15-413

Dynamic Modeling

Bernd Bruegge
Carnegie Mellon University
School of Computer Science

15 September 1998

Outline of Class

- Odds and Ends
- Entity, Interface and Control Objects
- Dynamic Modeling
  - **Sequence Diagrams: Interaction between objects**
  - **State Diagrams: Behavior of a single object**
  - **Navigation Paths: Dynamic model of user interface**
- Requirements Analysis Example
- Requirements Analysis Template

Odds and Ends

- Event Teams and Learning Teams are merged

- Quizzes

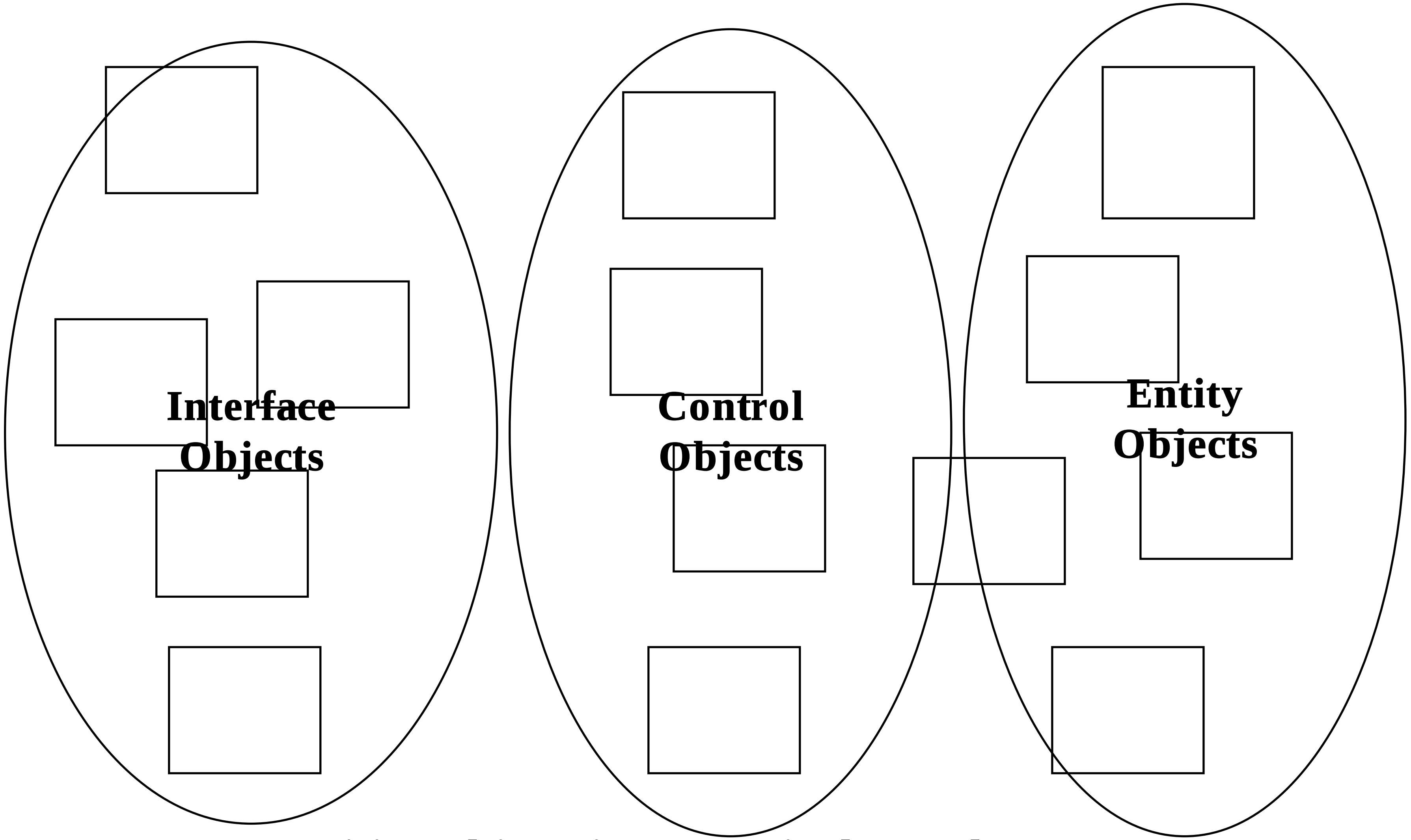
Suggestion:

No quiz as long as attendance is high
We keep an attendance sheet
We will reintroduce quizzes if attendance becomes low

Recap: How do you find classes?

- Scenarios
  - **Natural language formulation of a concrete usage of the system**
- Use Cases
  - **Natural language formulation of the functions of the system**
- Application domain: Domain entities lead to objects
- General world knowledge and intuition
- Textual analysis of problem statement (Abbot)
- Dynamic model (This lecture)
  - **Sequence diagrams as sources for objects**
  - **State Charts:**
    - **Events: Candiates for operations to be offered by classes**
- Design Patterns (Coming soon…)

I found all my use cases and participating objects. Now what?



1. **Partition objects into 3 equivalence classes...**
2. **Do dynamic modeling....**

Different Object Types

- Interface objects
  - **Implement the interaction with the user**
  - **Constructed from UI components (User Interface Management Systems and Toolkits)**
  - ***Subject to most modification***
- Entity objects
  - **Represent the application domain**
  - **Often represent persistent data (=> database management system)**
  - ***Least often modified***
- Control objects
  - **Coordinate between interface and entity objects**
  - **Constructed with Command objects**
  - ***Modified frequently but less often than interface objects***

Identifying Interface Objects

- In each use case, each actor interacts at least through one interface object.
- The interface object collects the information from the actor and translates it into an interface neutral form that can be used by the control and entity objects.
- Naming:
  - **<name>_Interface**
  - **<name>_Menu**
  - **<name>_Button**
  - **….**

Identifying Control Objects

- There is often a close relationship between a use case and a control object.
- A control object is usually created at the beginning of a use case and ceases to exist at its end.
- It is responsible for collecting information from the interface objects and dispatching it to entity objects.
  - **For example, control objects describe the behavior associated with the sequencing of forms, undo and history queues, and dispatching information in a distributed system**
- Naming
  - **<name>_Control**

Some Heuristics for Identification of Interface and Control Objects

- ❖ Identify one control object per use case or more if the use case is complex and can be divided into shorter flows of events,
- ❖ Identify one control object per actor in the use case,
- ❖ The life span of a control object should be the extent of the use case or the extent of a user session.
- ❖ If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case may not have a well defined entry and exit condition.

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 single objects => Statechart Diagram**

UML Notations for Dynamic Models

- 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**
- Statechart Diagram:
  - **A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).**
- Activity Diagram:
  - **Special type of statechart where all states are action states**

Sequence Diagrams

- A sequence diagram includes time but does not include object relationships.
- Sequence diagrams are used to describe use cases (i.e., all possible interactions betweem participating objects) and scenarios (i.e., one possible interaction)
- In other words: A sequence diagram is useful to model a use case or scenario with its participating objects. It also leads to the detection of new participating objects.
- Sequence Diagrams are basically DAGs (direct acyclic graphs)
  - **Time increases from top to bottom**
  - **Spacing is irrelevant. Objects are shown as vertical rectangles**
  - **Events are shown as horizontal arrows from the sending to the receiving object**

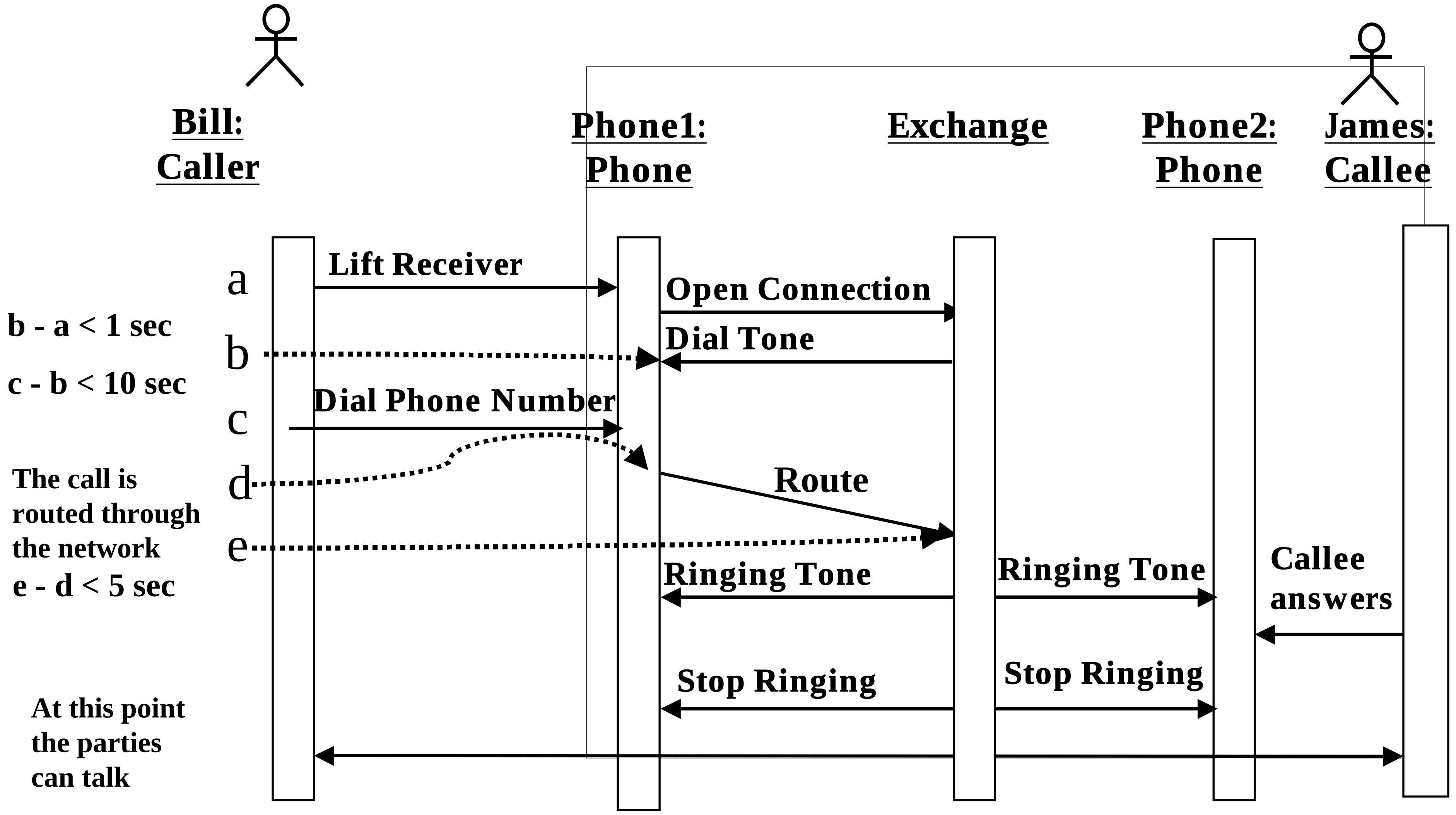
Flow of Events: What is an Event?

- Event: Something that happens at a point in time
- Relation of events to each other:
  - **Causally related: Before, after**
  - **Causally unrelated: concurrent**
- Events can be grouped in event classes (event hierarchy).
  - **Event class "Sound", Subclass "Phone Ring", Subclass "Thunder"**
- The term 'Event' is often used in two ways:
  - **As an instance of an event class: "Phone rings at 12:25 in WeH 4123".**
  - **As an attribute of an event class**
    - **Begin of ringing (12:25 PM)**
    - **Mouse button down(button#, tablet-location)**
- Events have a sender and a receiver. Find them for each event: These are the participating objects

Start with Event Flow in Use case

- ***Name of Use Case:***
  - **Initiate Phone Call**
- ***Actors:***
  - **Bill**
  - **Callee**
- ***Entry condition:***
  - **The phone receiver is on the hook**
- ***Flow of events:***
  - **The Caller lifts receiver**
  - **The Dial tone begins**
  - **The Caller dials**
  - **The Phone rings**
  - **The Callee answers phone**
- ***Exit Condition:***
  - **Caller and Callee are connected**
- ***Exceptions:***
  - **The Callee does not anser the phone**
- ***Special Requirements:***
  - **Time between lifting the receiver and getting the dial tone must be less than 1 sec**
  - **User has 10 secs to start dialing after the dial tone appears**
  - **Routing takes not longer than 5 secs**

Sequence Diagram for “Initiate Phone Call”



Drawing Sequence Diagrams

- Each column represents an object that is participating in the interaction.
- The vertical axis represents time (from top to bottom). Messages are shown by full arrows.
- Labels on full arrows represent message names and arguments.
- Activations (i.e., the time it takes to perform an operation) are depicted by a rectangle attached to an object. The height of the rectangle is indicative for the duration of the operation
  - **The vertical rectangle shows that an object is active, that is, it is handling a request made by another object.**
  - **The operation can itself send other requests to other objects**
  - **An object can request an operation from itself (looping arrow)**

Style Guide for Sequence Diagrams

- Column 1:
  - **Models the actor or object who initiates the use case**
- Column 2:
  - **Should be an interface object (that the actor used to initiate the use case)**
- Column 3:
  - **Should be the control object that manages the rest of the use case**

Some Heuristics for Good Sequence Diagrams

- Control objects are created by interface objects initiating use cases
- Entity objects are accessed by control and interface objects
- Entity objects never access interface or control objects. This makes it easier to share them across use cases.

Using Sequence Diagrams for Design Patterns

- In the text book “Design Patterns” *sequence diagrams* are called *interaction diagrams*..
- A solid vertical line indicates the lifetime of a particular object
- If the object does not get instantiated until after the beginning of time as recorded in the diagram, then its vertical line is dashed until the point of creation, where it becomes solid.

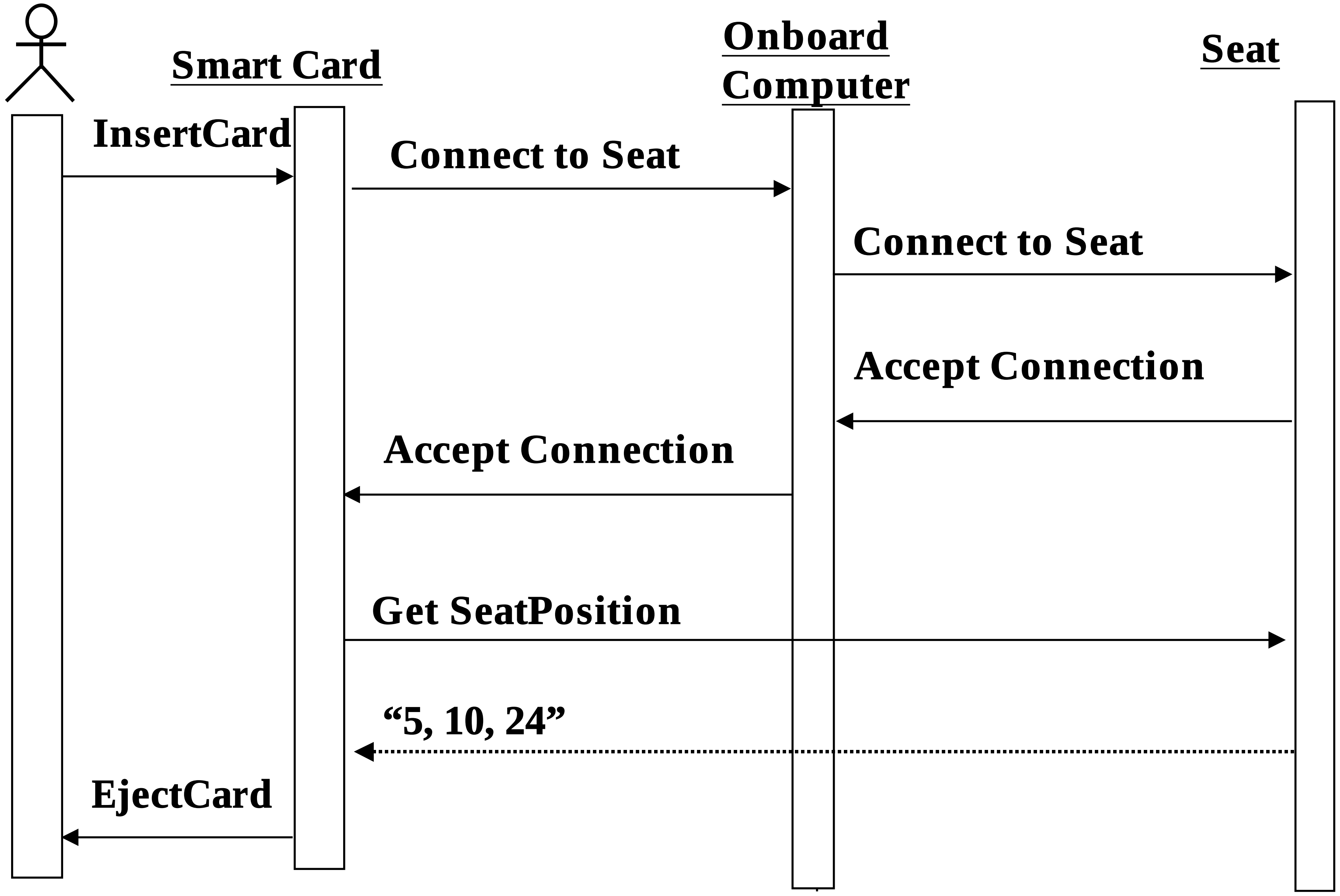
Another Example

- Flow of events in a “Get SeatPosition” use case :
  1. **Driver inserts smart card**
  2. **Connection between smart card and onboard comuter is established**
  3. **Connection between onboard computer and seat is established**
  4. **Current seat position is obtained and stored on smart card**
  5. **Driver ejects card**

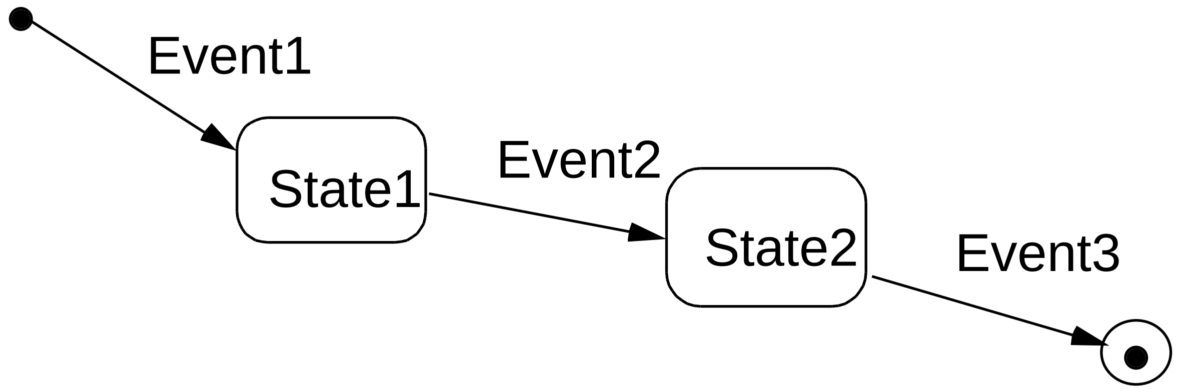
- What are the participating objects?

  Smart Card, Onboard Computer, Seat

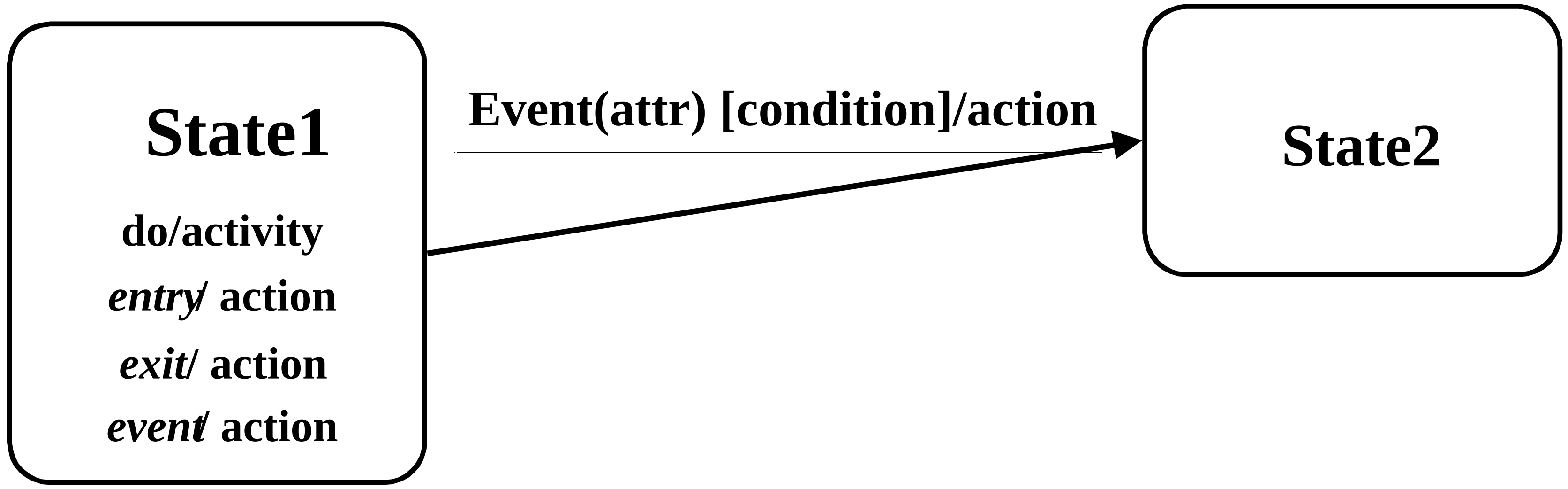
Sequence Diagram for "Get SeatPosition"



State Chart Diagrams: Relating states and events



UML Statechart Diagram Notation



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

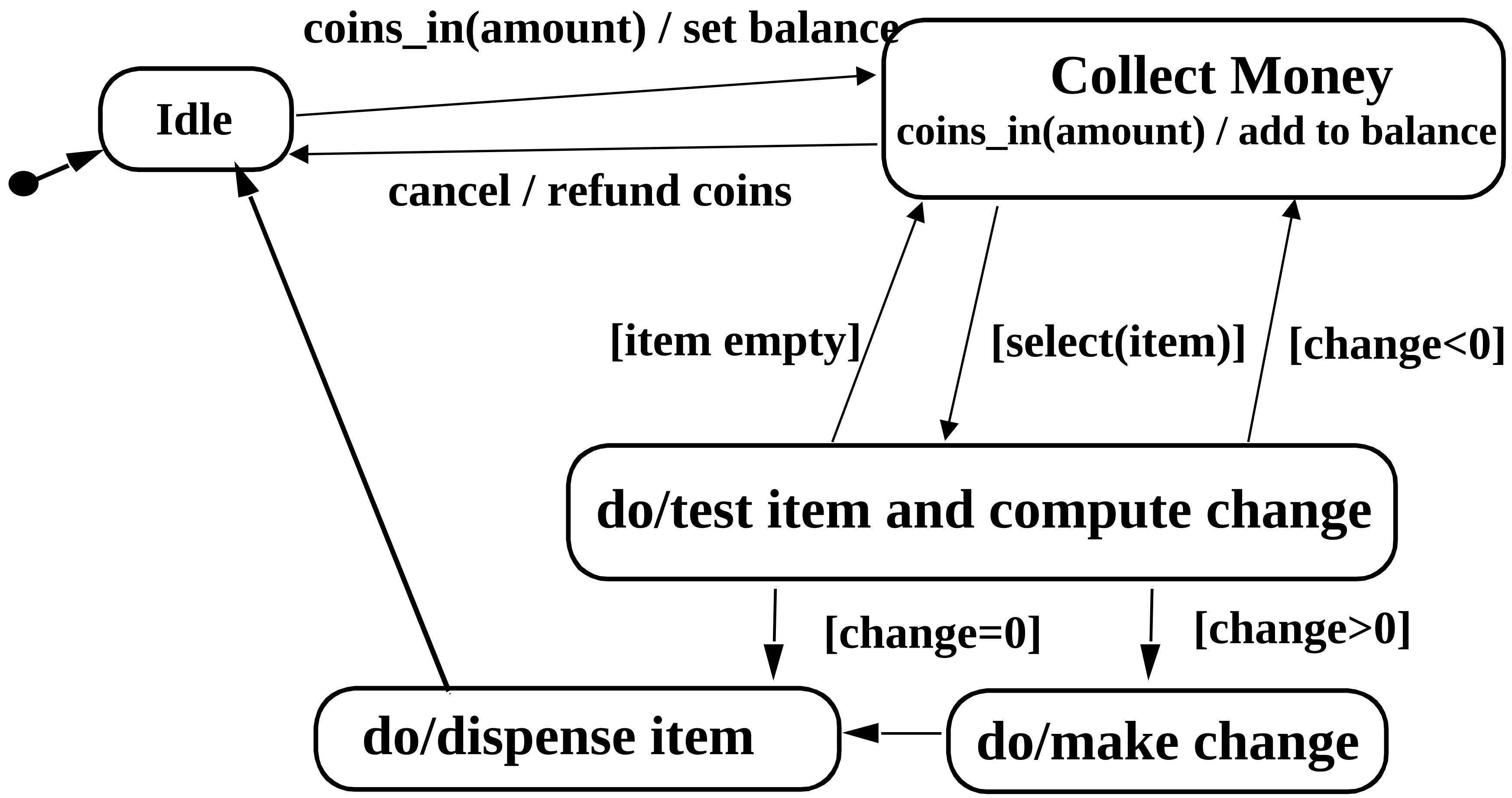
Statechart Diagrams

- Graph whose nodes are states and whose directed arcs are transitions labeled by event names.
- Distinguish between two types of operations:
  - ***Activity:*** **Operation that takes time to complete**
    - **associated with states**
  - ***Action:*** **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*

State

- An abstraction of the attribute 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

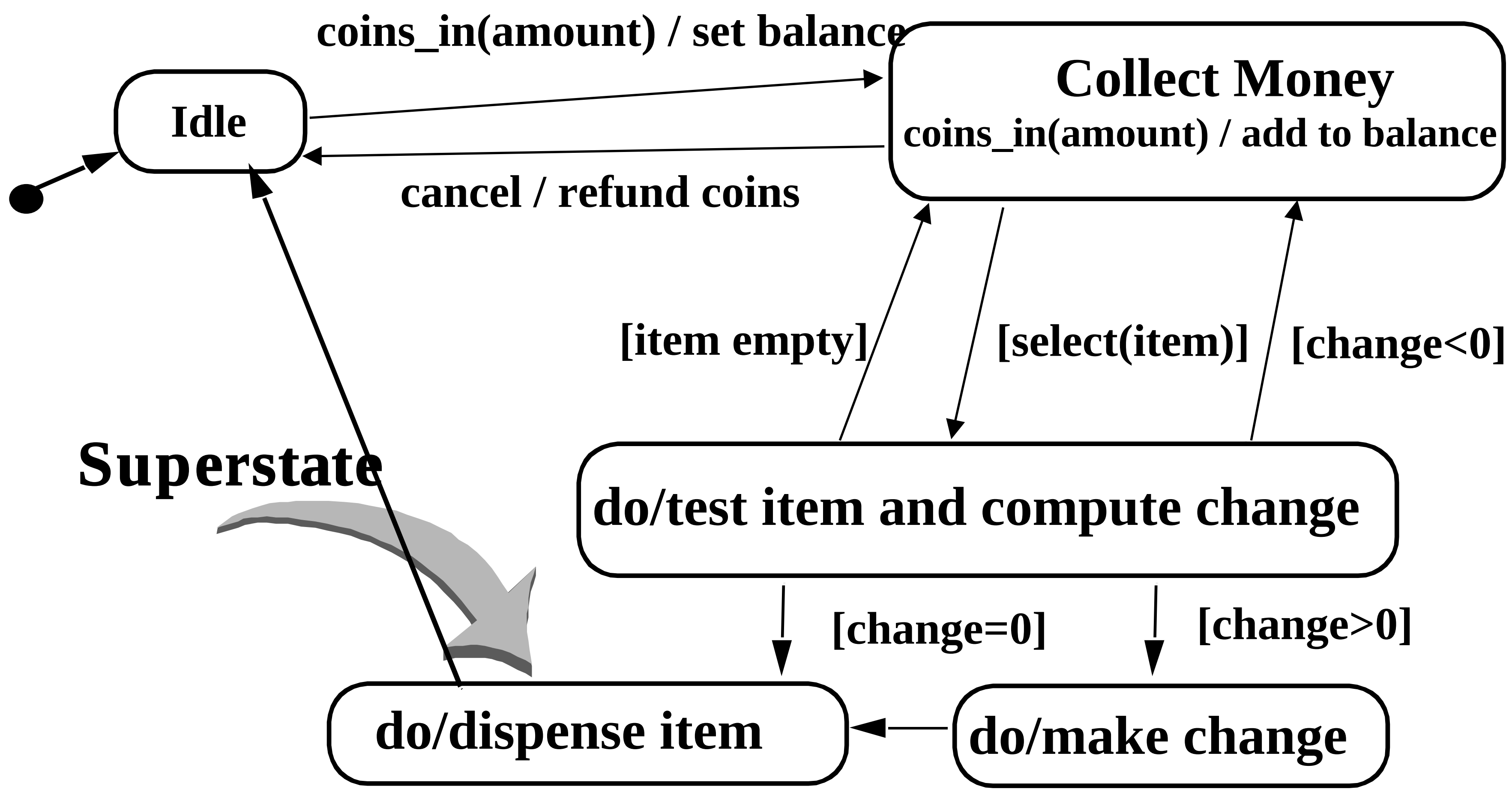
Example of a StateChart Diagram



Nested State Diagram

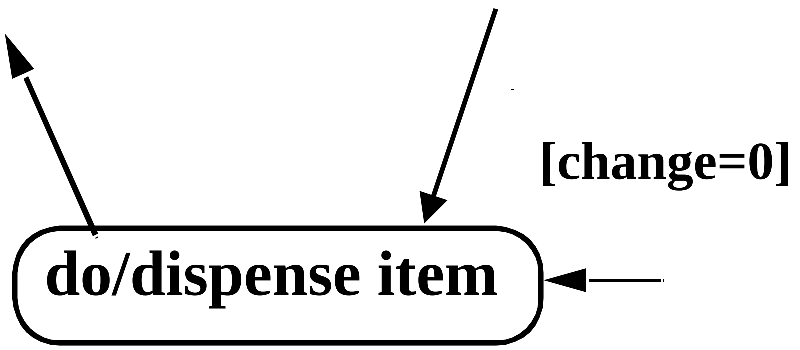
- Activities in states are composite items denoting other lower-level state diagrams
- A lower-level state diagram corresponds to a sequency of lower-level states and events that are invisible in the higher-level diagram.
- Sets of substates in a nested state diagram denoting a superstate are enclosed by a large rounded box, also called contour.

Example of a Nested Statechart Diagram

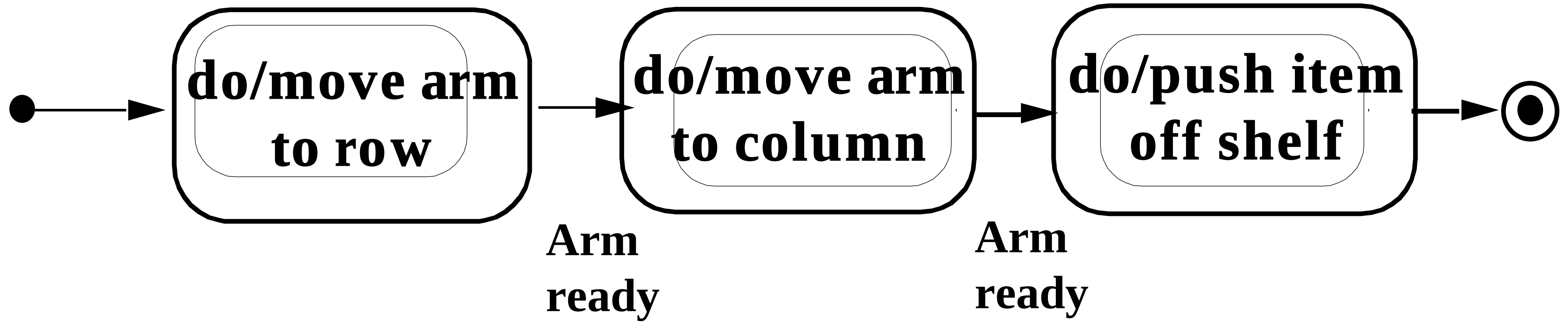


Expanding activity "do:dispense item"

'Dispense item' as an atomic activity:



'Dispense item' as a composite activity:



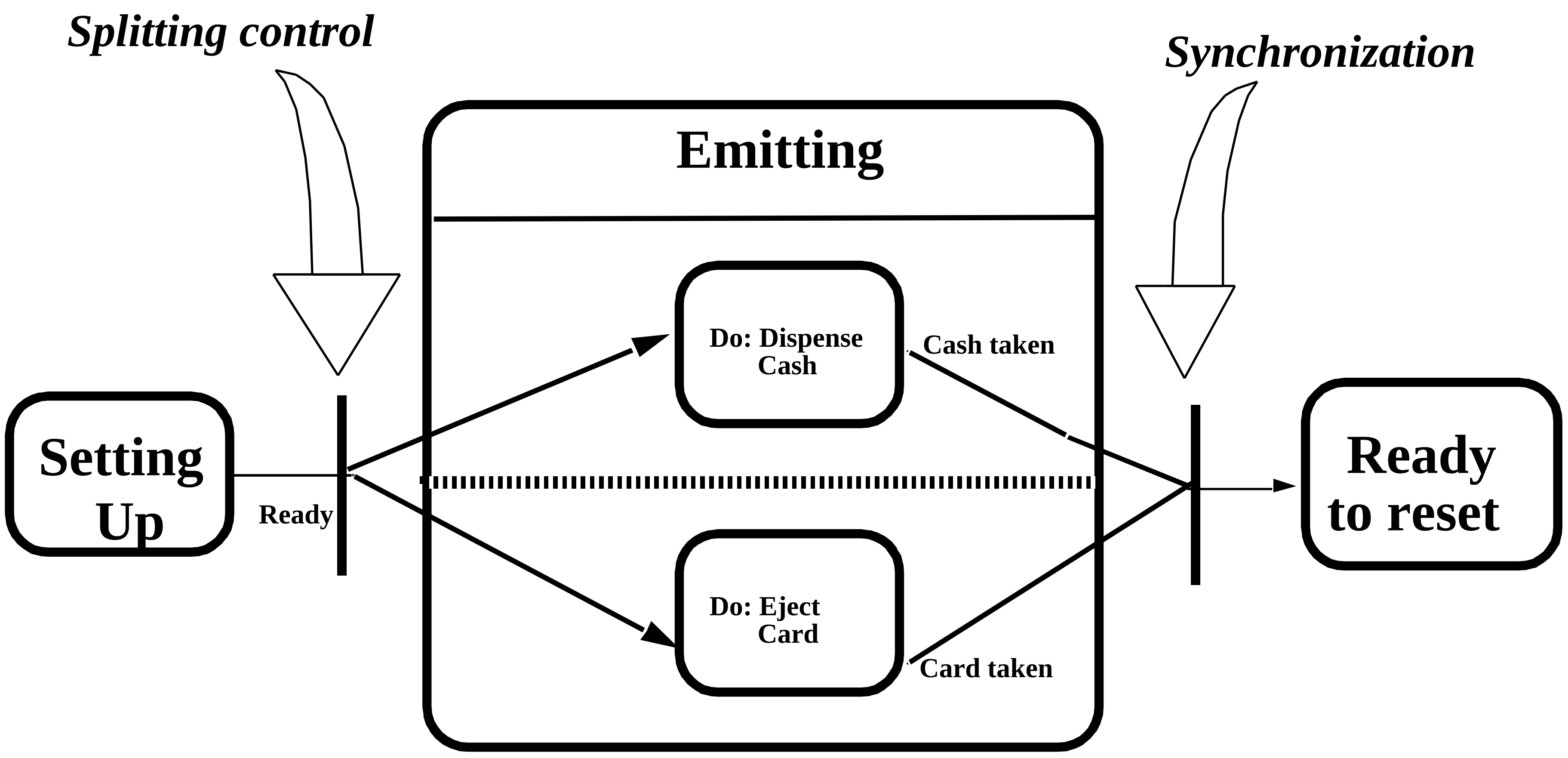
Superstates

- Goal:
  - **Avoid ravioli models**
  - **Reduce the number of lines in a state diagram**
- Transitions *from* other states to the superstate enter the first substate of the superstate.
- Transitions *to* 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
  - **The 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* 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 screen associated with the state (think instance diagram!) helps significantly when presenting the dynamic model of the user interface**
- Activities and 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**

Practical Tips for Dynamic Modeling

- Use use cases and scenarios when constructing statechart diagrams (ask the client)
- Construct state charts only for classes with significant dynamic behavior
- 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 control structure? **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 Requirements Analysis

1. Analyze the Problem Statement
   - **Identify functional requirements**
   - **Identify nonfunctional requirements**
   - **Identify constraints**
2. Build the Functional Model:
   - **Develop use cases to illustrate functionality requirements**
3. Build dynamic model:
   - **Develop sequence diagrams to illustrate the interaction between objects**
   - **Develop state diagrams for objects with interesting behavior**
4. Build 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 Cases continued

- *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?

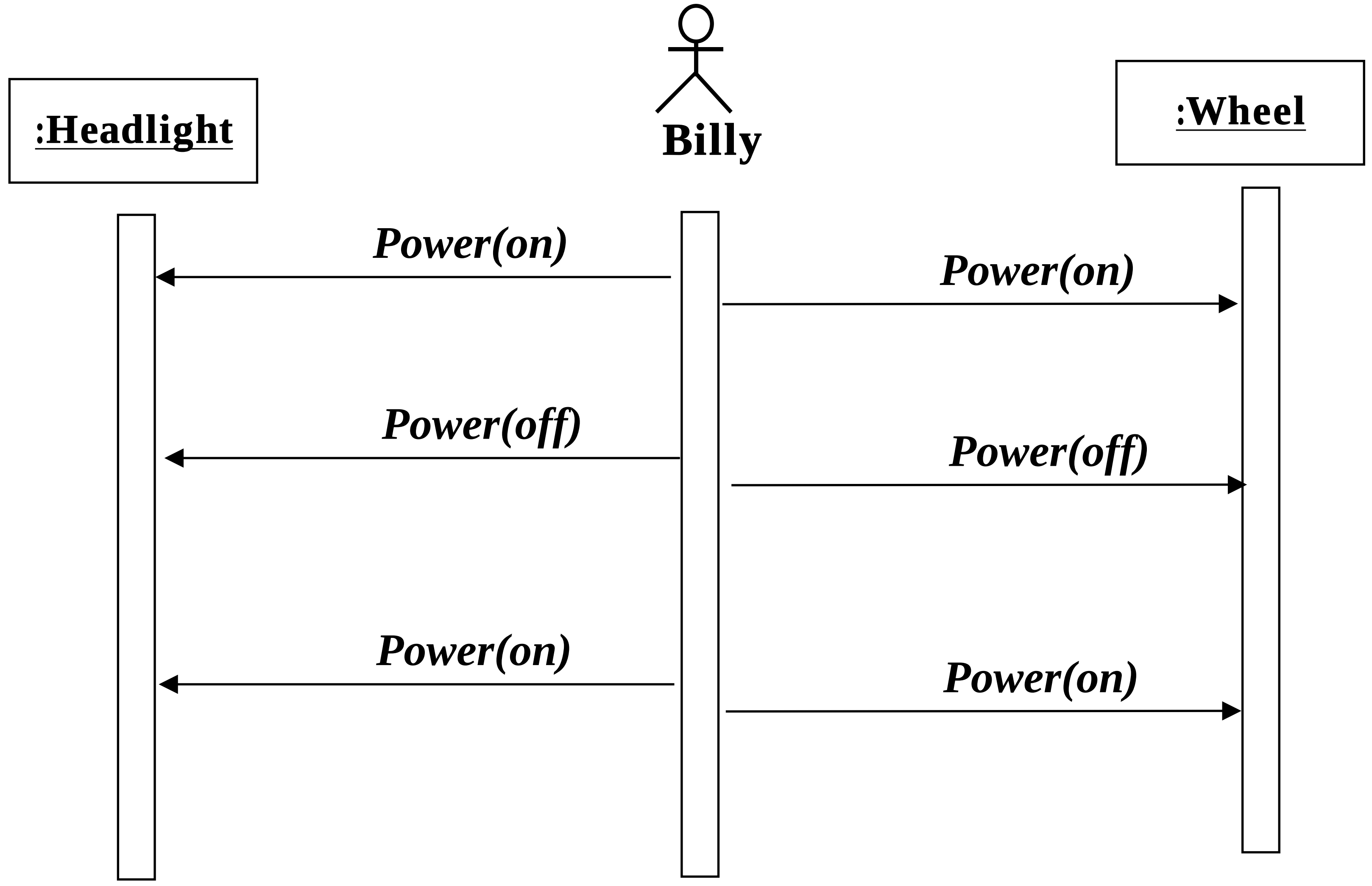
- *Use case 1: System Initialization*
  - **Entry condition: Power is off, car is not moving**
  - **Flow of events:**
    - **Driver turns power on**
  - **Exit condition: Car moves forward, headlight is on**

- *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.**

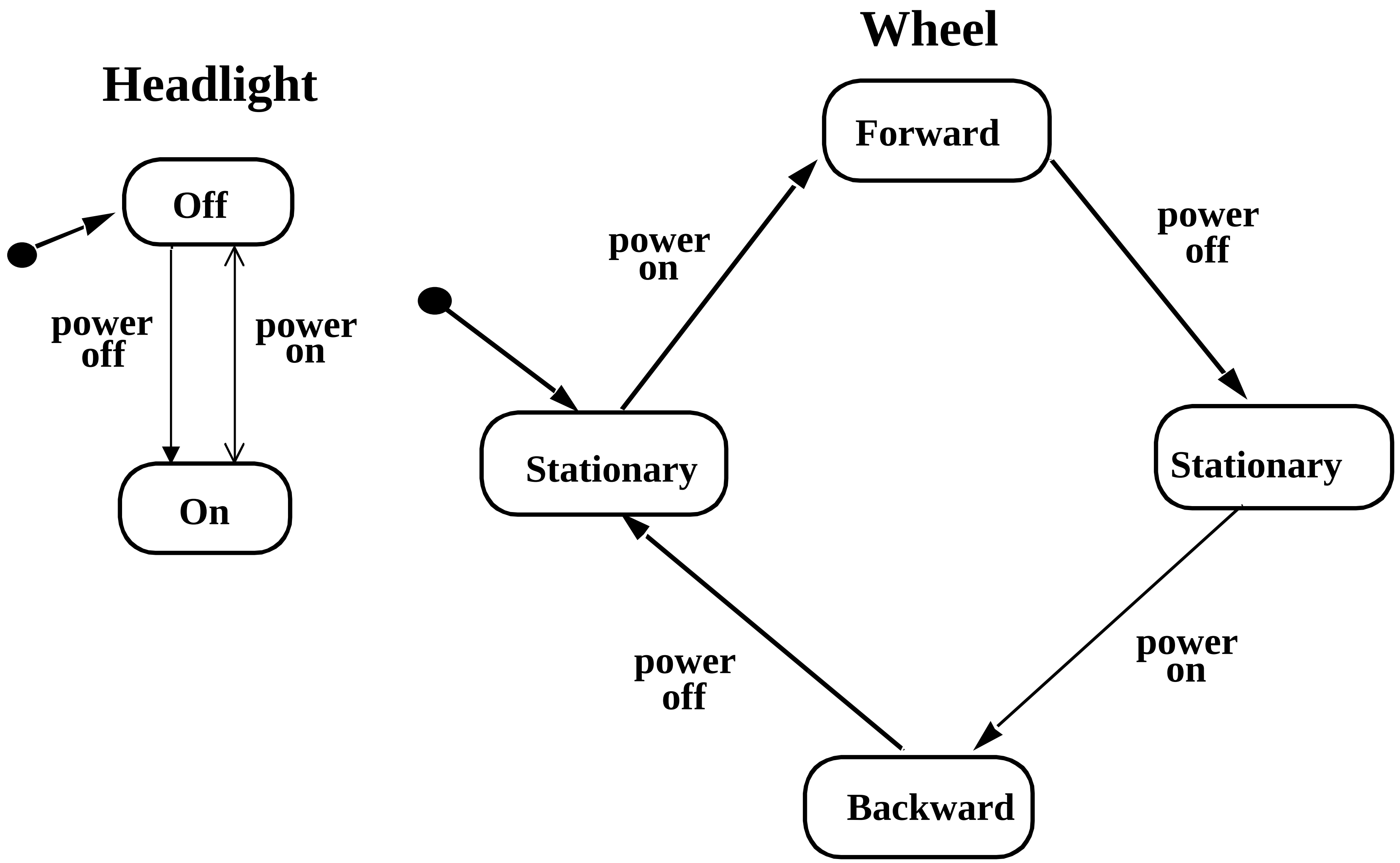
Create a Scenario

- Name: Drive Car
- Sequence of events:
  - **Billy (Actor!) turns power on**
  - **Headlight goes on**
  - **Wheels starts moving forward**
  - **Wheels keeps moving forward**
  - **Billy turns power off**
  - **Headlight goes off**
  - **Wheels stops moving**
  - **. . .**

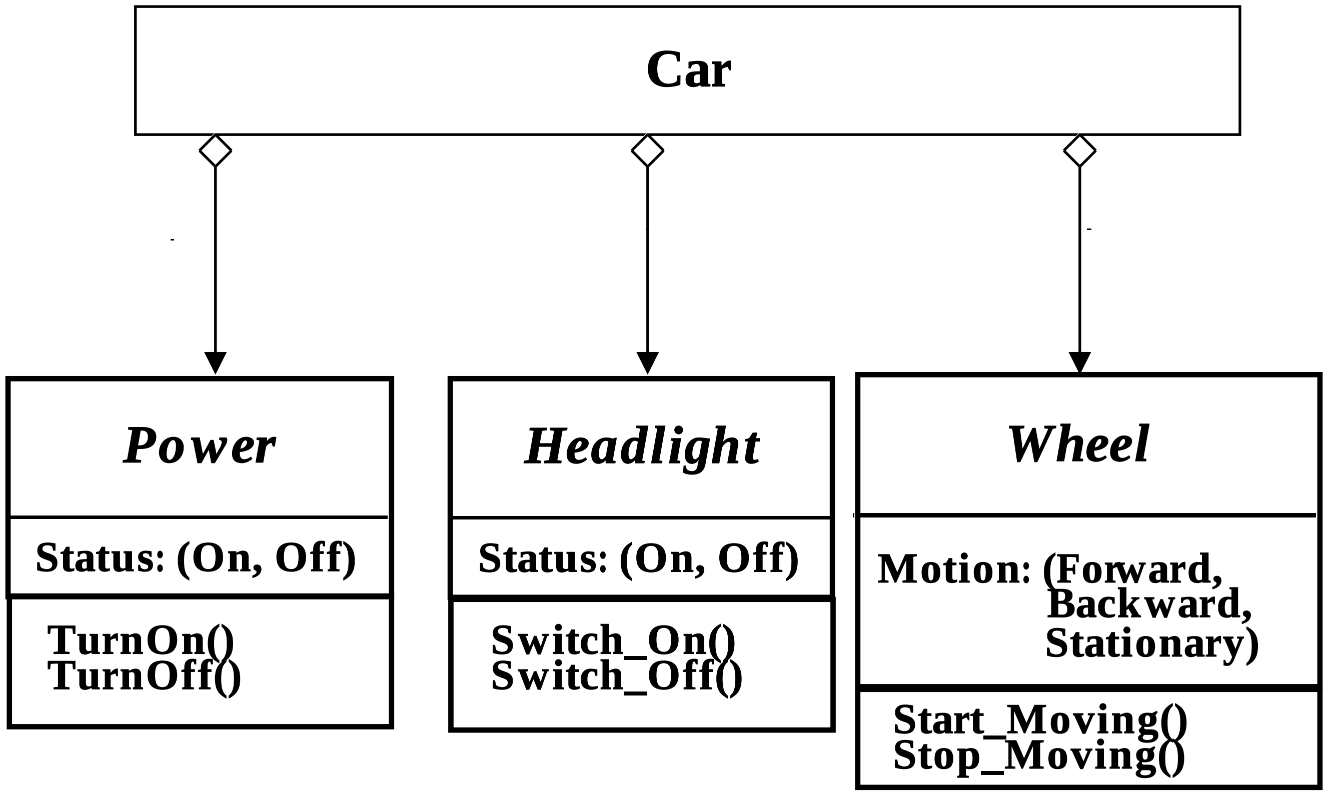
Sequence Diagram for Drive Car Scenario



Dynamic Model of Toy Train: 2 Statecharts



Toy Car: Object Model



When is a model dominant?

- ***Object model***:
  - The system has non-trivial data structures
- ***Dynamic model***:
  - The system has many different types of events: Input, output, exceptions, errors, …
- ***Functional model***:
  - The system performs complicated transformations such as difficult computations consisting of many steps.

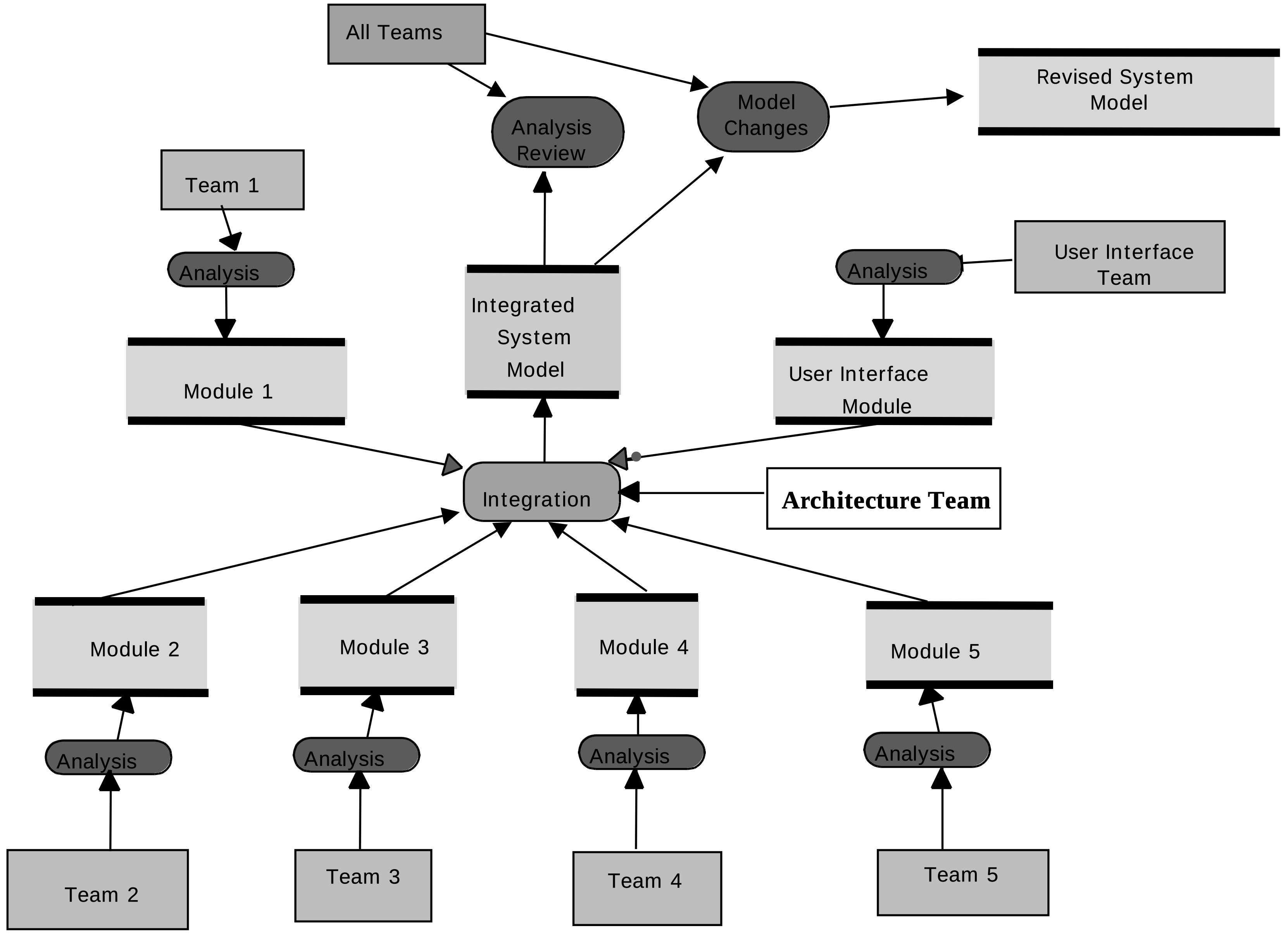
Examples of Dominant Models

- *Compiler:*
  - Functional model most important.
  - Dynamic model is trivial because there is only one type input and only a few outputs.
- *Database Management system:*
  - Object model most important.
  - Functional model is trivial, because their purpose is usually only to store, organize and retrieve data.
- *Spreadsheet program:*
  - Functional model most important.
  - Object model is trivial, because the spreadsheet values are trivial and cannot be structured further. The only interesting object is the cell.

Collaborative Requirements Analysis

- A system is a collection of subsystems providing services
- Analysis of services is provided by each of the teams who provide the models for their subsystems
- Integration of team models into the full system model by 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

Model Integration in Complex System Development



Requirements Analysis Document Template

- Location of Requirements Analysis Document Template for PAID:
  - **PAID Homepage, Work Products, Project Documents**
    - **Requirements Analysis Document (RAD)**

- Example of a RAD from an earlier 15-413 project
  - **OWL Project: http://cascade1.se.cs.cmu.edu/15-413**

Requirements Analysis Document Template

- 1.0 General Goals
- 2.0 Current System
  - **Description of current system**
- 3.0 Proposed System

  3.1 Overview

  3.2 Functional Requirements

  3.3 Nonfunctional requirements

  3.4 Constraints

  3.5 System Model

Section 3.5 System Model

3.5.1 Use case model

- Actors

- Use cases

3.5.2 Object model

- Data dictionary

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

3.5.3 Dynamic model

- State diagrams for classes with significant dynamic behavior

3.5.4 User Interface

- Navigational Paths

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?**

Nonfunctional Requirements, ctd

- 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?**
- 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?**

Nonfunctional Requirements, ctd

- 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?**
- 3.3.7 Quality issues
  - **What are the requirements for reliability?**
  - **Must the system trap faults?**
  - **Is there a maximum acceptable 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)?**

Nonfunctional Requirements, ctd

- 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, ...)?**
- 3.3.10 Security Issues
  - **Must access to any data or the system itself be controlled?**
  - **Is physical security an issue?**

Nonfunctional Requirements ctd

- 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?**

RAD Schedule

- 9/15 Release of Template (project management)
    - **Each team works on the RAD for their subsystem**
- 9/30 Team RADs are due (teams)
    - **Project management reviews the team RADs**
- 10/5 Reviews are due (project management)
    - **Teams incorporate review comments**
- 10/10 Second revision of team RADs due (teams)
- 10/15 Integrated version of RAD (architecture team, documentation team)