CSE 308

UML Overview
Use Case Diagrams

Reference

- Class diagrams
  en.wikipedia.org/wiki/Use_case
What is Modeling?

- Modeling consists of building an abstraction of reality
- Abstractions are simplifications because:
  - They ignore irrelevant details and
  - They only represent the relevant details
- What is *relevant* or *irrelevant* depends on the purpose of the model, the audience, and other factors

This is a very difficult decision

---

Why Model Software?

Software is getting increasingly more complex

- Some versions of Windows > 40M lines of code
  
  Could you comprehend 40M LOC?

- Modifying a model of a system is much, much easier than modifying software
- We need simpler representations for complex systems
  - Modeling is a way for dealing with complexity

Remember, a course goal is to think first, code second
How Do We Deal With Complexity?

- Break it down into simpler parts
- Example - design specifications for a building
- Helps in
  - getting user/peer feedback
  - Getting approval
  - Avoiding construction problems

Systems, Models and Views

- A **model** is an abstraction describing a system or a subset of a system
- A **view** depicts selected aspects of a model
- A **notation** is a set of graphical or textual rules for depicting views
- Views and models of a single system may overlap each other

Unlike DB design, we often just generate different views, which together constitute a model
What is UML?

- **UML (Unified Modeling Language)**
  - A standard for modeling object-oriented software.
  - Resulted from the convergence of notations from three leading object-oriented methods:
    - OMT (James Rumbaugh)
    - OOSE (Ivar Jacobson)
    - Booch (Grady Booch)

- Supported by several CASE tools
  - Visio
  - Workbench
  - Visual Paradigm
  - Lucidchart

You can model 80% of most problems by using about 20% of UML (maybe 90/10)

UML Approach for CSE308

- **Use case Diagrams**
  - Describe the functional behavior of the system as seen by the user
  - Great for decomposing a system into buildable units

- **Sequence diagrams**
  - Describe the dynamic behavior between actors and the system and between objects of the system
  - Helps to define the objects that are needed to implement a use-case

- **Class diagrams**
  - Describe the static structure of the system: Objects, Attributes, Associations
  - Can be revised based on discoveries made from sequence diagrams
Other UML Notations

- UML provide other notations that are used less often
- Implementation diagrams
  - Component diagrams
  - Deployment diagrams
  - State-chart diagrams (essentially a finite state automaton)
  - Activity diagrams (essentially a flow chart)

UML Core Conventions

- Rectangles are classes or instances
- Ovals are functions or use cases
- Instances are denoted with colon notation
  - myWatch:SimpleWatch
  - joe:Firefighter
- Diagrams are graphs
  - Nodes are entities
  - Arcs are relationships between entities

A consistent code and design style is essential for group communication

Note the camel case notation
**CamelCase**

- A compound word begins each element with a capital letter
  - Upper camel case (UCC)
  - Lower camel case (LCC) – first letter not capitalized

**Examples**
- **UCC** – "CamelCase"
- **LCC** – "camelCase"

---

**Naming Conventions**

- Camel case for classes (upper cc) and attributes (lower cc)
- Classes – singular
- Attributes – singular (plural for collections)
- Avoid acronyms and abbreviations except where well known (e.g., PI for Principal Investigator)

Names should describe the application domain, not the implementation approach

Naming conventions are part of the "teamwork" approach to CSE308

Conventions apply very early in the process
### Use Case

- Used during requirements elicitation to represent external behavior
- *Use cases* represent an interaction sequence for a type of functionality
- The *use case* model is the set of all *use cases*. It is a complete description of the functionality of the system and its environment
- A *use case* consists of:
  - Unique name
  - Participating actors
  - Entry conditions
  - Trigger
  - Flow of events (scenario)
  - Exceptions
  - Build location (when available)
  - Exit conditions
  - Issues

A *use case* represents a class of functionality provided by the system as an event flow

### Actors

- An actor models an external entity which communicates with the system
- It can be a:
  - User,
  - External system, or
  - Physical environment
- An actor has a unique name
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Similar to a role
Example of a textual use case

Design issues:

- No overlap in use cases (instead think of preconditions)
- Look for use cases that cover multiple roles (with exceptions that differentiate the roles)
- Proper size (not too many steps or too few steps)

Class Exercise

Let’s review the use cases from a previous CSE308 class
Use Case: Summary

- Use case documentation
  - represents external behavior
  - are useful as an index into the use cases
  - Includes text and diagrams
  - Should be complete (all use cases need to be described)

We use use-case text for all processes

UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - Powerful, but complex language
  - Can be misused to generate unreadable models
  - Can be misunderstood when using too many exotic features

UML should be used to the extent that it improves communications concerning the system to be built