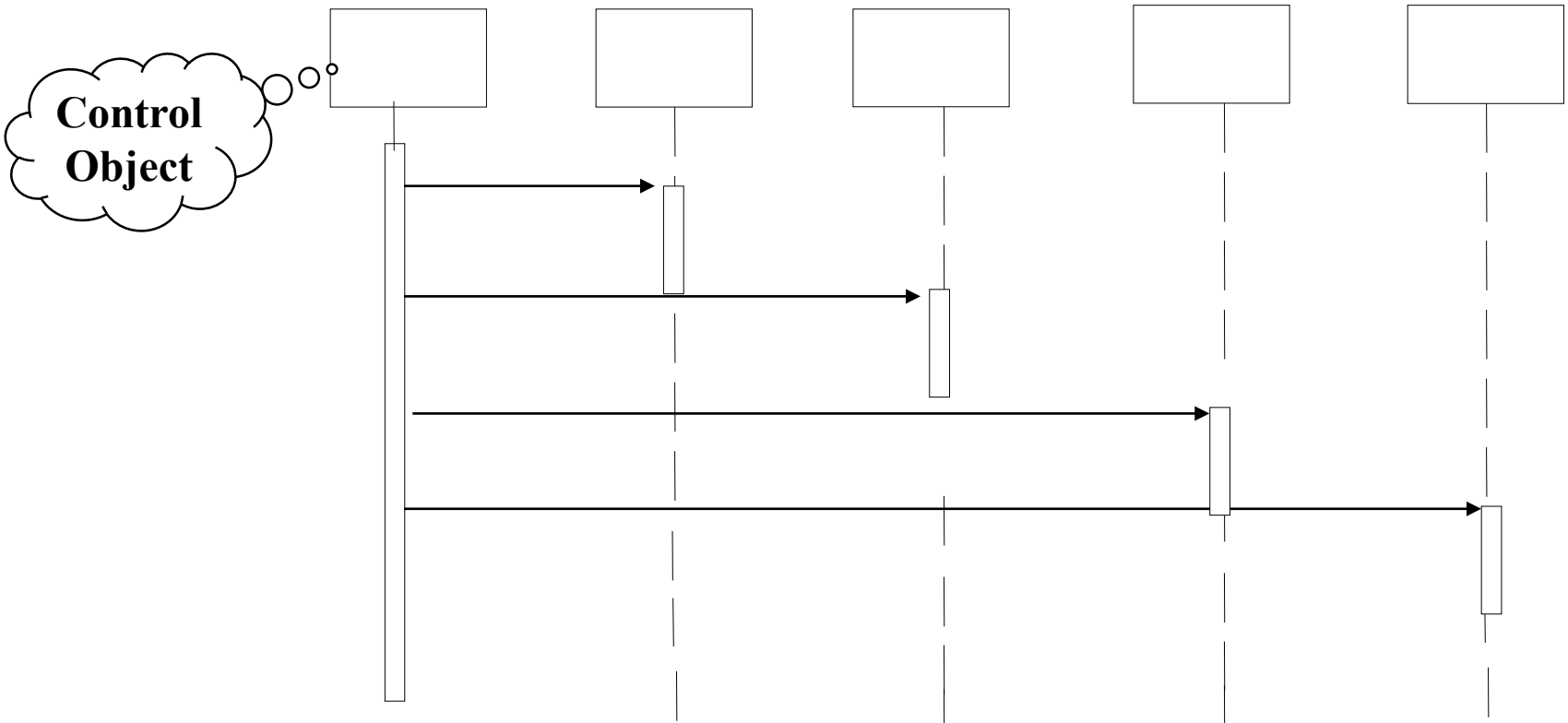
What else can we get out of Sequence Diagrams?

- 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)

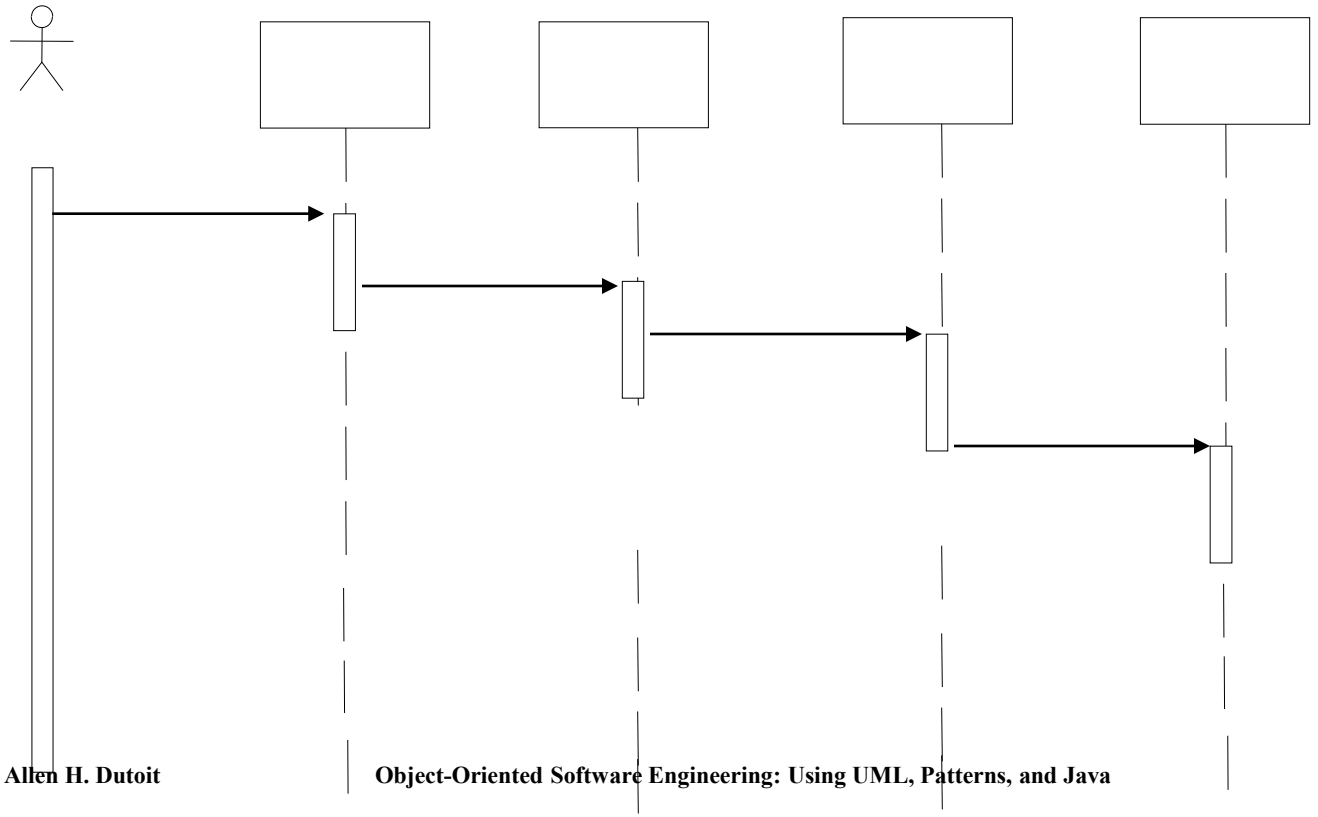
Fork Diagram

- 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

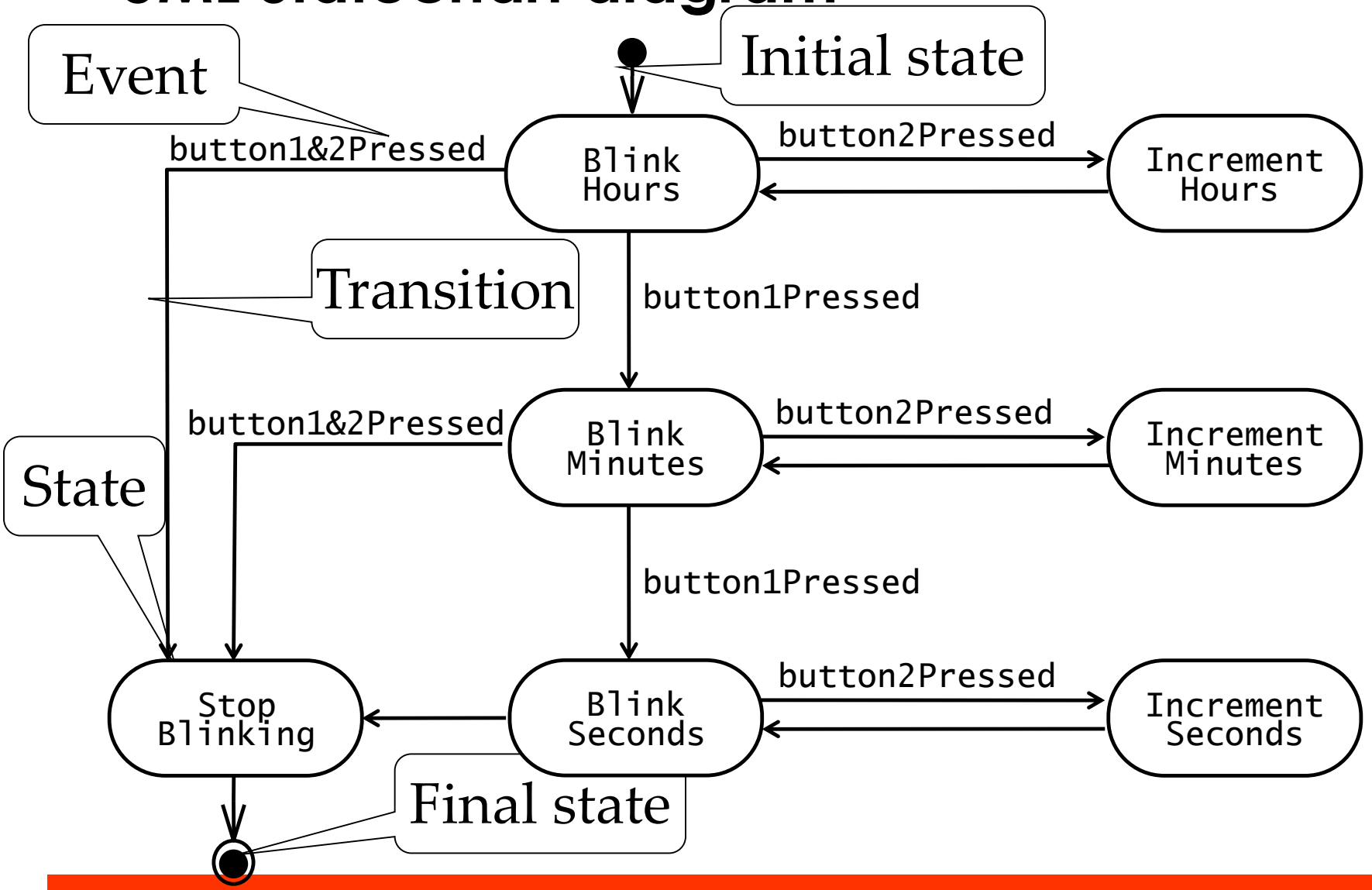
- The dynamic behavior is distributed. Each object delegates responsibility to other objects
  - Each object knows only a few of the other objects and knows which objects can help with a specific behavior



Fork or Stair?

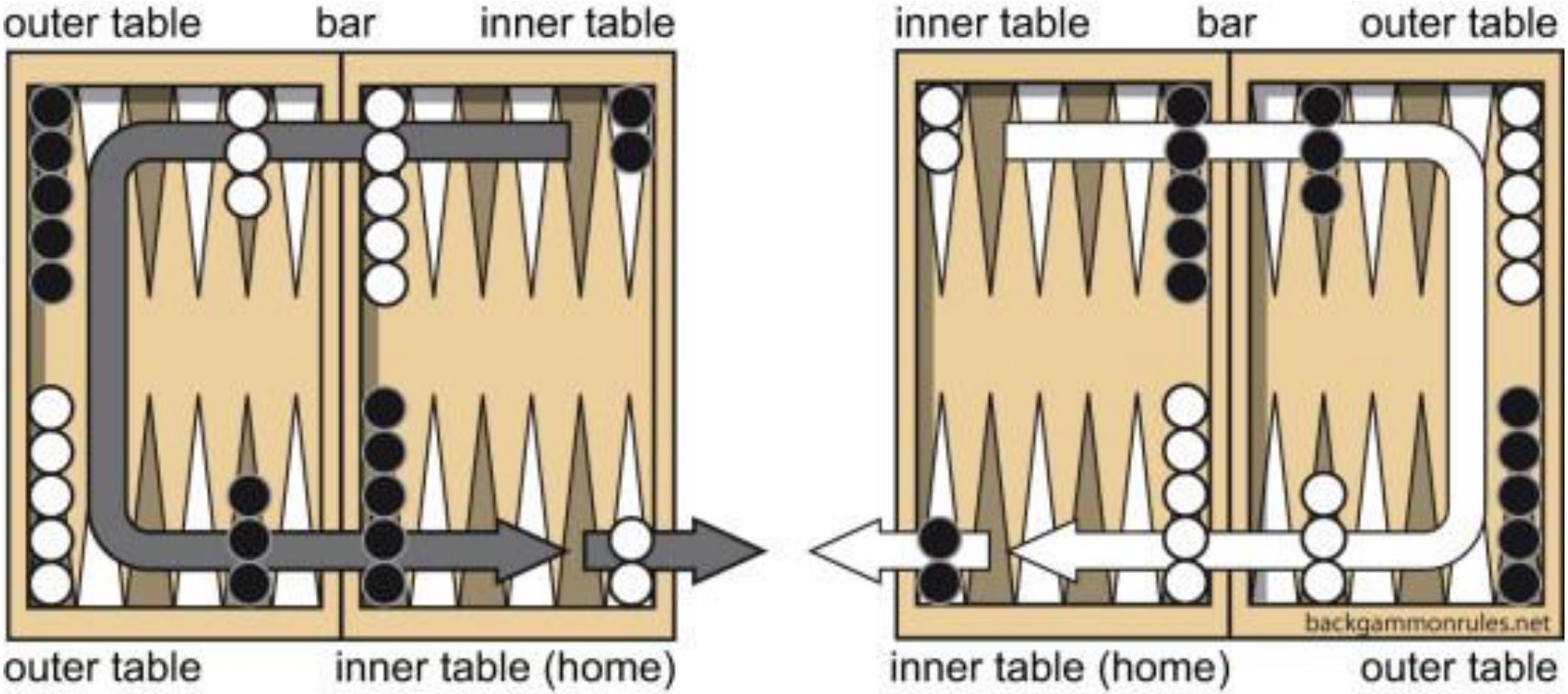
- Object-oriented supporters claim that the stair structure is better
- Modeling Advice:
  - Choose the stair - a decentralized control structure - if
    - The operations have a strong connection
    - The operations will always be performed in the same order
  - Choose the fork - a centralized control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

UML Statechart diagram



Represents behavior of *a single object* with interesting dynamic behavior.

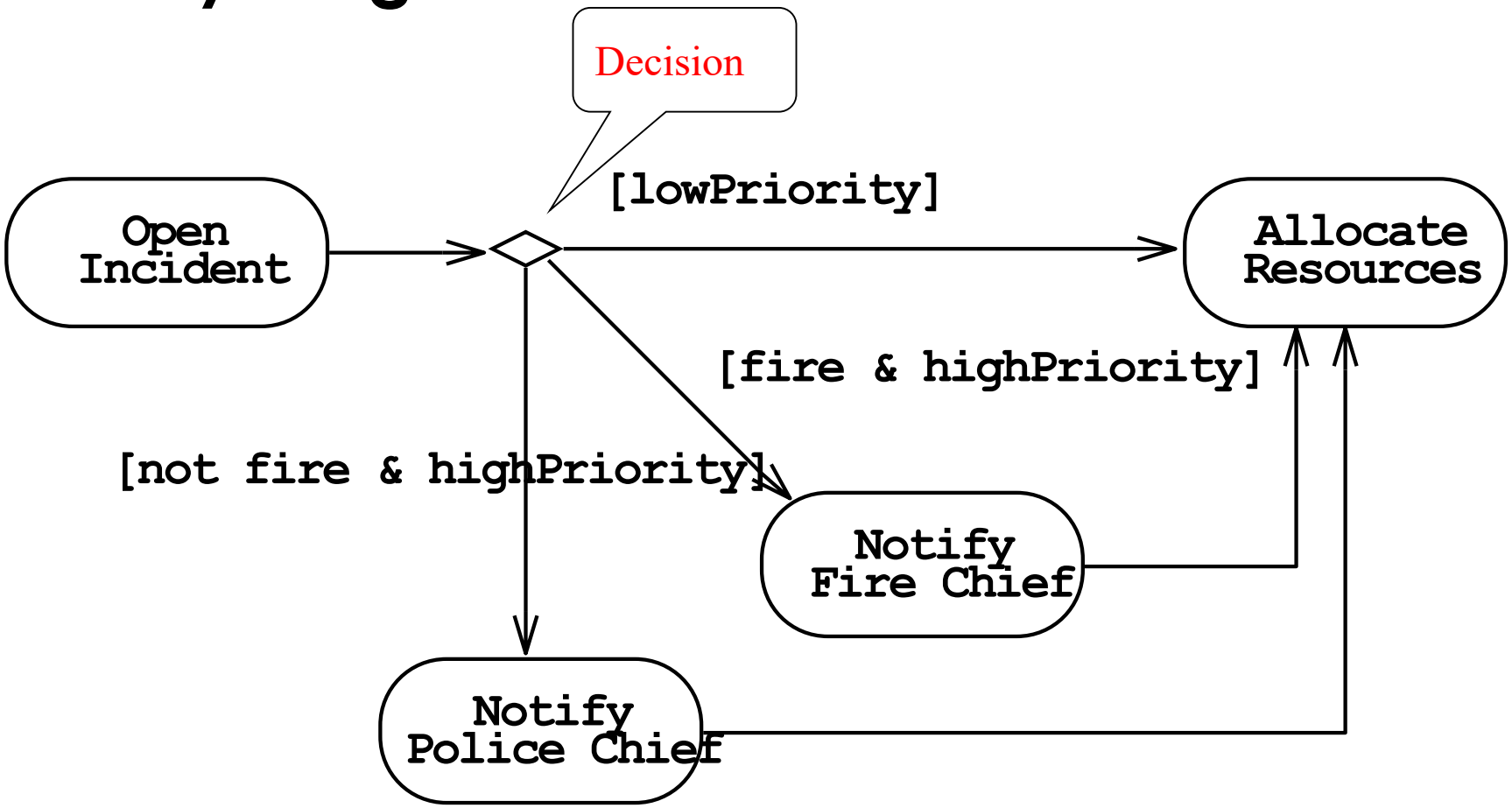
Backgammon


outer table
bar
inner table
outer table
inner table (home)
inner table
bar
outer table
backgammonrules.net
inner table (home)
outer table

Activity Diagrams

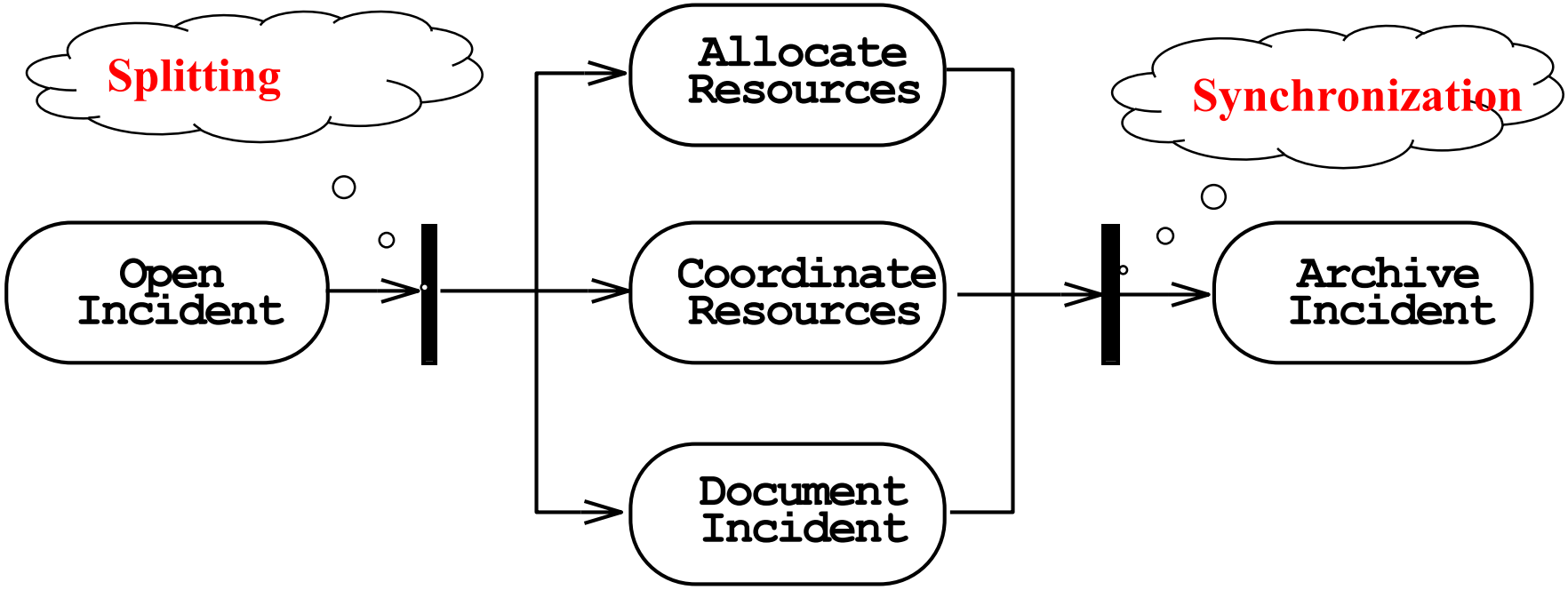
- An activity diagram is useful to depict the workflow in a system

Activity Diagrams allow to model Decisions



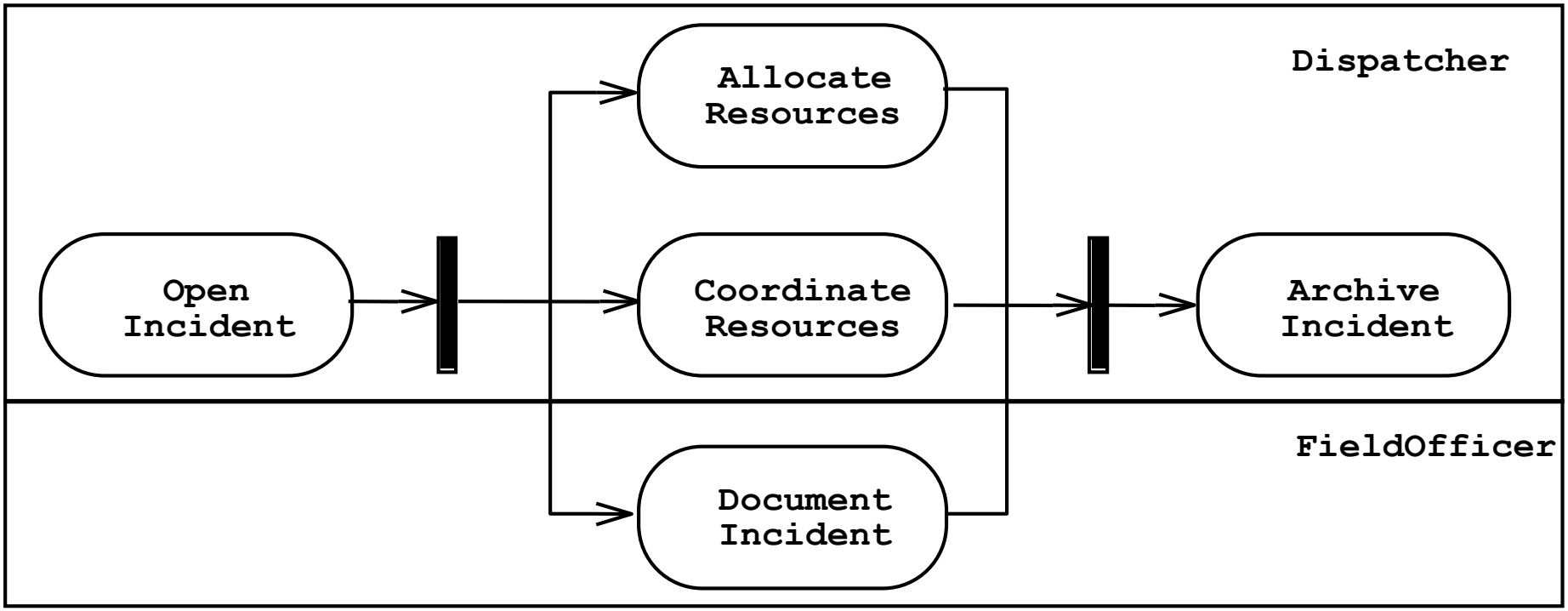
Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting 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.



Practical Tips for Dynamic Modeling

- Construct dynamic models only for (avoid "analysis paralysis"):
  - Classes with significant dynamic behavior and
  - Use cases that are non-trivial
- Consider only relevant attributes
  - Use abstraction if necessary
- Look at granularity of application when deciding on actions and activities
- Reduce notational clutter
  - Try to put actions into super-state boxes (look for identical actions on events leading to the same state).

Checklist for a Requirements Review

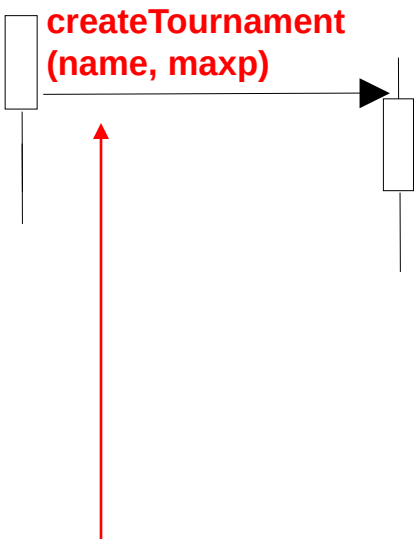
- Is the model correct?
  - Represents client's view of the system?
- Is the model complete?
  - Every scenario described?
- Is the model consistent?
  - Has components that contradict?
- Is the model unambiguous?
  - Describes one system, not many?
- Is the model realistic?
  - Can be implemented?

Examples for syntactical Problems

- Different spellings in different UML diagrams
- Omissions in diagrams

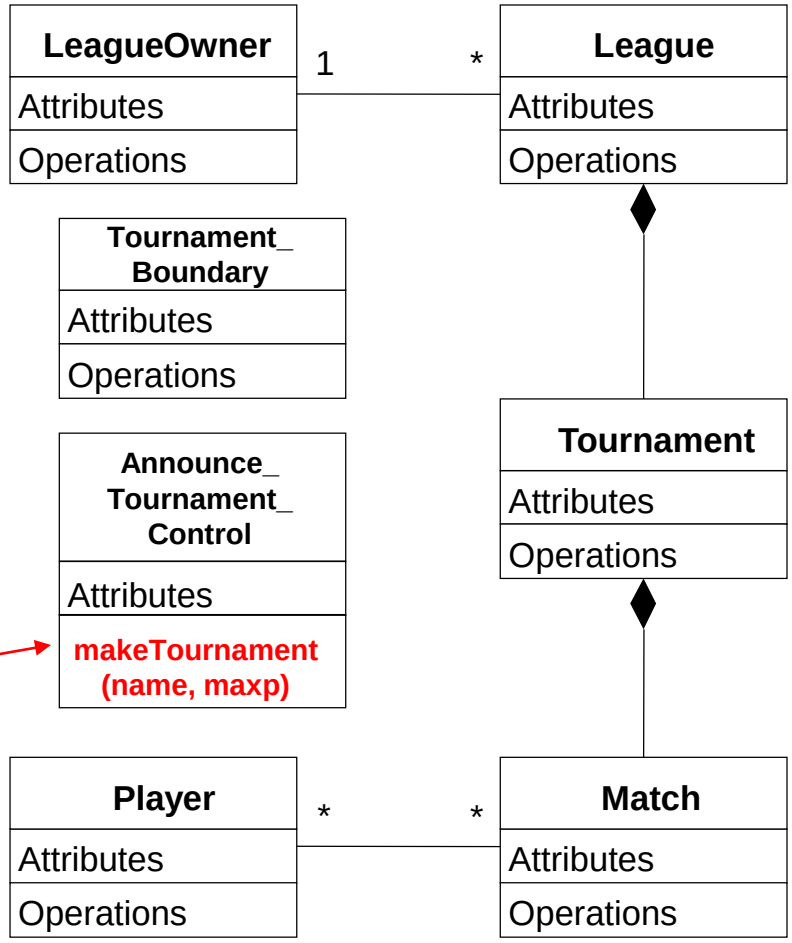
Different spellings in different UML diagrams

UML Sequence Diagram



Different spellings
in different models
for the same operation

UML Class Diagram



Checklist for a Requirements Review (2)

- Syntactical check of models
  - Consistent naming of classes, attributes, methods in different subsystems
  - Dangling associations ("pointing to nowhere")
  - Doubly-defined classes
  - Missing classes (mentioned in one model but not defined anywhere)
  - Classes with 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 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)
- Activity diagrams for complex business rules/logic

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. Associations between them? Their multiplicity?
   - What are the attributes of objects? Operations on objects?

- 3. What is its behavior?

  Create *sequence diagrams*  **Dynamic Modeling**

  - 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

  Create *activity diagrams*

Summary

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