Object-Oriented Software Engineering
Conquering Complex and Changing Systems

Chapter 5, Analysis: Dynamic Modeling

Bernd Brügge
Allen H. Dutoit

Technische Universität München
Applied Software Engineering

22 November 2001

Outline of the Lecture

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

A realistic example: ARENA

- ARENA Problem Statement available on SE Discuss bboard
  - **http://tum.globalse.org/teaching/ws01/SE/discuss.nsf/f7069403ac0dc217852566270075 4e21/1f1d9871dd61c5f3c1256b0a003c484c?OpenDocument**
- We will be developing the requirements analysis document (RAD) for ARENA concurrently to the class.
- You can participant by using the REQuest tool to write one or more use cases corresponding to this problem statement.
- Request an account from Allen Dutoit and then log into the ARENA system of REQuest:
  - **http://atbruegge13.in.tum.de:8080/arena/servlet/SYSLogin**

How do you find classes?

- In previous lectures we have already established the following sources
  - **Application domain analysis: Talk to client to identify abstractions**
  - **Application of general world knowledge and intuition**
  - **Scenarios**
    - **Natural language formulation of a concrete usage of the system**
  - **Use Cases**
    - **Natural language formulation of the functions of the system**
  - **Textual analysis of problem statement (Abbot)**
- Today we show how 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 also candidates for objects**

Dynamic Modeling with UML

- Diagrams for dynamic modeling
  - ***Interaction diagrams*** **describe the dynamic behavior between objects**
  - ***Statecharts*** **describe the dynamic behavior of a single object**
- Interaction diagrams
  - **Sequence Diagram:**
    - **Dynamic behavior of a set of objects arranged in time sequence.**
    - **Good for real-time specifications and complex scenarios**
  - **Collaboration Diagram :**
    - **Shows the relationship among objects. Does not show time**
- 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:
  - **A collection of multiple state chart diagrams, one state chart diagram for each class with important dynamic behavior.**
- Purpose:
  - **Detect and supply methods 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**

Start with Flow of Events from Use Case

- Flow of events from “Dial a Number” Use case:
  - **Caller lifts receiver**
  - **Dail tone begins**
  - **Caller dials**
  - **Phone rings**
  - **Callee answers phone**
  - **Ringing stops**
  - **....**

What is an Event?

- Something that happens at a point in time
- Relation of events to each other:
  - **Causally related: Before, after,**
  - **Causally unrelated: concurrent**
- An event sends information from one object to another
- Events can be grouped in event classes with a hierarchical structure. ‘Event’ is often used in two ways:
  - **Instance of an event class: “New IETM issued on Thursday September 14 at 9:30 AM”.**
    - **Event class “New IETM”, Subclass “Figure Change”**
  - **Attribute of an event class**
    - **IETM Update (9:30 AM, 9/14/99)**
    - **Car starts at (4:45pm, Monroeville Mall, Parking Lot 23a)**
    - **Mouse button down(button#, tablet-location)**

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 objects participating in a use case or scenario using a DAG (direct 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:
  - **An event always has a sender and a receiver. Find them for each event => These are the objects participating in the use case**

An Example

- Flow of events in a “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

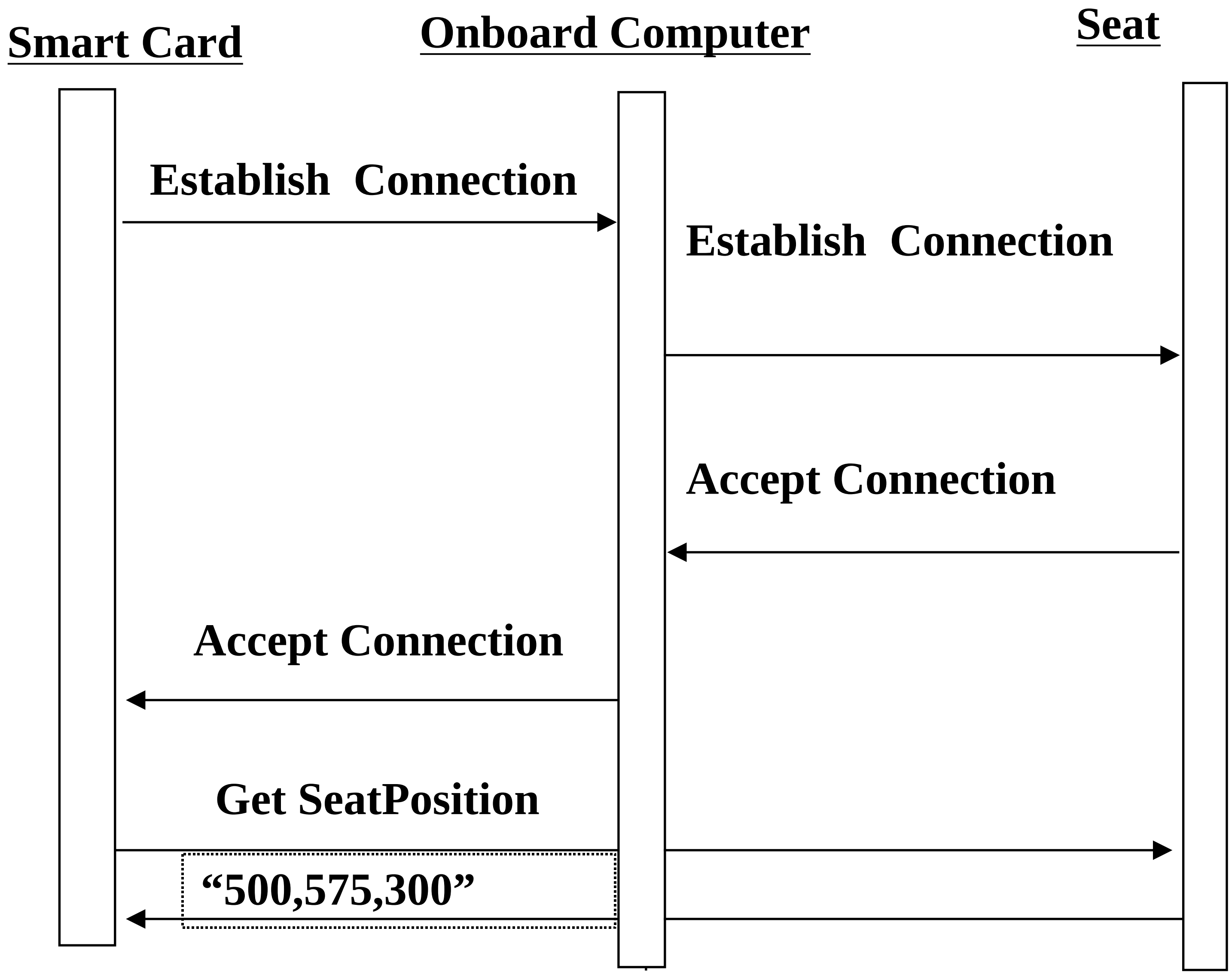
- Which are the objects?

Sequence Diagram for “Get SeatPosition”

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



time

Heuristics for Sequence Diagrams

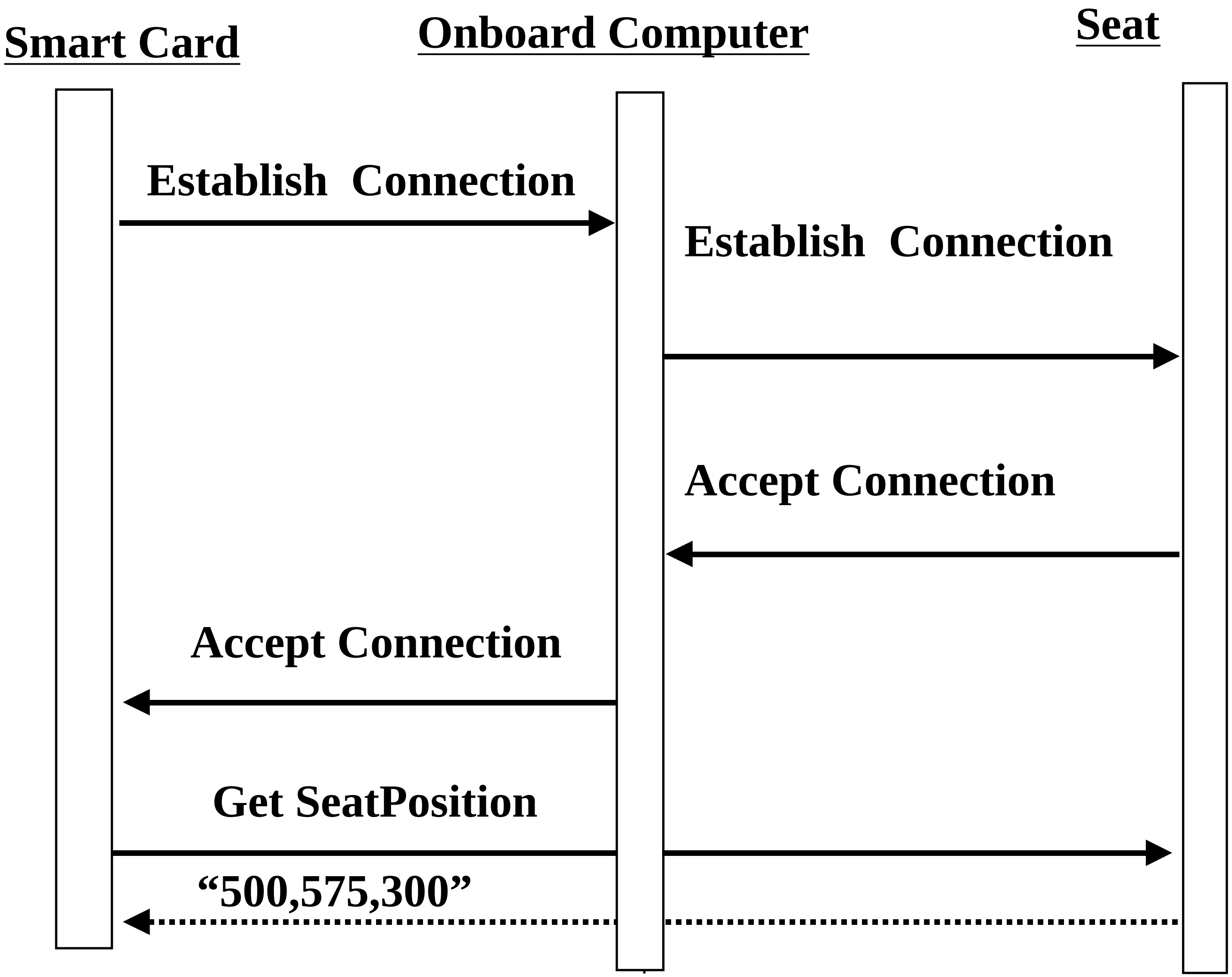
- Layout:
  - **1st column: Should correspond to the actor who initiated the use case**
  - **2nd column: Should be a boundary object**
  - **3rd column: Should be the control object that manages the rest of the use case**
- Creation:
  - **Control objects are created at the initiation of a use case**
  - **Boundary objects are created by control objects**
- Access:
  - **Entity objects are accessed by control and boundary objects,**
  - **Entity objects should never call boundary or control objects: This makes it easier to share entity objects across use cases and makes entity objects resilient against technology-induced changes in boundary objects.**

Is this a good Sequence Diagram?

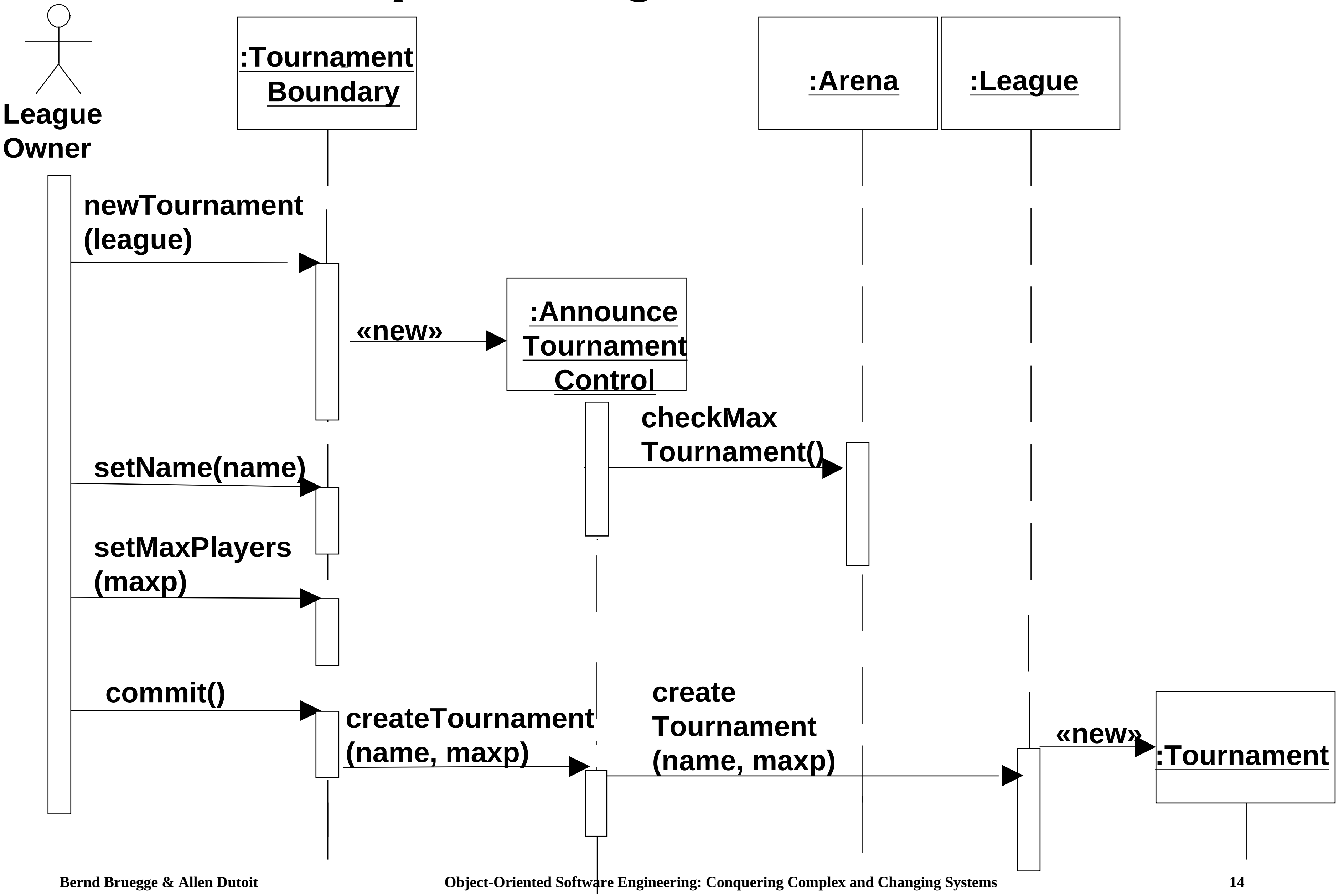
- First column is not the actor

- It is not clear where the boundary object is

- It is not clear where the control object is

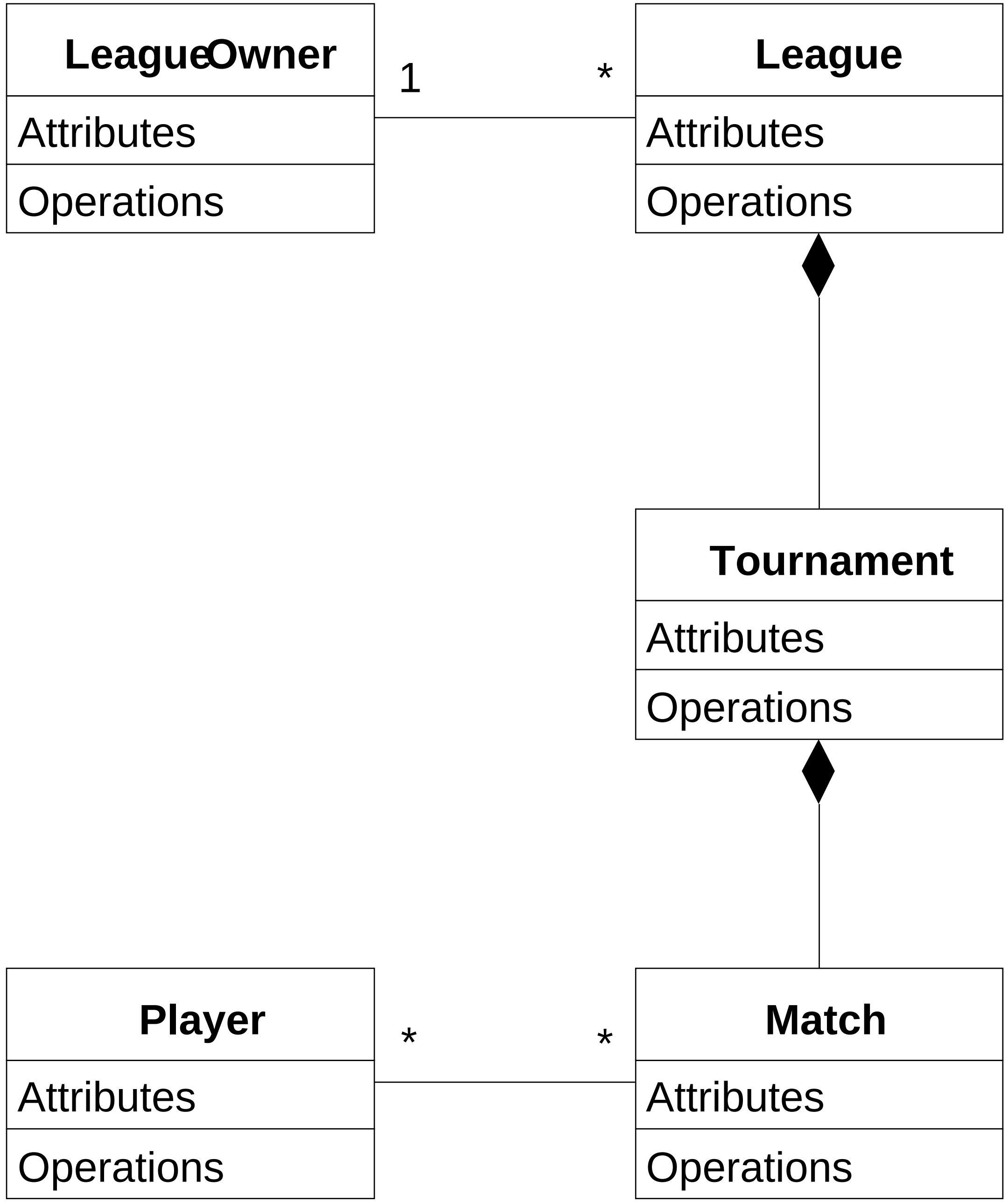


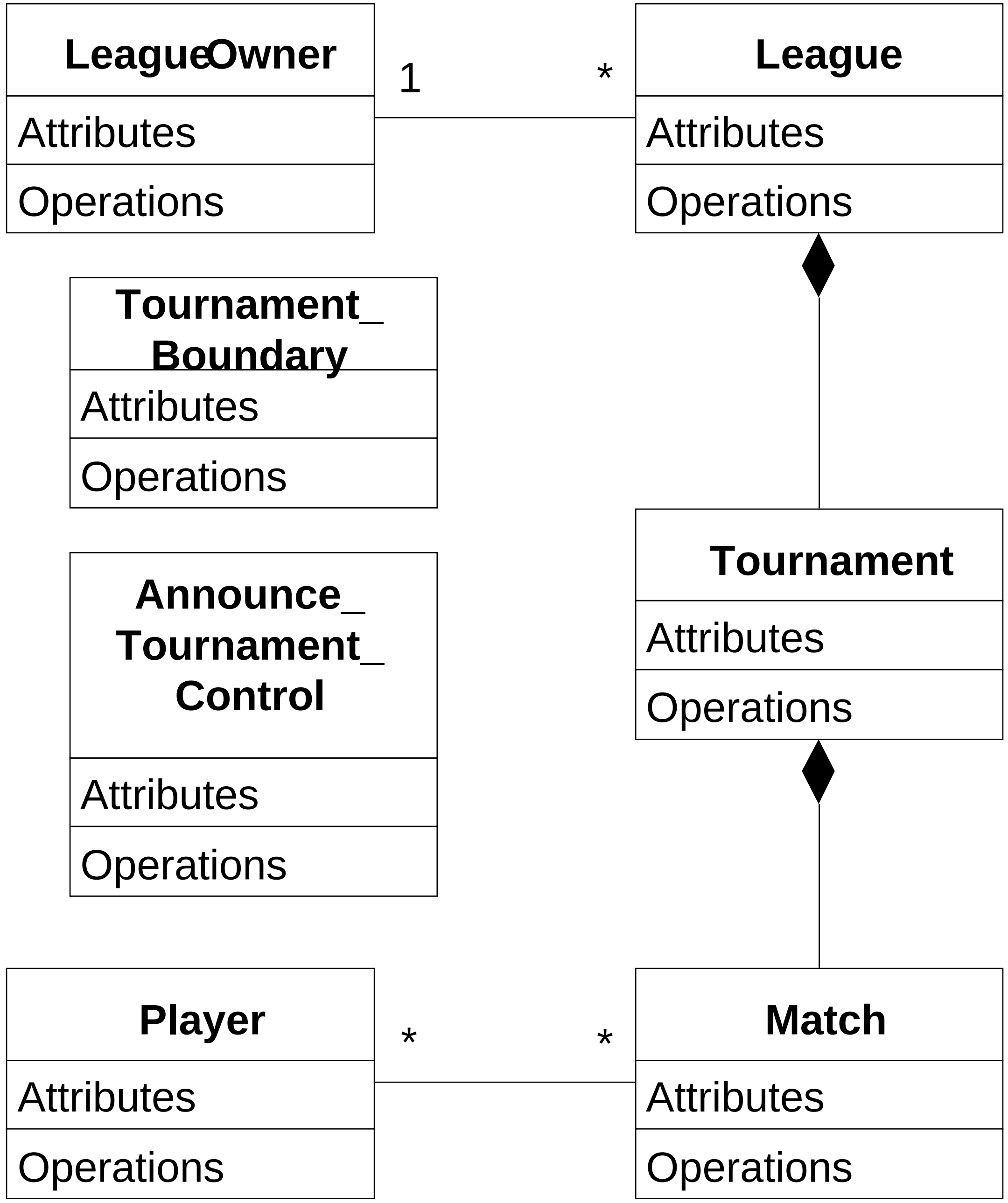
An ARENA Sequence Diagram : Create Tournament



Impact on ARENA's Object Model

- Let's assume, before we formulated the previous sequence diagram, 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**

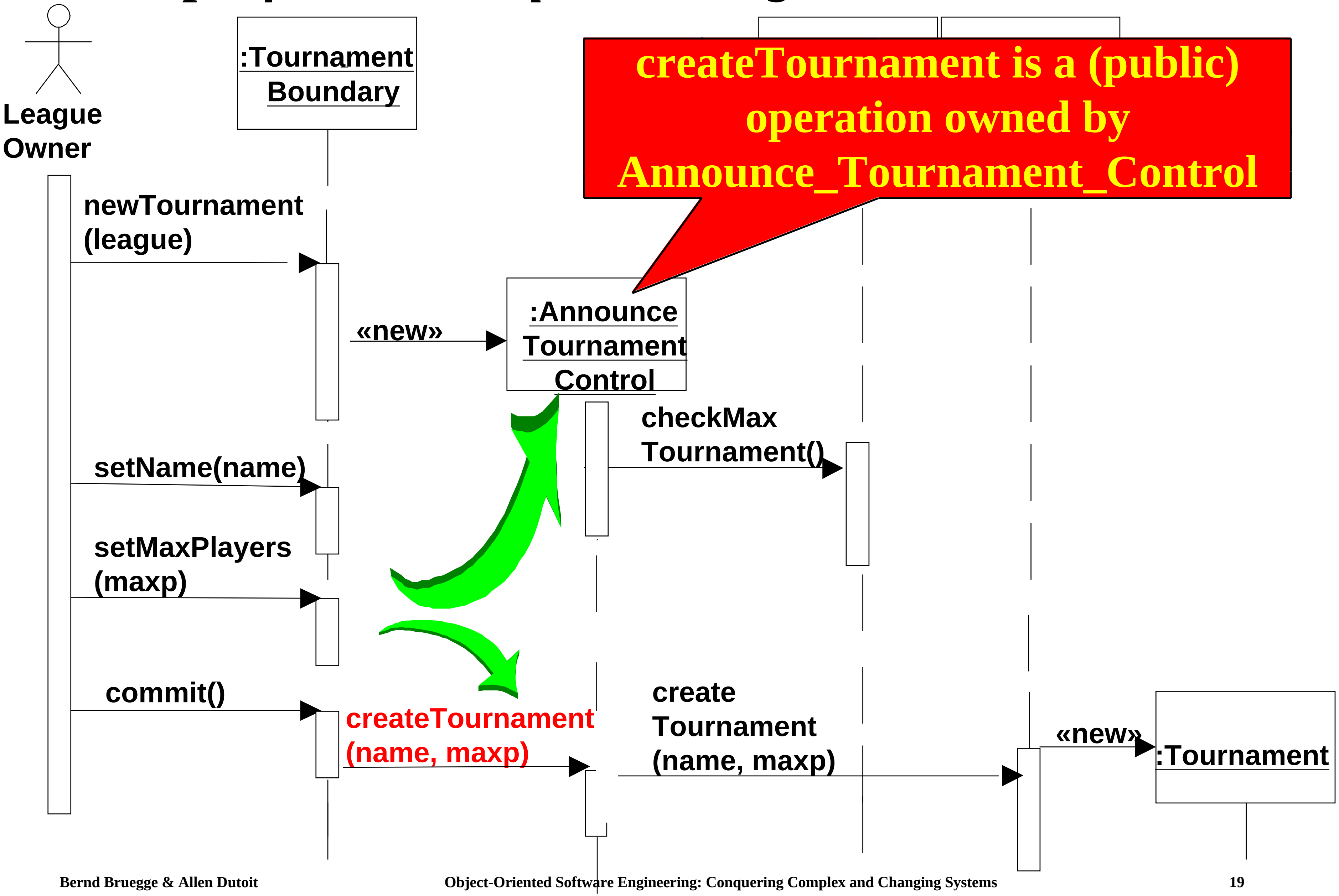

LeagueOwner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

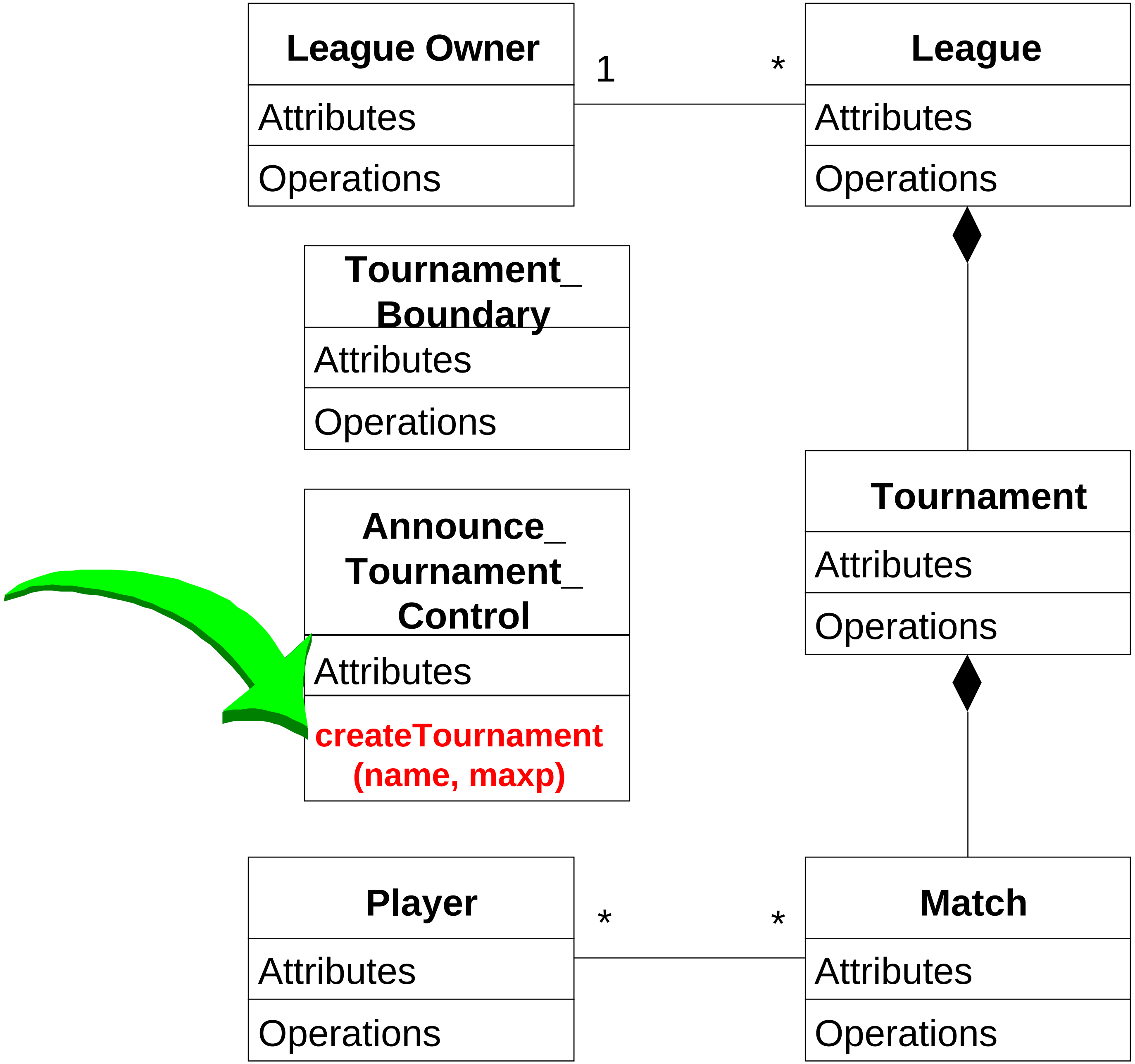

LeagueOwner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament_ Boundary
Attributes
Operations
Announce_ Tournament_ Control
Attributes
Operations
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

Impact on ARENA's Object Model (ctd)

- The Sequence Diagram also supplied us with a lot of new events
  - **newTournament(league)**
  - **setName(name)**
  - **setMaxPlayers(max)**
  - **Commit**
  - **checkMaxTournaments()**
  - **createTournament**
- Question: Who owns these events?
- Answer: For each object that receives an event there is a public operation in the associated class.
  - **The name of the operation is usually the name of the event.**

Example from the Sequence Diagram



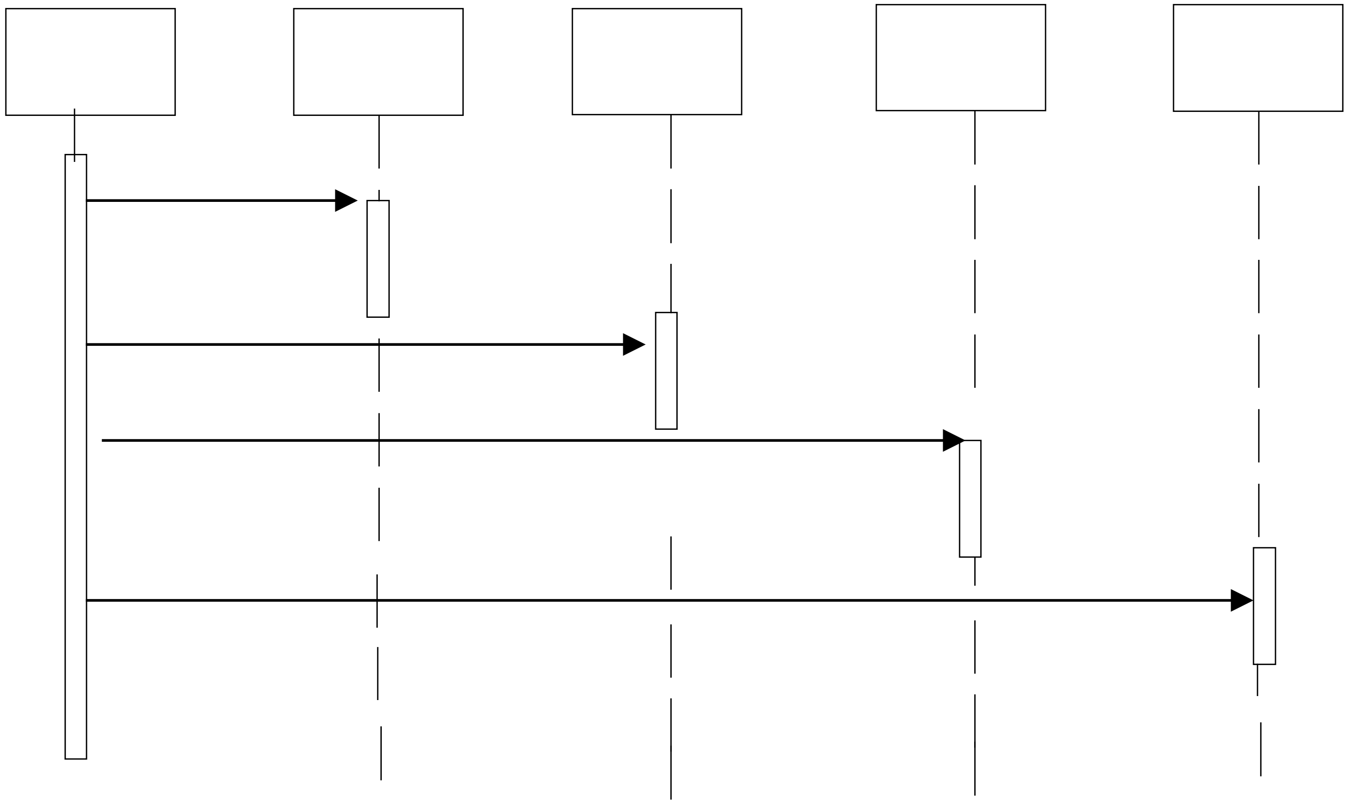

League Owner
Attributes
Operations
1
*
League
Attributes
Operations
Tournament_
Boundary
Attributes
Operations
Announce_
Tournament_
Control
Attributes
createTournament
(name, maxp)
Tournament
Attributes
Operations
Player
Attributes
Operations
*
*
Match
Attributes
Operations

What else can we get out of sequence diagrams?

- Sequence diagrams are derived from the use cases. We therefore see the structure of the 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 and Stair Diagrams (Ivar Jacobsen)

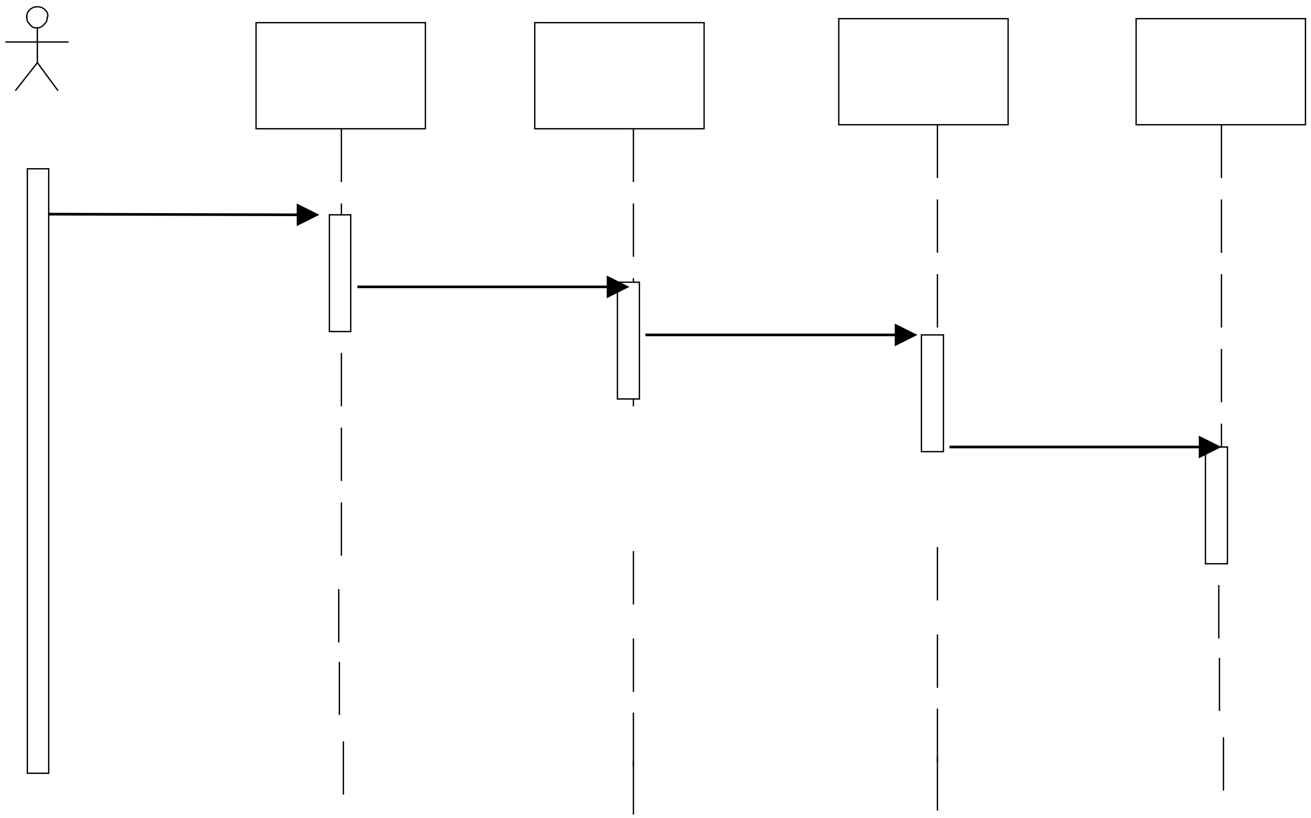
Fork Diagram

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



Stair Diagram

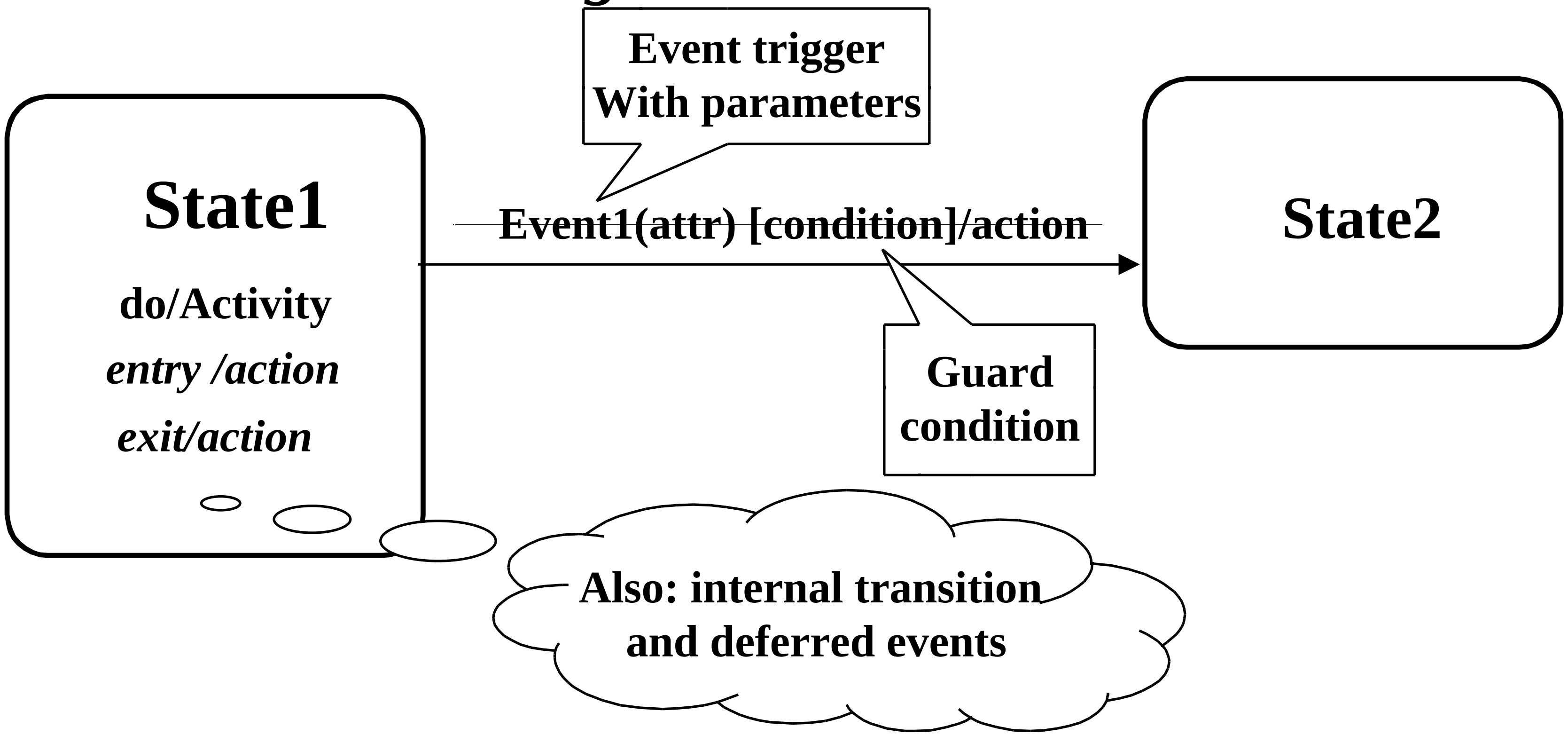
- The dynamic behavior is distributed. Each object delegates some responsibility to other objects. Each object knows only a few of the other objects and knows which objects can hel with a specific behavior.



Fork or Stair?

- Which of these diagram types should be chosen?
- Object-oriented fans claim that the stair structure is better
  - **The more the responsibility is spread out, the better**
- However, this is not always true. Better heuristics:
- Decentralized control structure
  - **The operations have a strong connection**
  - **The operations will always be performed in the same order**
- Centralized control structure (better support of change)
  - **The operations can** *change* **order**
  - **New operations can be inserted as a result of new requirements**

UML Statechart Diagram Notation



- Notation based on work by Harel
  - **Added are a few object-oriented modifications**

- A UML statechart diagram can be mapped into a finite state machine

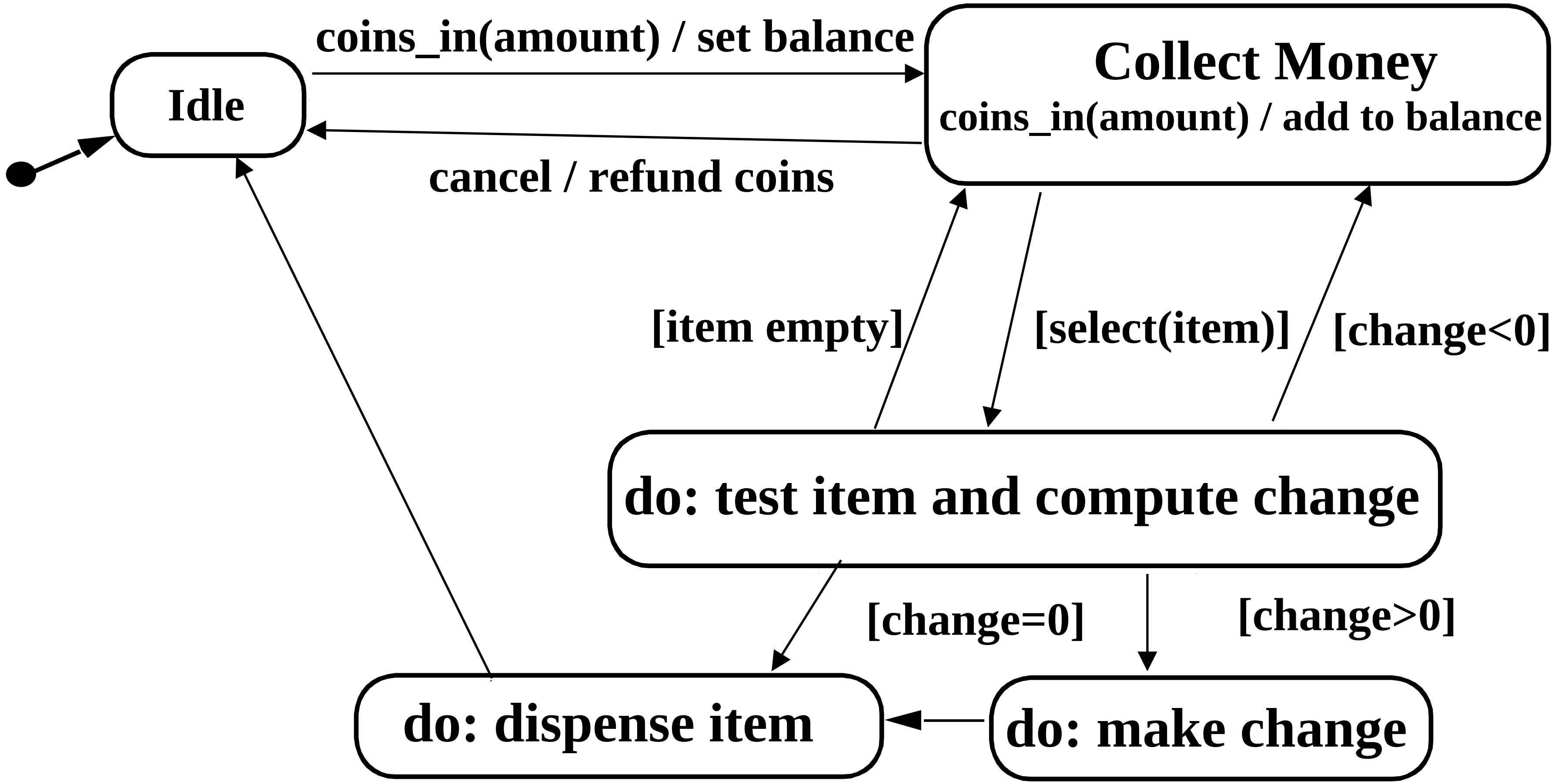
Statechart Diagrams

- Graph whose nodes are states and whose directed arcs are transitions labeled by event names.
- We distinguish between two types of operations in statecharts:
  - **<u>Activity</u>: Operation that takes time to complete**
    - **associated with states**
  - **<u>Action</u>: Instantaneous operation**
    - **associated with events**
    - **associated with states (reduces drawing complexity): Entry, Exit, Internal Action**
- A statechart diagram relates events and states for *one class*
  - **An object model with a <u>set</u> of objects has a <u>set</u> of state diagrams**

State

- An abstraction of the attributes of a class
  - **State is the aggregation of several attributes a class**
- Basically an equivalence class of all those attribute values and links that do no need to be distinguished as far as the control structure of the system is concerned
  - **Example: State of a bank**
    - **A bank is either solvent or insolvent**
- State has duration

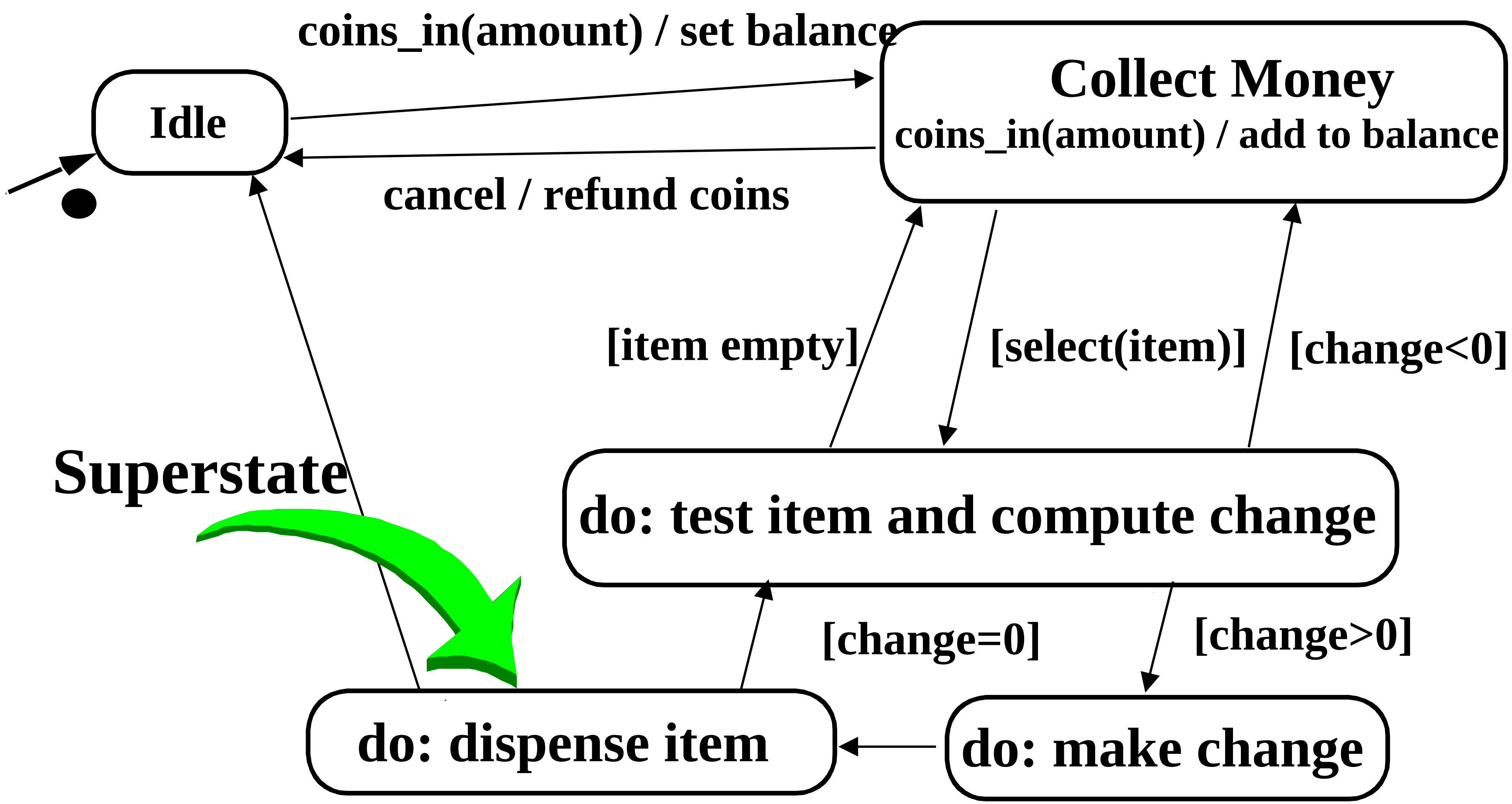
Example of a StateChart Diagram



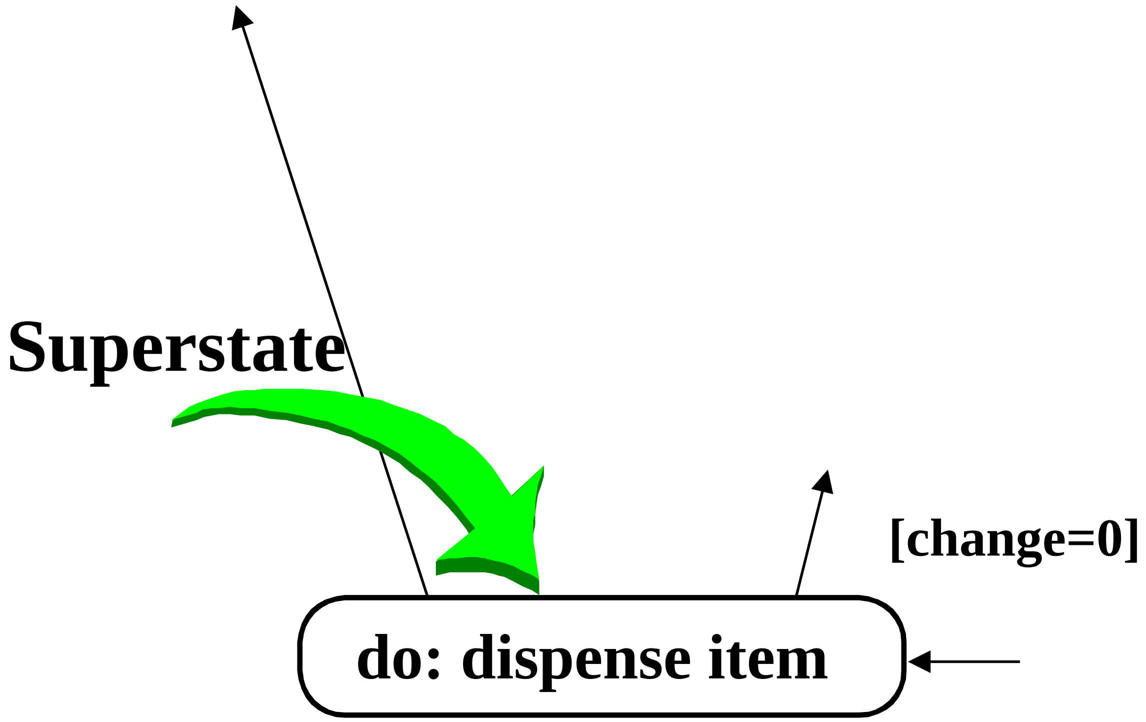
Nested State Diagram

- Activities in states are composite items denoting other lower-level 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.
- Sets of substates in a nested state diagram denote a **superstate** are enclosed by a large rounded box, also called contour.

Example of a Nested Statechart Diagram

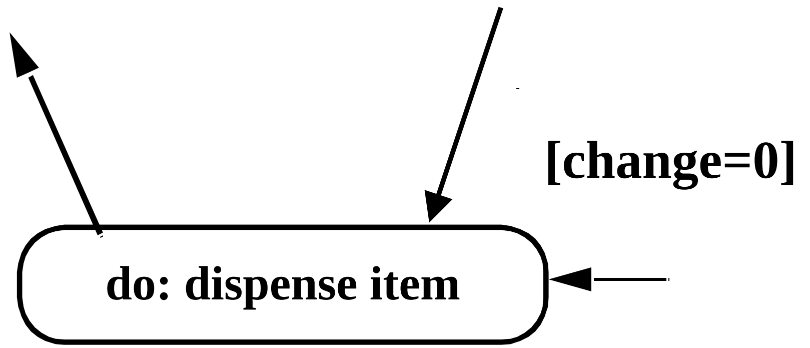


Example of a Nested Statechart Diagram

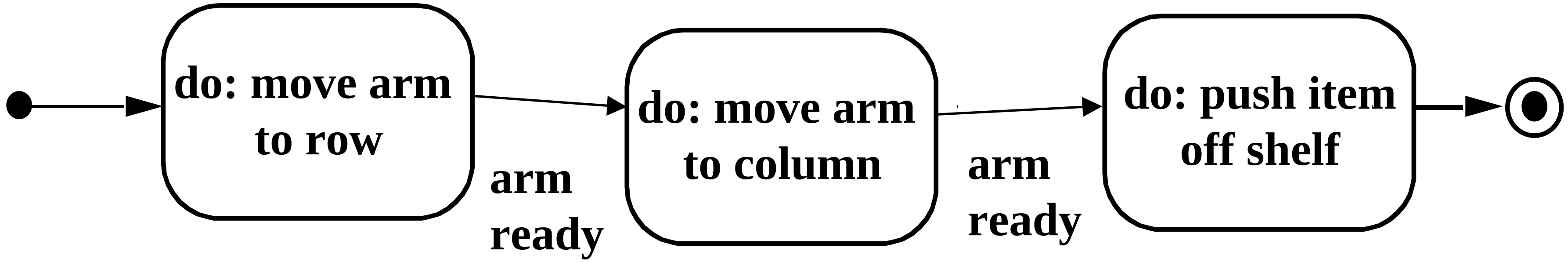


Expanding activity "do:dispense item"

'Dispense item' as an atomic activity:



'Dispense item' as a composite activity:



Superstates

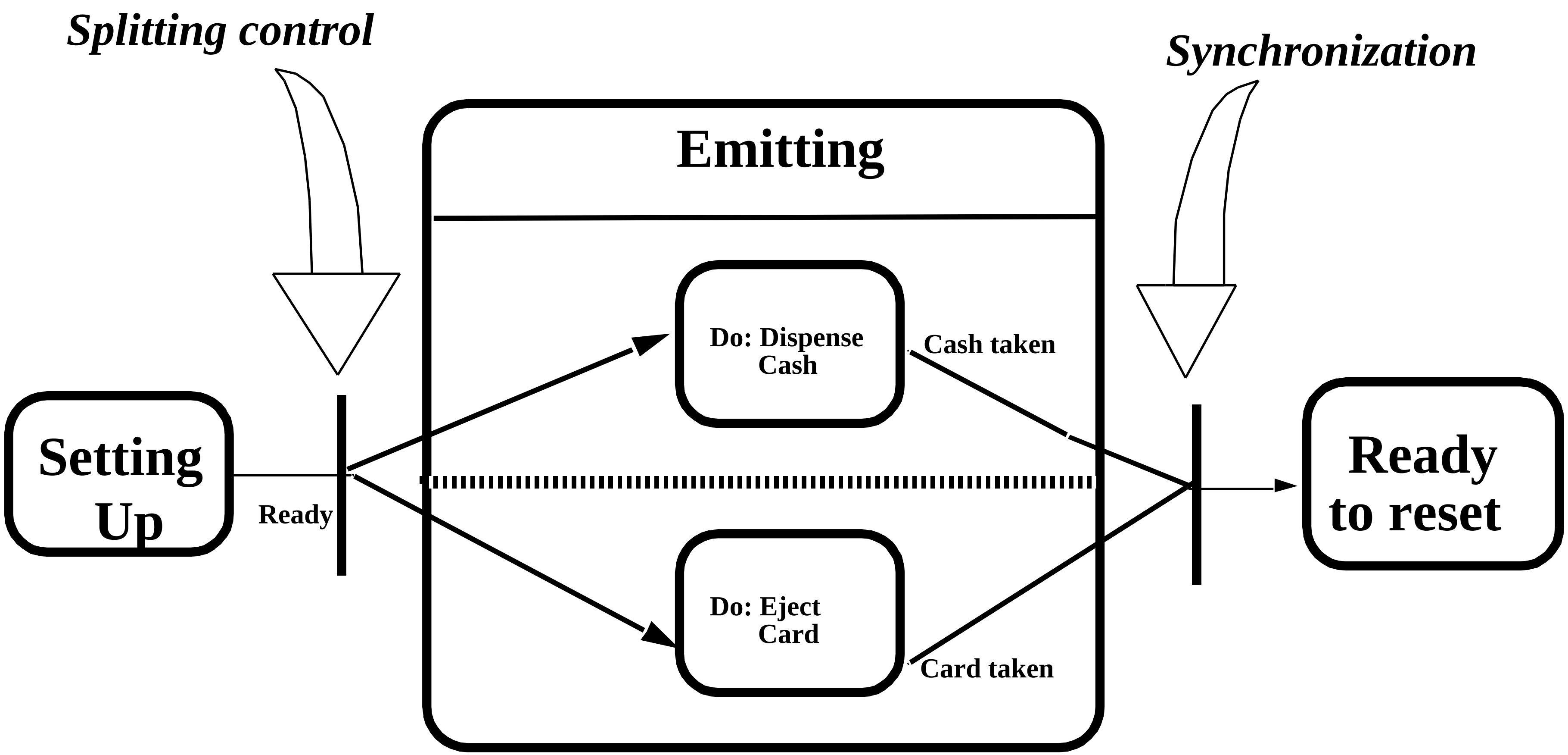
- Goal:
  - **Avoid spaghetti models**
  - **Reduce the number of lines in a state diagram**
- Transitions <u>from</u> other states to the superstate enter the first substate of the superstate.
- Transitions <u>to</u> other states from a superstate are inherited by all the substates (state inheritance)

Modeling Concurrency

Two types of concurrency

1. System concurrency
    - **State of overall system as the aggregation of state diagrams, one for each object. Each state diagram is executing concurrently with the others.**
2. Object concurrency
    - **An object can be partitioned into subsets of states (attributes and links) such that each of them has its own subdiagram.**
    - **The state of the object consists of a set of states: one state from each subdiagram.**
    - **State diagrams are divided into subdiagrams by dotted lines.**

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
  - **Also called Navigation Path**
- States: Name of screens
  - **Graphical layout of the screens associated with the states helps when presenting the dynamic model of a user interface**
- Activities/actions are shown as bullets under screen name
  - **Often only the exit action is shown**
- State transitions: Result of exit action
  - **Button click**
  - **Menu selection**
  - **Cursor movements**
- Good for web-based user interface design

Navigation Path Example

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

List of events

• User selects event(s)

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

List of sensor events

• User selects sensor event(s)

Graph

• User selects data group and type of graph

Selection

• User selects data group
    • Field site
    • Car
    • Sensor group
    • Time range
• User selects type of graph
    • time line
    • histogram
    • pie chart

Visualize

• User views graph
• User can add data groups for being viewed

Link

• User makes a link (doclink)

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 state 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, do thought experiments**
- 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

1. Analyze the problem statement
   - **Identify functional requirements**
   - **Identify nonfunctional requirements**
   - **Identify constraints (pseudo requirements)**
2. Build the functional model:
   - **Develop use cases to illustrate functionality requirements**
3. Build the dynamic model:
   - **Develop sequence diagrams to illustrate the interaction between objects**
   - **Develop state diagrams for objects with interesting behavior**
4. Build the object model:
   - **Develop class diagrams showing 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: 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?

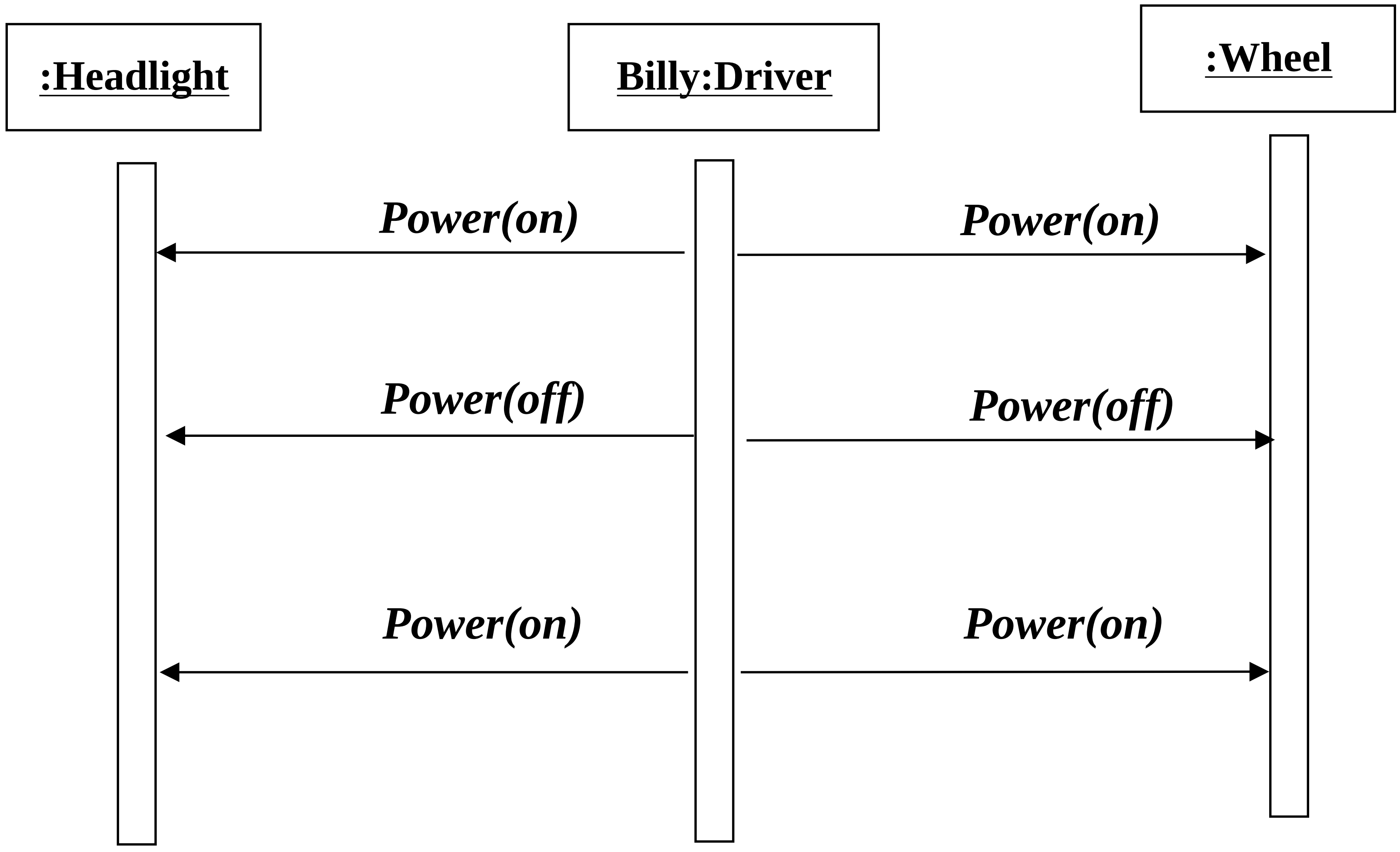
- Use case 1: System Initialization
  - **Entry condition: Power is off, car is not moving**
  - **Flow of events:**
    - **Driver turns power on**
  - **Exit condition: Car moves forward, headlight is on**

- 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.**

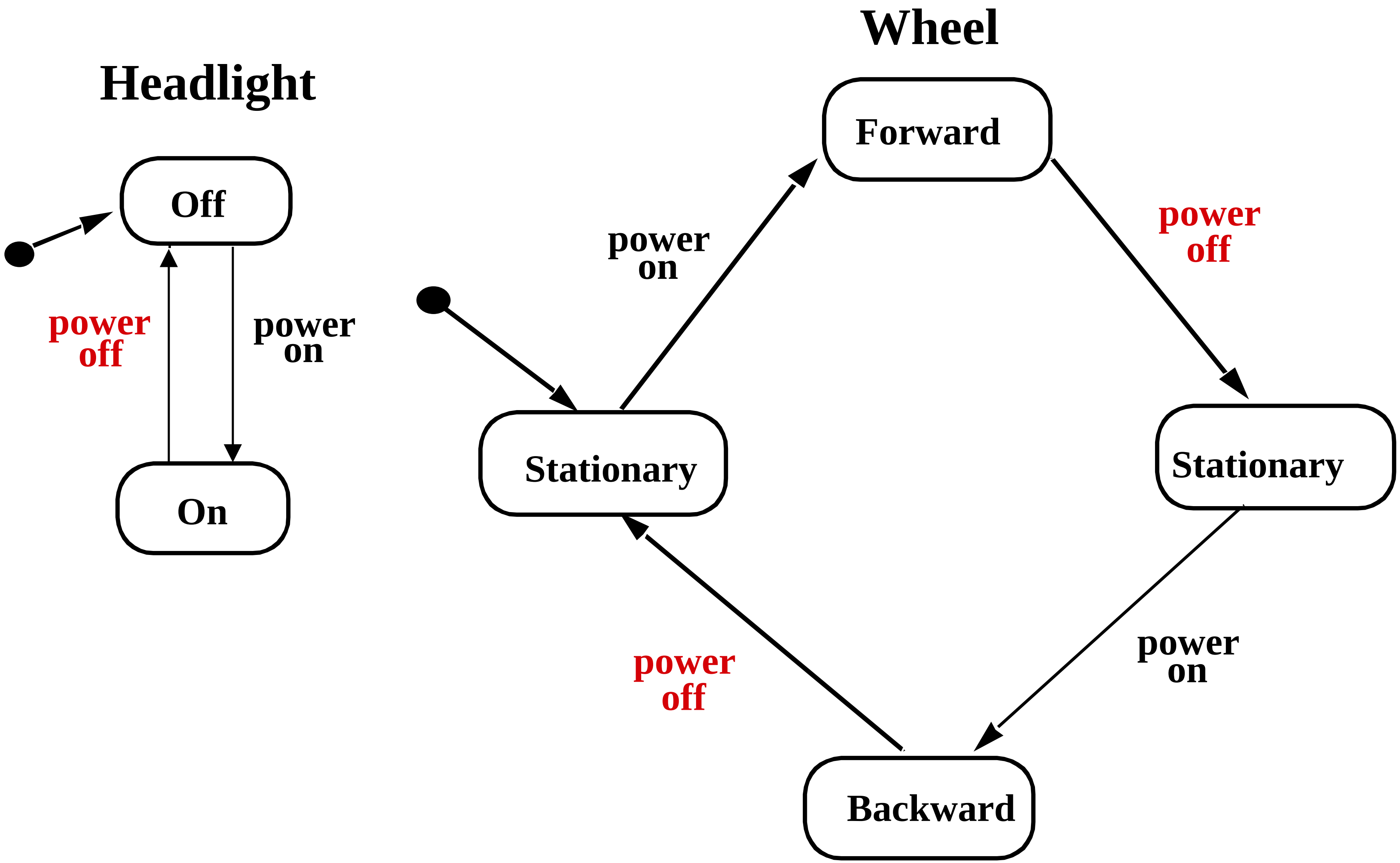
Find the Dynamic Model: Create sequence diagram

- 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**
  - . . .

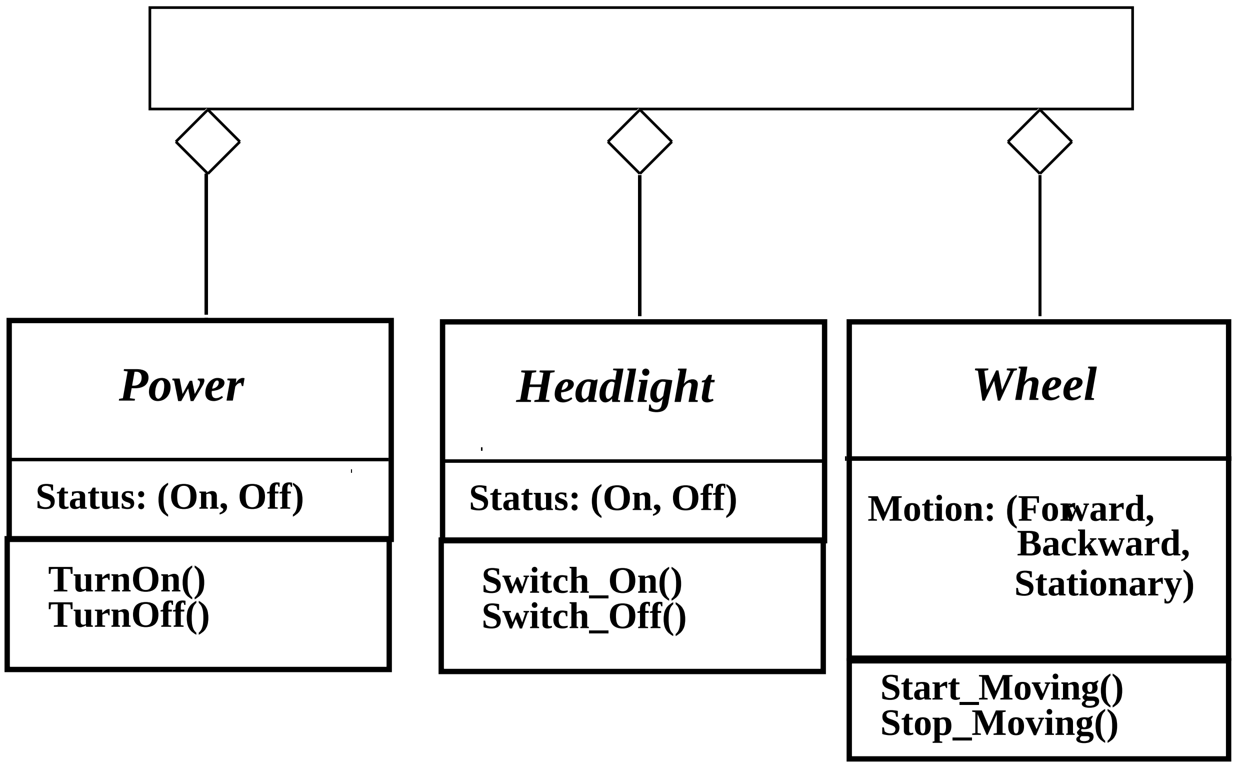
Sequence Diagram for Drive Car Scenario



Toy Car: Dynamic Model



Toy Car: Object Model



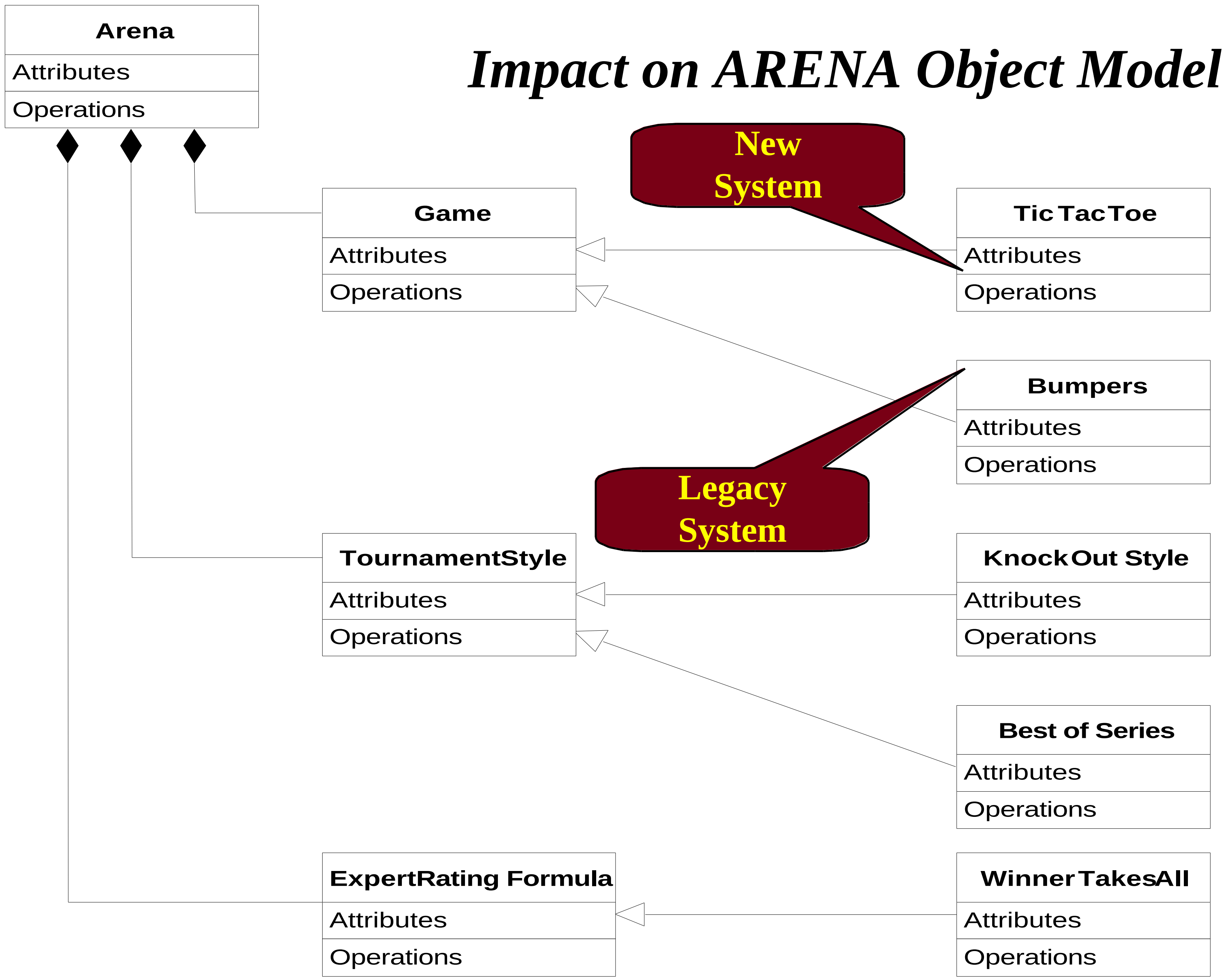
A more realistic example: ARENA11/22/01

- ARENA Problem Statement available on SE Discuss bboard
  - **http://tum.globalse.org/teaching/ws01/SE/discuss.nsf/f7069403ac0dc2178525662700754e21/1f1d9871dd61c5f3c1256b0a003c484c?OpenDocument**
- We will be developing the RAD for ARENA concurrently to the class.
- You can participate by using the REQuest tool to write one or more use cases corresponding to this problem statement.
- Request an account from Allen Dutoit and then log into the ARENA system of REQuest:
  - **http://atbruegge13.in.tum.de:8080/arena/servlet/SYSLogin**

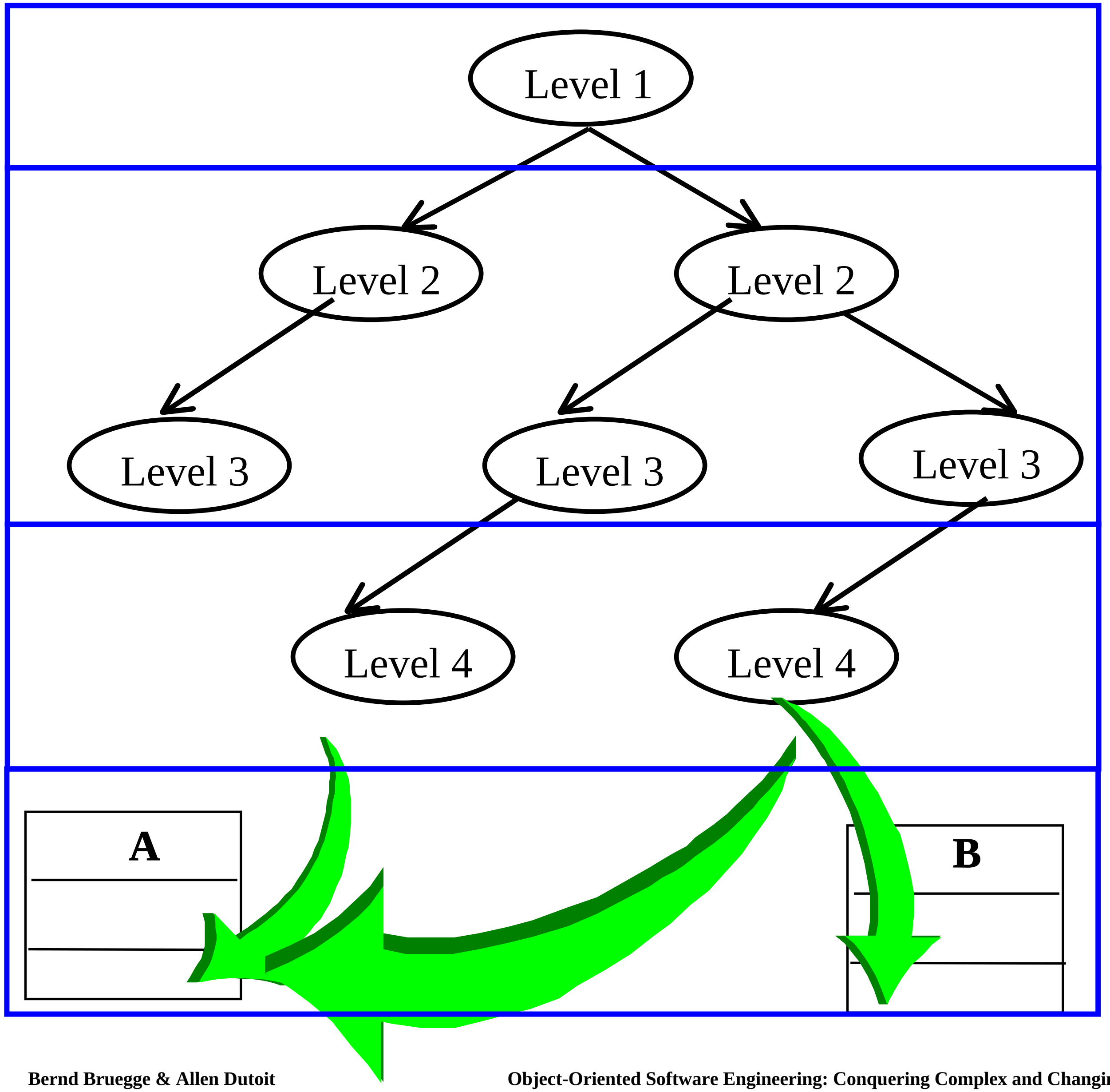
Additional constraints in ARENA Project

- Interface Engineering
  - **Provide ARENA players with access to an existing game: Bumpers**
  - **Complete Java Code for Bumpers posted on SE Discuss**
- Greenfield Engineering
  - **Design a new game and provide ARENA players with access to the new game**
- Constraints:
  - **Extensibility**
  - **Scalability**
- Additional Constraint:
  - **The existing ARENA code does not have to be recompiled when the new game is introduced**
  - **ARENA does not have to be shut down (currently running games can continue) when the new game is introduced**
- Is the “NotShutDown” requirement realistic?

Impact on ARENA Object Model



Clarification: Terminology in REQuest



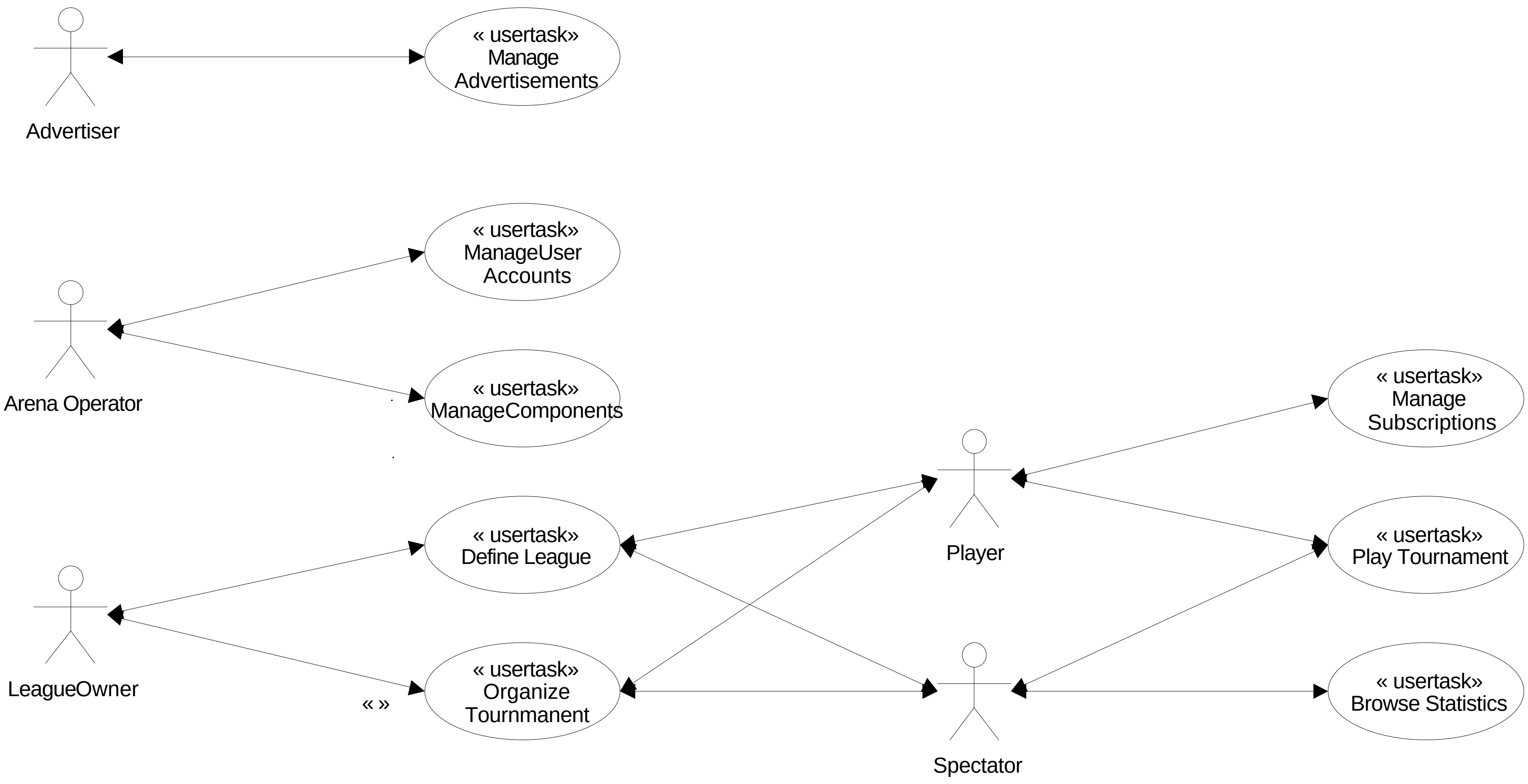
User tasks
describe application domain

Use Cases
describe interactions

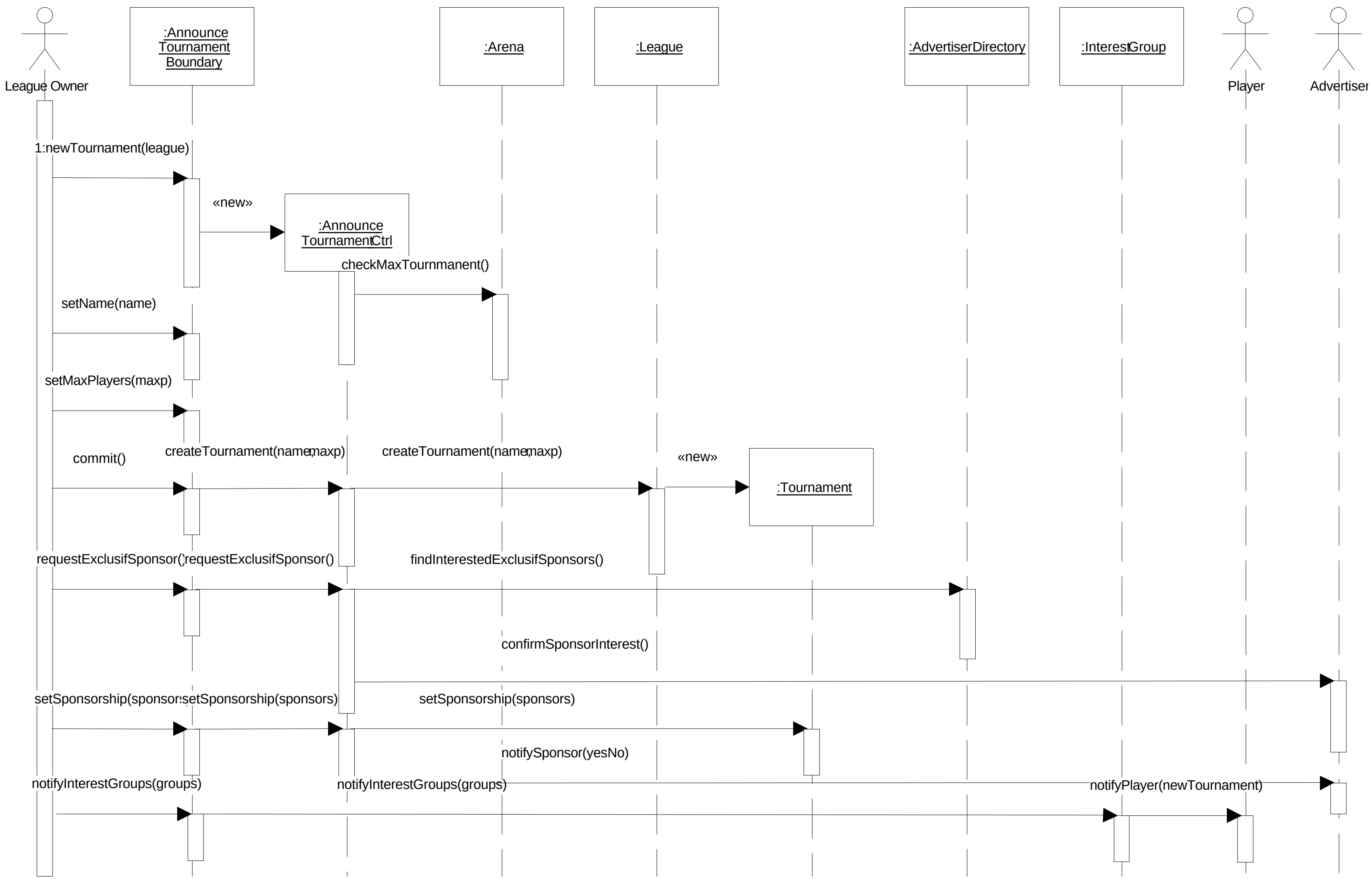
Services
describe system

Participating Objects
describe domain entitie

ARENA user tasks (top level use cases)



AnnounceTournament (Part of OrganizeTournament)



When is a model dominant?

- ***Object model*: The system has objects with nontrivial state.**
- ***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 of these models is dominant in the following three cases?
  - **Compiler**
  - **Database system**
  - **Spreadsheet program**

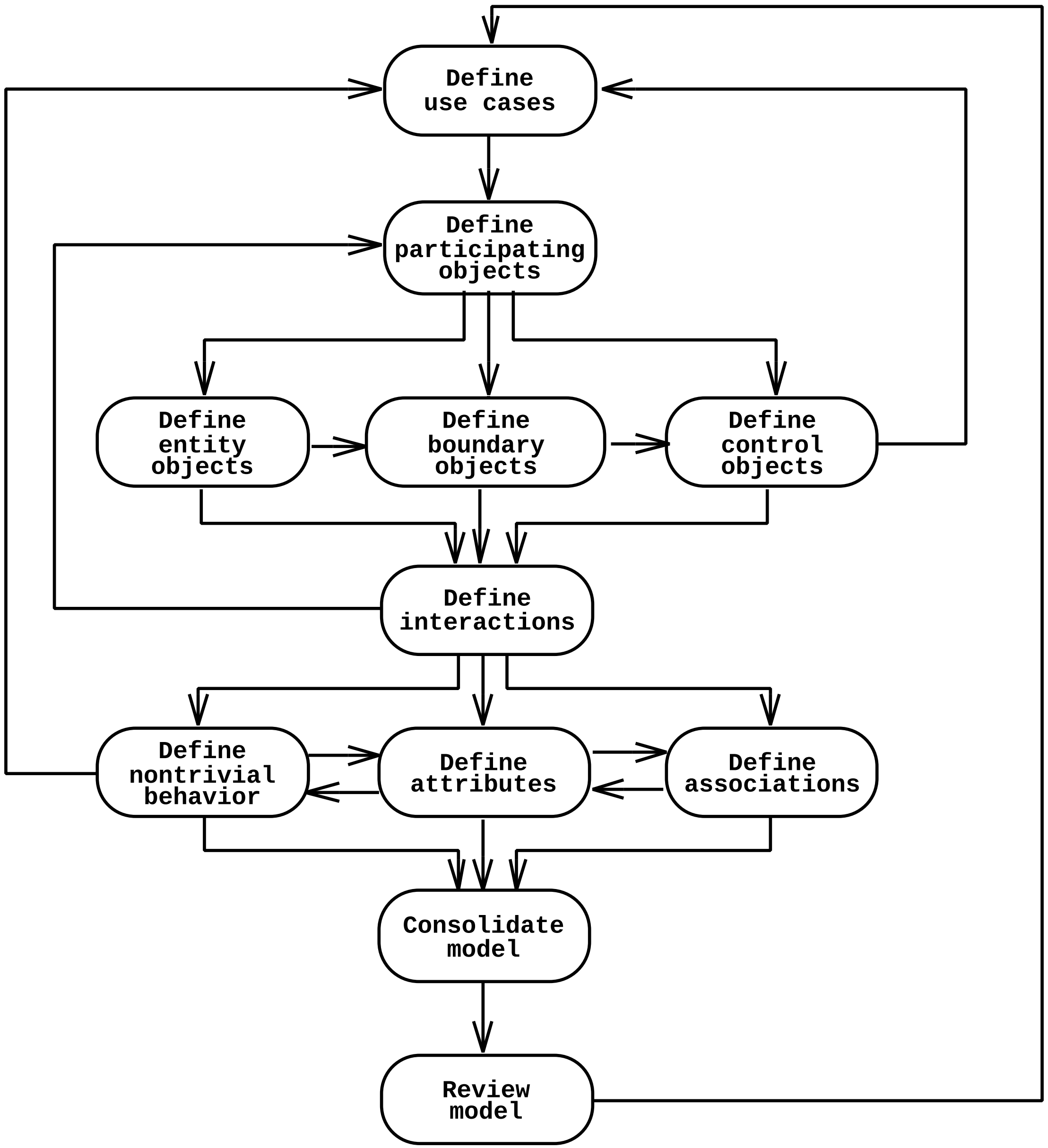
Dominance of models

- *Compiler:*
  - **The functional model most important. (Why?)**
  - **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 usually 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, because the spreadsheet values are trivial and cannot be structured further. The only interesting object is the cell.**

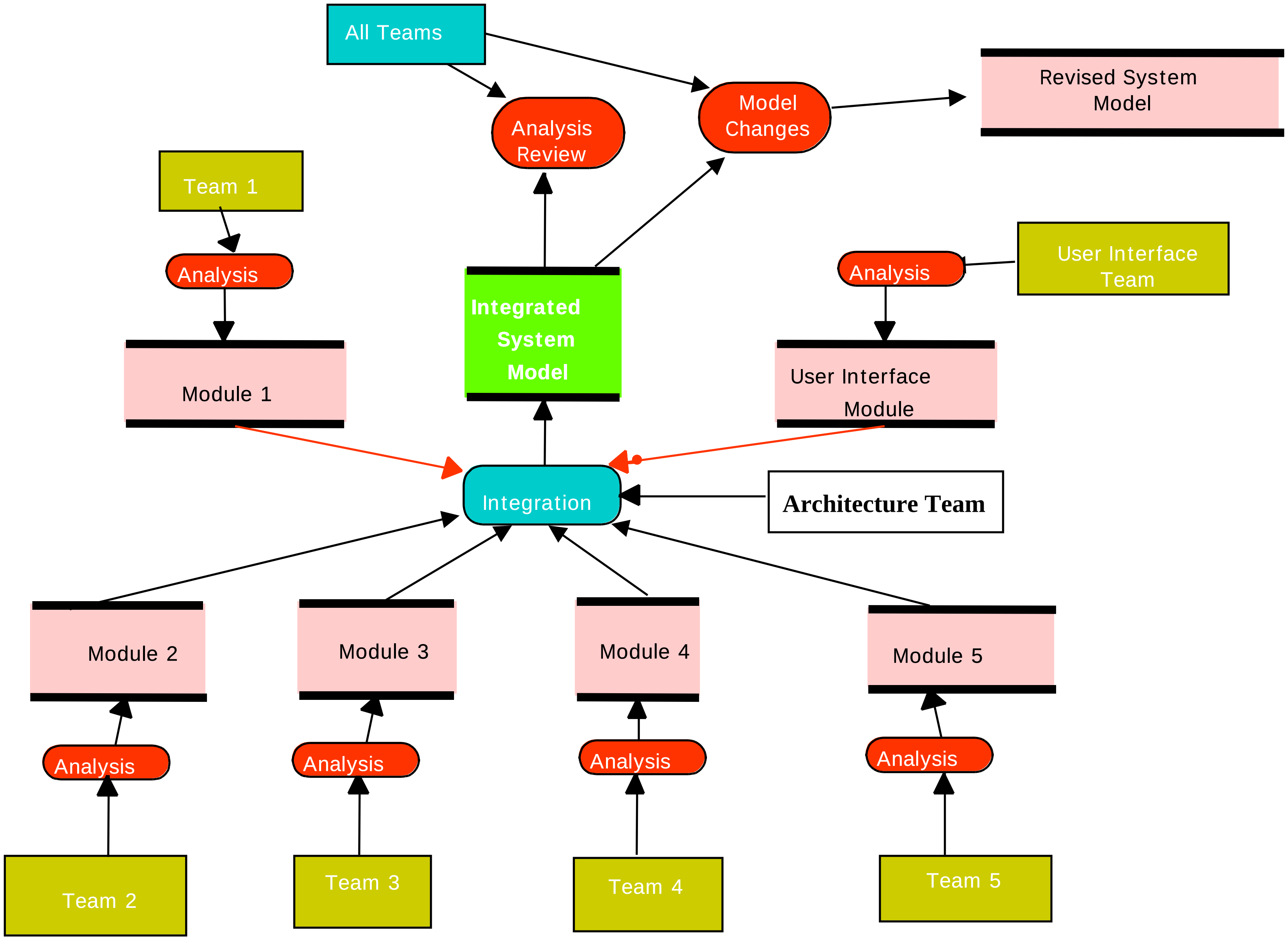
Collaborative Analysis (into SS 2002)

- A system is a collection of subsystems providing services
- Analysis of services is provided by a set of the teams who provide the models for their subsystems
- Integration of subsystem models into the full system model by the architecture team
- Analysis integration checklist:
  - Are all the classes mentioned in the data dictionary?
  - Are the names of the methods consistent with the names of actions, activities, events or processes?
  - Check for assumptions made by each of the services
    - Missing methods, classes
    - Unmatched associations

Analysis: UML Activity Diagram (into SS 2002)



Object Model Integration in a large Project (into SS 2002)



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**

Section 3.3 Nonfunctional Requirements

3.3.1 User interface and human factors

3.3.2 Documentation

3.3.3 Hardware considerations

3.3.4 Performance characteristics

3.3.5 Error handling and extreme conditions

3.3.6 System interfacing

3.3.7 Quality issues

3.3.8 System modifications

3.3.9 Physical environment

3.3.10 Security issues

3.3.11 Resources and management issues

Nonfunctional Requirements: Trigger Questions

3.3.1 User interface and human factors

- **What type of user will be using the system?**
- **Will more than one type of user be using the system?**
- **What sort of training will be required for each type of user?**
- **Is it particularly important that the system be easy to learn?**
- **Is it particularly important that users be protected from making errors?**
- **What sort of input/output devices for the human interface are available, and what are their characteristics?**

3.3.2 Documentation

- **What kind of documentation is required?**
- **What audience is to be addressed by each document?**

3.3.3 Hardware considerations

- **What hardware is the proposed system to be used on?**
- **What are the characteristics of the target hardware, including memory size and auxiliary storage space?**

Nonfunctional Requirements, ctd

3.3.4 Performance characteristics

- **Are there any speed, throughput, or response time constraints on the system?**
- **Are there size or capacity constraints on the data to be processed by the system?**

3.3.5 Error handling and extreme conditions

- **How should the system respond to input errors?**
- **How should the system respond to extreme conditions?**

3.3.6 System interfacing

- **Is input coming from systems outside the proposed system?**
- **Is output going to systems outside the proposed system?**
- **Are there restrictions on the format or medium that must be used for input or output?**

Nonfunctional Requirements, ctd

- 3.3.7 Quality issues
  - **What are the requirements for reliability?**
  - **Must the system trap faults?**
  - **What is the maximum time for restarting the system after a failure?**
  - **What is the acceptable system downtime per 24-hour period?**
  - **Is it important that the system be portable (able to move to different hardware or operating system environments)?**
- 3.3.8 System Modifications
  - **What parts of the system are likely candidates for later modification?**
  - **What sorts of modifications are expected?**
- 3.3.9 Physical Environment
  - **Where will the target equipment operate?**
  - **Will the target equipment be in one or several locations?**
  - **Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, ...)?**

Nonfunctional Requirements, ctd

- 3.3.10 Security Issues
  - **Must access to any data or the system itself be controlled?**
  - **Is physical security an issue?**
- 3.3.11 Resources and Management Issues
  - **How often will the system be backed up?**
  - **Who will be responsible for the back up?**
  - **Who is responsible for system installation?**
  - **Who will be responsible for system maintenance?**

Constraints (Pseudo Requirements)

- Constraint:
  - **Any client restriction on the solution domain**
- Examples:
  - **The target platform must be an IBM/360**
  - **The implementation language must be COBOL**
  - **The documentation standard X must be used**
  - **A dataglove must be used**
  - **ActiveX must be used**
  - **The system must interface to a papertape reader**

Outline of the Lecture

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

Verification and Validation of models

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

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

Correctness, Completeness and Consistency

- Verification is an equivalence check between the transformation of two models:
  - **We have two models, is the transformation between them correct?**
- Validation is different. We don't have two models, we need to compare one model with reality
  - **"Reality" can also be an artificial system, like an legacy system**
- 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 the following checks
  - **Correctness**
  - **Completeness**
  - **Ambiguity**
  - **Realistism**

Modeling Checklist for the Review

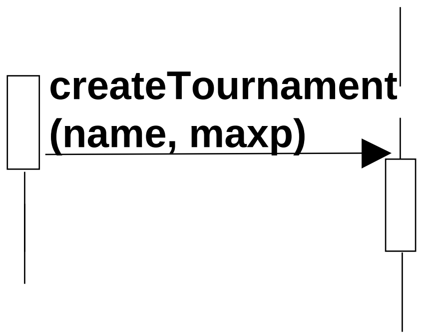
- Is the model correct?
  - **A model is correct if it represents the client's view of the the system: Everything is the model represents an aspect of reality**
- Is the model complete?
  - **Every scenario through the system, including exceptions, is described.**
- Is the model consistent?
  - **The model does not have components that contradict themselves (for example, deliver contradicting results)**
- Is the model unambigous?
  - **The model describes one system (one reality), not many**
- Is the model realistic?
  - **The model can be implemented without problems**

Diagram Checklist for the RAD

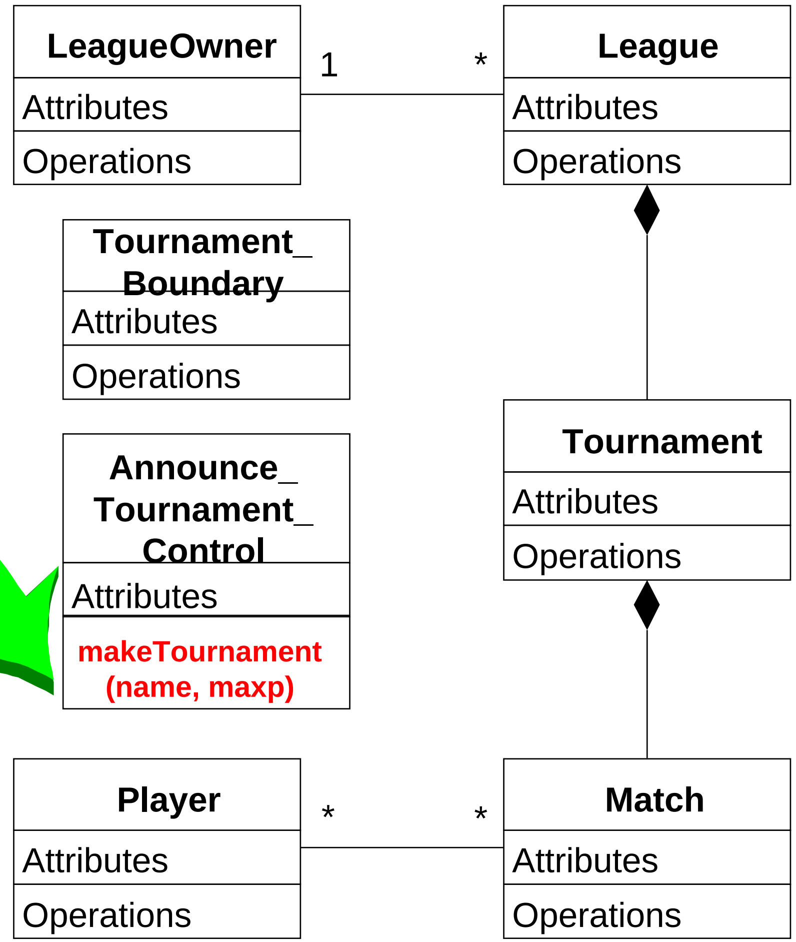
- One problem with modeling:
  - **We describe a system model with many different views (class diagram, use cases, sequence diagrams,)state charts)**
- We need to check the equivalence of these views as well
- Syntactical check of the models
  - **Check for consistent naming of classes, attributes, methods in different subsystems**
  - **Identify dangling associations (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**
- Don’t rely on CASE tools for these checks
  - **Many of the existing tools don’t do all these checks for you.**
- Examples for syntactical problems with UML diagrams

Different spellings in different diagrams

**(from)
UML Sequence Diagram**



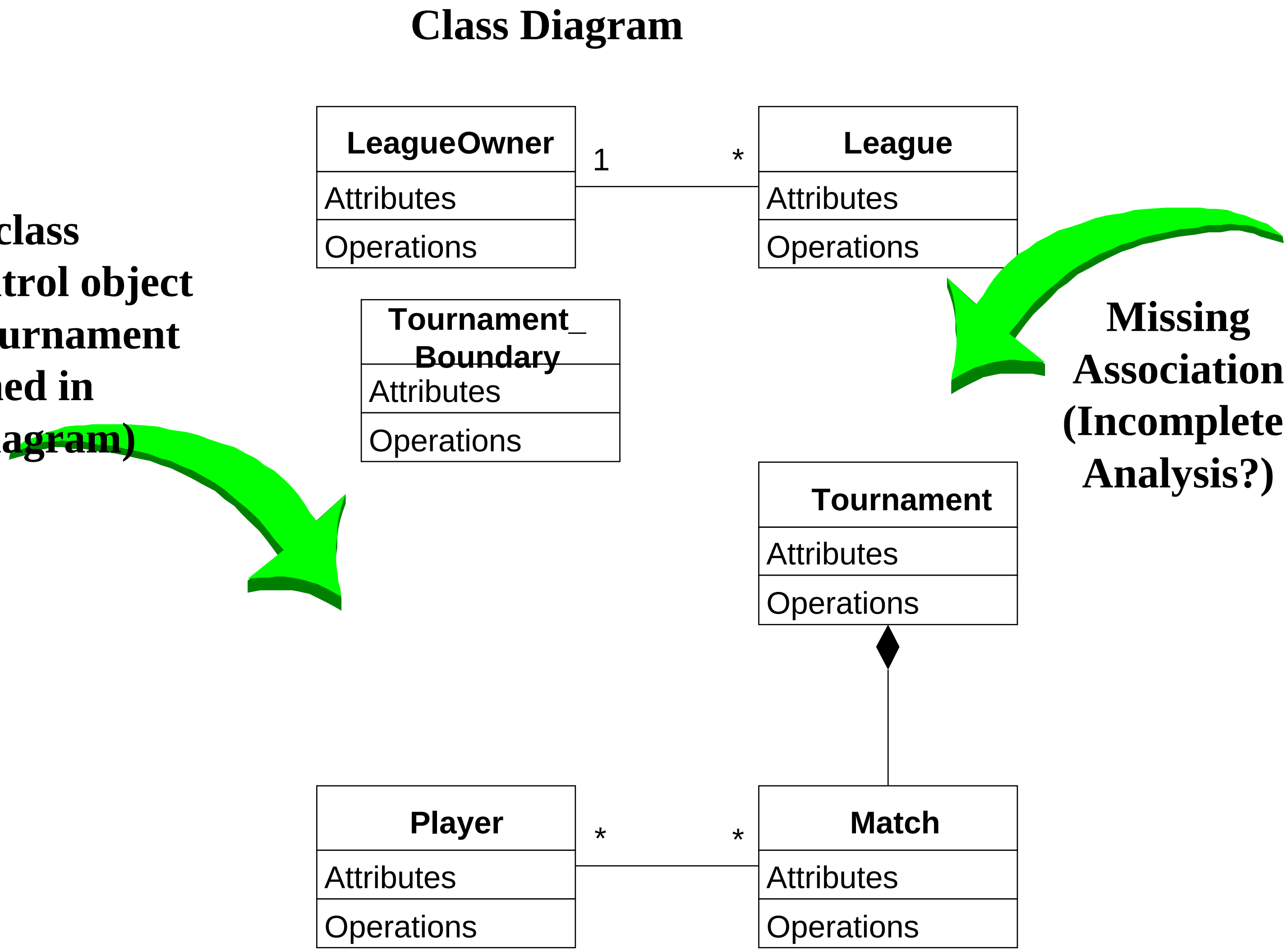
UML Class Diagram



**Different spellings
In different models
for the same operation?**

Omissions in some diagrams

Missing class
(associated control object
Announce_Tournament
is mentioned in
Sequence diagram)



Missing
Association
(Incomplete
Analysis?)

Project Agreement

- The project agreement represents the acceptance of (parts of) the analysis model (as documented by the requirements analysis document) by the client.
- The client and the developers converge on a single idea and agree about the functions and features that the system will have. In addition, they agree on:
  - **a list of prioritized requirements**
  - **a revision process**
  - **a list of criteria that will be used to accept or reject the system**
  - **a schedule, and a budget**
- Translates to “Lastenheft”

Prioritizing requirements

- High priority ("Core requirements")
  - **Must be addressed during *analysis, design, and implementation.***
  - **A high-priority feature must be demonstrated successfully during client acceptance.**
- Medium priority ("Optional requirements")
  - **Must be addressed during *analysis and design*.**
  - **Usually implemented and demonstrated in the second iteration of the system development.**
- Low priority ("Fancy requirements")
  - **Must be addressed during *analysis* ("very visionary scenarios").**
  - **Illustrates how the system is going to be used in the future if not yet available technology enablers are available**

Requirements Analysis Document: REQuest vs book

The Requirements Analysis Document described in the book is slightly different from the structure used by the REQuest tool. However, this difference is structural only, their content is similar.

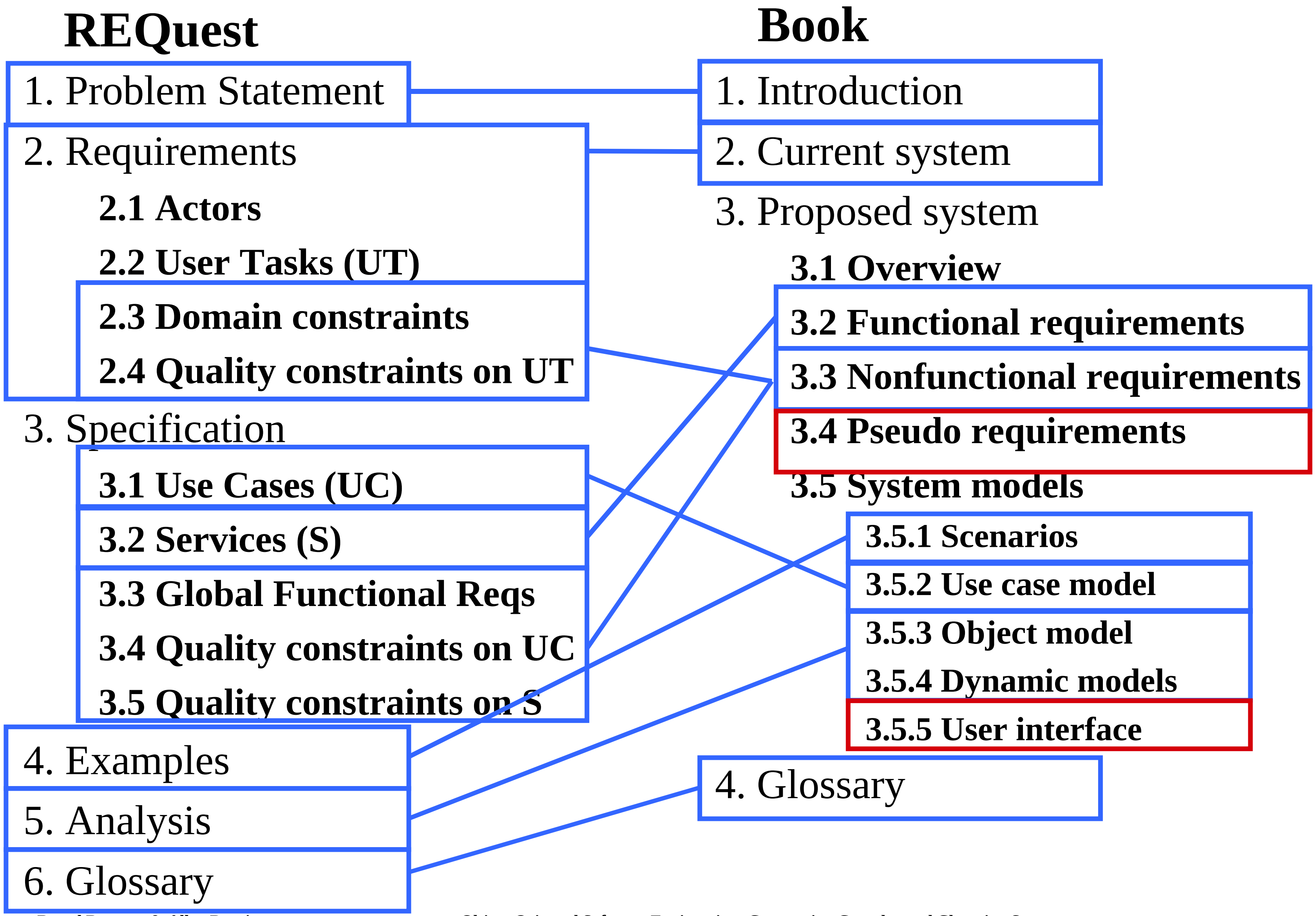
Book RAD

- General (You will see RADs like this in industry)
- Follows IEEE Specification standard (adapted to UML).
- Follows the order in which the elements are needed by management and developers.
- Optimizes readability.

REQuest RAD

- Method & tool specific (this structure is not used anywhere else)
- Follows the order in which elements are usually created.
- Optimizes tool usability.

Structure of the Requirements Analysis Document: Book vs REQuest



Summary

In this lecture, we reviewed the construction of the dynamic model from use case and object models. In particular, we described: 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