How do you find classes?

• We have already established several sources for class identification:
  • Application domain analysis: We find classes by talking to the client and identifying abstractions by observing the end user
  • General world knowledge and intuition
  • Textual analysis of event flow in use cases (Abbot)
• Today we identify classes from dynamic models
  • Two good heuristics:
    • 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

Dynamic Modeling with UML

• Two UML diagrams types for dynamic modeling:
  • Interaction diagrams describe the dynamic behavior between objects
  • Statechart diagrams describe the dynamic behavior of a single object.

UML Interaction Diagrams

• Two types of interaction diagrams:
  • 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,
How do we detect Operations?

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

What is an Event?

- Something that happens at a point in time
- An event sends information from one object to another
- Events can have associations with each other:
  - Causally related:
    - An event happens always before another event
    - An event happens always after another event
  - Causally unrelated:
    - Events that happen concurrently
- Events can also be grouped in event classes with a hierarchical structure => Event taxonomy

The term ‘Event’ is often used in two ways

- Instance of an event class:
  - "Slide 14 shown on Thursday May 9 at 8:50".
  - Event class "Lecture Given", Subclass "Slide Shown"
- Attribute of an event class
  - Slide Update(7:27 AM, 05/07/2009)
  - 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.

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 be the actor of the use case
  2nd column: Should be a boundary object
  3rd column: Should be the control object that manages the rest of the use case

- **Creation of objects:**
  - Create control objects at beginning of event flow
  - The control objects create the boundary objects

- **Access of objects:**
  - Entity objects can be accessed by control and boundary objects
  - Entity objects should not access boundary or control objects.

### ARENA Sequence Diagram: Create Tournament

- **Tournament**
  - `new` `newTournament(league)`
  - `setName(name)`
  - `setMaxPlayers(max)`
  - `commit()`
  - `createTournament(name, max)`

### Impact on ARENA’s Object Model

- Let’s assume ARENA’s object model contains - at this modeling stage - the objects
  - League Owner, Arena, League, Tournament, Match and Player

- The Sequence Diagram identifies 2 new Classes
  - Tournament Boundary, Announce_Tournament_Control

### Impact on ARENA’s Object Model (2)

- The sequence diagram also supplies us with 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

createTournament(name, maxp)
League
Owner
create Tournament(name, maxp)
Announce Tournament, Control
«new»
setName(name)
setMaxPlayers(maxp)
commit()
checkMax Tournament()
newTournament(league)
League Owner
Tournament Boundary
new Tournament(name, maxp)
create Tournament(name, maxp)
League Owner

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.
Dynamic Modeling

- We distinguish between two types of operations:
  - **Activity**: Operation that takes time to complete
  - **Action**: Instantaneous operation
- A state chart 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

- Note:
  - Events are italics
  - Conditions are enclosed with brackets: 
  - Actions and activities are prefixed with a slash /
  - Notation is based on work by Harel
  - Added are a few object-oriented modifications.

Example of a StateChart Diagram

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

- **Diagnostics Menu**
  - User moves cursor to Control Panel or Graph

- **Graph**
  - User selects data group and type of graph

- **Selection**
  - User selects data group
  - Field site
  - Car
  - Sensor group
  - Time range

- **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

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).

Let’s Do Analysis: A Toy Example

- **Analyze the problem statement**
  - Identify functional requirements
  - Identify nonfunctional requirements
  - Identify constraints (pseudo requirements)
- **Build the functional model**:
  - Develop use cases to illustrate functional requirements
- **Build the 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 forward with its headlight shining
- **Power is turned off**
  - Car stops and headlight goes out
  - Power is turned on
  - Headlight shines
  - Power is turned off
  - Headlight goes out
  - Power is turned on
  - Car runs forward with its headlight shining

Find the Functional Model: Use Cases

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

- **Use case 2: Turn headlight off**
  - Entry condition: Car moves forward with headlights on
  - Flow of events:
    1. Driver turns power off, car stops and headlight goes out.
    2. Driver turns power on, headlight shines and car does not move.
    3. Driver turns power off, headlight goes out
  - Exit condition: Car does not move, headlight is out

Use Cases continued

- **Use case 3: Move car backward**
  - Entry condition: Car is stationary, headlights off
  - Flow of events:
    1. Driver turns power on
  - Exit condition: Car moves backward, headlight on

- **Use case 4: Stop backward moving car**
  - Entry condition: Car moves backward, headlights on
  - Flow of events:
    1. Driver turns power off, car stops, headlight goes out.
    2. Power is turned on, headlight shines and car does not move.
    3. Power is turned off, headlight goes out.
  - Exit condition: Car does not move, headlight is out
Use Cases Continued

- **Use case 5: Move car forward**
  - **Entry condition:** Car does not move, headlight is out
  - **Flow of events**
    1. Driver turns power on
  - **Exit condition:** Car runs forward with its headlight shining

Use Case Pruning

- **Do we need use case 5?**
- **Let us compare use case 1 and use case 5:**
  - **Use case 1: System Initialization**
    - **Entry condition:** Power is off, car is not moving
    - **Flow of events:**
      1. Driver turns power on
    - **Exit condition:** Car moves forward, headlight is on
  - **Use case 5: Move car forward**
    - **Entry condition:** Car does not move, headlight is out
    - **Flow of events**
      1. Driver turns power on
    - **Exit condition:** Car runs forward with its headlight shining

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

- **Headlight:**
  - **State:** On, Off
- **Wheel:**
  - **State:** Forward, Stationary

Toy Car: Object Model

- **Car:**
  - **Methods:**
    - Start_Moving()
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

Model Validation and Verification

- **Verification** is an equivalence check between the transformation of two models
- **Validation** is the comparison of the model with reality
  - Validation is a critical step in the development process Requirements should be validated with the client and the user.
  - Techniques: Formal and informal reviews (Meetings, requirements review)
- **Requirements validation** involves several checks
  - Correctness, Completeness, Ambiguity, Realism

Checklist for a Requirements Review

- Is the model correct?
  - A model is correct if it represents the client's view of the system
- Is the model complete?
  - Every scenario is described
- Is the model consistent?
  - The model does not have components that contradict each other
- Is the model unambiguous?
  - The model describes one system, not many
- Is the model realistic?
  - The model can be implemented

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

Omissions in some UML Diagrams

- Class Diagram

  - Missing Association (Incomplete Analysis?)

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

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 (e.g. computations consisting of many steps).
  - Which model is dominant in these applications?
    - Compiler
    - Database system
    - Spreadsheet program
Examples of Dominant Models

- Compiler:
  - The functional model is most important
  - The dynamic model is trivial because there is only one type input and only a few outputs
  - Is that true for IDEs?
- 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.

Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
  3.1 Overview
  3.2 Functional requirements
  3.3 Nonfunctional requirements
  3.4 Constraints ("Pseudo requirements")
  3.5 System models
    3.5.1 Scenarios
    3.5.2 Use case model
    3.5.3 Object model
      3.5.3.1 Data dictionary
      3.5.3.2 Class diagrams
      3.5.4 Dynamic models
      3.5.5 User interface
4. Glossary

Section 3.5 System 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

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
   - 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.

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