CS 494
Object-Oriented Analysis & Design

On to Design

Reminder: Analysis models
• Earlier we modeled requirements using...
  • Class Diagrams: Known as the Conceptual Model
    – Sometimes known as the logical model.
    – Classes represent domain-level entities. (E.g. things in the user’s world.)
      • Thus no classes for implementation-level things.
    – Associations model domain-level relationships. (E.g. user-understood relationships between things in the user’s world.)
      • Usually don’t show navigation on associations

Reminder: Analysis models (2)
• Use Cases and Sequence Diagrams
  – Scenarios in a Use Case can be represented by UML sequence diagrams
  – Objects in the sequence diagram could be either:
    • The system and the actors, or...
    • Domain-level entities modeled in the conceptual model (a class diagram)
  – Messages between objects are:
    • Again, at a high-level of abstraction
    • Scenario descriptions become messages

Reminder: Goals for design
• Create detailed “plans” (like blueprints) for implementation
• Build these from requirements models so we are confident that all user needs will be met
• Create design models before coding so that we can:
  – Compare different possible design solutions
  – Evaluate efficiency, ease of modification, maintainability, etc

UML Notations for Design
• Several UML notations provide various views of a design
• Class diagrams: Possibly created at two different levels of abstraction for design:
  – Specification level: Classes model types, and we focus solely on interfaces between software modules
  – Implementation level: Think of this as a true “software blueprint”. We can go directly to code from this model.
• Two types of Interaction Diagrams:
  – Sequence diagrams and Collaboration diagrams

UML Notations for Design (2)
• Sequence diagrams
  – Objects will be variables implemented in code
  – Messages are operations (e.g. C++ member functions) applied to objects
  – Sequence diagrams thus show how a sequence of operations called between a set of objects accomplishes a larger task
  – Sequence diagrams for a particular scenario help identify operations needed in classes
  – They also allow us to verify that a design can support requirements (e.g. a use-case scenario)
UML Notations for Design (3)

- State diagrams
  - Models how a particular object responds to messages according to its state
  - For a single object, show states and transitions between states
  - Transitions may be conditional based on a guard condition
  - May show an action an object takes on transition, or also activity carried out within a state
  - Occasionally used to model a system’s or subsystem’s behavior (not just one object’s)

UML Notations for Design (4)

- Packages
  - A simple notation that groups classes together
  - Possible to use this to show contents of a subsystem
    - Show dependencies between packages
    - Show visibility of classes between packages
  - Not really a rich enough notation for diagramming software architectures
- Component Diagrams
  - Models physical modules of code (e.g. files, DLLs, physical databases)

Design Process

- There are many different approaches to design, but here is something typical.
- First, create a model of the high-level system architecture
  - UML does not really provide a notation this
- Next, use the conceptual class model to build a design-level class model or models
  - Here we’ll assume we’re just building an implementation-level class model
- Also, model dynamic behavior using interaction diagrams.

Design Process (cont’d)

- We’ll use sequence diagrams with objects from the implementation-level class model
  - Sequence diagrams show how design-level objects will carry out actions to implement scenarios defined as part of use-case analysis
  - Messages between objects are member-function calls between objects
  - Important: Only member-function calls are shown, but other language statements (e.g. assignments) are executed between calls (of course).

Design Process (cont’d)

- Important: Development of class and sequence diagrams is iterative and concurrent
- When we create sequence diagrams for a new scenarios, we discover classes and operations that need to be added to the class model
- The two models grow together. Neither is a complete view of the system.
- Other documentation in text form is often used to provide details about class diagrams and sequence diagrams

Specification-Level Class Diagrams

- How does a design-level class diagram differ from a conceptual-level diagram?
  - No longer just an external view!
  - We are now modeling “how” not just “what”.
- This class diagram must document:
  - Additional classes
  - How you will implement associations
    - Multiplicity, Navigability or Direction;
    - Association classes
Additional Classes in a Design

- Are additional classes needed? Of course!
- Design-level “internal” classes
  - Data manager classes. E.g. collection objects that were simply associations before
  - Facilitator or helper classes that assist with complex tasks (e.g. ObservableComponent)
  - Factory classes that assist in creating new objects
  - Classes to implement other design patterns
- Is there any guidance or strategy for determining these?

Class Types in a Layered Architecture

- From Ambler, Sect. 7.1
- 5-layer model
- Classes only interact within layers, or as shown by arrows
  - Direction matters!
- Next slide describes these

Possible Design Class Types

- UI classes
- Business/Domain classes
  - Implement domain objects from Analysis
  - Data objects plus their behaviors
- Controller/Process classes
  - Implement business logic, collaborations between business objects and/or other controller
- Persistence classes
  - How to store, retrieve, delete objects
  - Hides underlying data stores from rest of system
- System classes
  - Wrap OS-specific functionality in case that changes

Controller/Process Layer

- Implements business logic
  - Encapsulate a business rule (Ambler, Sect. 3.6)
  - These often require interactions etc. between objects of different classes
- Example from a student course enrollment system:
  
  When can a Student enroll in a Seminar?
  - Depends on schedule, pre-requisites, other constraints

More on Controllers

- Why not just put business logic into the Domain class?
  - Business rules change. We want domain classes to be reusable.
  - In UI class? Then must use that UI to carry out this process. (Too tightly coupled.)
- How to find Controller classes?
  - To start: consider one for each use-case
  - If trivial or belongs in domain class, don’t.

Ambler’s Controller Class Example

- Example in Ambler, page 259
  - Class: EnrollInSeminar (what’s interesting about that name?)
    - Has link to a Student object
    - An instance given to SeminarSelector object (UI), which calls seminarSelected(seminar) on it
    - If Student/Seminar combination is OK
    - An instance given to FeeDisplay object (UI), which makes sure user willing to pay
    - If so, it’s verifyEnrollment() is called to finalize enrollment
Controller Classes: Good OO?
- Violates a principle of the OO approach!
  - Data and behavior kept together!
- Yes, but is this always the best solution?
  - DVDs and DVD players -- why not one unit?
  - Cameras and film vs. disposable cameras
- Consider coupling, change, flexibility...
- Controller classes are an example of the Mediator design pattern
- Mediator or control classes might grow to become god classes
  - too much control, or too centralized

Implementing Associations
- How associations are implemented is affected by multiplicity.
- Where they are implemented depends on navigability.
  - In one class or in both?
  - Until now we may not have worried about direction of associations. That's fine!
  - Often navigability cannot be determined until design phase.
  - Often it changes as we do more design.
  - In prototypes we often keep links bidirectional for flexibility.

Implementing Associations (2)
- Often we use class operations to hide implementation details of associations
  - getters, setters, traversal functions, update functions, etc.
  - Don’t forget: in C++, in-line functions are efficient
  - Also, derived associations (or attributes) are implemented as member functions that calculate something that is not stored directly in the class.

One-Way Associations
- If an association should just be navigable in just one direction, use the “arrow form” of the UML association in your class diagram.
  - In UML no arrows means two-way or bi-directional.
- For implementation, the “target” object becomes an attribute in the class
  - In C++, it could be stored as an embedded object or as a pointer
  - In Java, objects are always references variables (so embedded objects really are pointers)
- Consider using association name or role name from the class diagram to name this attribute

Multiplicity and One-Way Associations
- If the multiplicity is “1” or “0..1” then the attribute would be a pointer to an object of the target class
  - E.g. attribute in class Phone: selectedLine: Line*
- If the multiplicity is “many” but has a fixed maximum, then use array of pointers (or objects)
  - E.g. “3”, “0..3”, “2..4”
- If no fixed maximum, e.g. “1..” or “0..”**, then use a collection object as an attribute that stores an arbitrarily large number of pointers or objects
- For qualified associations use a hash-table or map object to associate key with target object

Multiplicity and One-Way Assoc. (2)
- Examples using the C++ standard library...
  - A vector class is like an array with no maximum capacity
    - Example attribute in class Phone:
      linkedLines: vector<Line*>
  - Other C++ classes might be appropriate too: set, list
    - Arrays should only be used if you know the maximum
- Note: Your team might agree not to show the “*” to indicate pointers. Conventions vary.
Implementing Two-Way Associations

• Three options, depending on your needs
  – Note: Sometimes it’s OK if traversal in one direction is slower than the other
• Option One: Just like one-way but in both classes
  – Advantages: Equally efficient in both directions
  – But, requires more space
  – Also, updating links between objects is more complex
    • Often a good idea to use member functions to handle updates to links.

Implementing Two-Way Assoc. (2)

• Option Two:
  – In one class, Class A, implement just like one-way (see above) to access Class B objects.
  – In second class, Class B, write an operation that uses some kind of search of all objects of Class A to find the one that points back to the current B object.
    • Why? Saves space if access from B to A is very rare
    • But, requires there to be some place where all objects of Class A are stored

Implementing Two-Way Assoc. (3)

• Option Three: Implement an Association Class
  – This class will have only one instance, which stores all the links between objects of the two classes
  – Implemented as two dictionary or map objects
    • One points to Class A objects, the other to Class B objects
  – Search of this object is used to find links for one object

Example of Assoc. Object

• A person works for one company. A company has many employees.
• If pointers are not “bi-directional”, then Works-For object must support efficient look-up of a Person object in order to find that object’s company.
• Note: This is not a UML diagram!

Flashback to previous slides...

• Slides on class diagrams had “unused slides” at the end.
• Let’s look at some of those now.

Association Classes

• Recall that qualified associations really mean that the link between two objects has an attribute
• Often associations are “first-class” things
  – They have a life-time, state, and maybe operations
  – Just like objects!
• Association classes
  – Same name as the association because...
  – They represent the same thing!
**Association Class Example**

```
Company
  0..* employer
  employee 1..*

Person
  Job
```

**World Cup Example**

- We need a system to handle the World Cup. Teams represent countries and are made up of 22 players.
- Countries qualify from zones, where each zone is either a country or a group of countries.
- Each team plays a given number of games in a specific city. Referees are assigned to games. Hotel reservations are made in the city where the teams play.

**World Cup Problem: Class Model**

```
Qualifying Unit
  Country    Zone
  Team       Player
  Hotel      City
  Game
  Reservation
  Assignment
```

**Implementing Association Classes**

- Implementation depends on multiplicity
- If one-to-one, then it would be possible to...
  - Put attributes and operations inside either object
  - Or, put them in a separate class that’s linked to either object
- If one-to-many, then same choices as one-to-one, but do this for the object on the “many” end
  - Again, could be a separate object (see next case)
- If many-to-many, you need a separate class with an object instantiated for each link

**Example of Association Class Implementation**

- Conceptual-Level Class Diagram
- Corresponding Design-Level Class Diagram
Notes on Example Implementation

• No direct link (pointer) in design or implementation between ClassA and ClassB instances! But...

• Each instance of an AssocClass object is linked to exactly one ClassA object and also to one ClassB object
  – This forms a 3-tuple for each conceptual-level link between a pair of ClassA and ClassB objects

• Note multiplicities reflect concept level:
  – One ClassA object is linked to 1-to-many AssocClass/ClassB pairs. Great!
  – One ClassB object links to 0-or-one AssocClass/ClassA pairs. Yes!