Lecture 11: Requirements Specification

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2003

Credit where Credit is Due

- Some material presented in this lecture is taken from section 4 of Maciaszek's "Requirements Analysis and System Design". © Addison Wesley, 2000

Goals for this Lecture

- Cover the material presented in Section 4 of the textbook
  - Introduce Requirements Specification
  - Provides more insight into OO Analysis
    - This chapter provides **many** examples

Requirements Specification

- Produces three types of models
  - State Models (This Lecture)
    - Use Cases (some actors become classes)
    - Class Diagrams
  - Behavior Models (Lecture 12)
    - Activity Diagrams
    - Interaction Diagrams
  - State Change Models (Lecture 12)
    - State Chart Diagrams

Requirements Specification

- Models are developed iteratively
  - Taking into account use cases and constraints (developed during requirements elicitation)
- Each model, or diagram, represents a view into the system; the models, taken together, allow developers and customers to view the system from multiple perspectives
- We now examine each type of model in more detail

State specifications

- The **state** of an object is determined by the values of its attributes and associations
  - A BankAccount may be "overdrawn" when its balance is negative
- Since object states are determined from data structures, the models of the data structures (e.g. classes) are called **state specifications**

State Specifications

- State specifications provide a static view of the system
  - The attributes and associations of classes do not change dynamically
- The main task is to specify the classes of an application domain
  - only attributes and associations; operations are derived from the behavior specification

State Specification

- Define entity classes
  - Persistent classes in the app. domain
    - aka business objects
- How to do this? The process is highly dependent on the analyst's
  - knowledge of class modeling
  - understanding of the application domain
  - experience with similar and successful designs
  - ability to think forward and predict consequences
  - willingness to revise the model iteratively

Discovering Classes

- Four Approaches
  - Noun Phrase Approach
  - Common Class Patterns
  - Use Case Driven (already covered)
  - Maciaszek's Guidelines
  - CRC (Class-Responsibility-Collaboration)
    - I will be providing expanded coverage of this technique (as compared to the information presented by your textbook)

Noun Phrase Approach

- Examine the requirements and underline each noun
  - Each noun is a *candidate class*
- Divide list of candidate classes into
  - Relevant Classes
    - Part of the application domain; occur frequently in reqs.
  - Irrelevant Classes
    - Outside of application domain
  - Fuzzy Classes
    - Unable to be declared relevant with confidence; require additional analysis
- Experience will eventually enable designers to avoid generating irrelevant classes

Noun Phrase Approach, continued

- This technique now considered naïve
  - While it may help in identifying domain objects, it is not good at identifying objects that live in the application domain
  - Thus, it can help at the beginning of analysis, but you will not return to it as you move into design
    - Finding good objects during design means identifying abstractions that are part of your application domain and its execution machinery
    - Objects that are part of your application domain will have a tenuous connection, at best, to real-world things
      - e.g. what's the correspondence of a scrollbar to the real-world

Common Class Patterns

- Derive classes from the generic classification theory of objects
  - Concept class - a notion shared by a large community
  - Events class - captures an event that demarks intervals within a system
  - Organization class - a collection or group within the domain
  - People class - roles people can play
  - Places class - a physical location relevant to the system

Common Class Patterns

- Rumbaugh proposes a different scheme
  - Physical Class (Airplane)
  - Business Class (Reservation)
  - Logical Class (FlightTimeTable)
  - Application Class (ReservationTransaction)
  - Computer Class (Index)
  - Behavioral Class (ReservationCancellation)
- These taxonomies are meant to help a designer think of classes, however it is difficult to be systematic. (This technique is probably only useful during early analysis as well)

Maciaszek's Guidelines

- Each class must have a statement of purpose in the system
- Each class is a template for a set of objects
  - avoid singleton classes
- Each class must house a set of attributes
- Each class should be distinguished from an attribute
  - e.g. Color may be an attribute of a Car class, but may be needed as a full class in a paint program
- Each class houses a set of operations that represents the interface of the class
  - operations can be derived from the statement of purpose

CRC Cards

- CRC stands for
  - Candidates, Responsibilities, Collaborators
- Meant primarily as a brainstorming tool for analysis and design
  - Rather than use diagrams, use index cards
  - Rather than record attributes and methods, record responsibilities
- *Some material on CRC cards drawn from Object Design by Wirfs-Brock and McKean, © 2003*

Unlined Side of Card

- On the unlined side of the index card, we write an informal description of each candidate's purpose and role

Document

Purpose: A Document acts as a container for graphics and text

Role: Container
Pattern: Composite

Lined Side of Card

- On the unlined side of the index card, we write an informal description of each candidate's purpose and role

Document ← *candidate*	
Knows contents	TextFlow
Knows storage location	
Inserts and removes text,	
graphics, and other elements	

responsibilities *collaborators*

Not Just Index Cards

- Post-It Notes can be used for even less "structure"; might be easier when brainstorming

> **Document**
> Purpose: A document represents a container that holds text and/or graphics that the user can enter and visually arrange on pages

Why index cards?

- Forces you to be concise and clear
  - and focus on major responsibilities
  - since you must fit everything onto one index card
- Inherent Advantages
  - cheap, portable, readily available, and familiar
- Affords Spatial Semantics…
  - Close collaborators can be overlapped
  - Vertical dimension can be assigned meanings
  - Abstract classes and specializations can form piles
- …which provides benefits
  - Beck and Cunningham report that they have seen designers talk about a new card by pointing at where it will be placed

Class Activity Section

- Let's try it!
- Pick one of four domains
  - Banking (checking & saving accounts, etc.)
  - Airline Reservations
  - Document Processor
  - Weblog Reader/Editor

Examples in Textbook

- Pages 112-133 work through four examples of class specification in detail
  - class discovery
  - then specifying
    - attributes
    - associations
    - aggregations/compositions
    - inheritance
  - We shall follow the University Enrollment example

University Enrollment

- Requirements specified on pages 112-113 and 117
- After reading the first set of requirements, candidate classes are identified on page 114
  - We will delay creating a class diagram until we consider attributes

Specifying Attributes

- Attributes are specified in parallel with classes
  - initial set of attributes will be "obvious"
  - important to initially select attributes that help to determine the states of the class
    - additional attributes can be added in subsequent iterations
- Example cont.: After reading a second set of requirements on page 117, an initial class diagram (with attributes) is presented on page 4.1

Specifying Associations

- Associations connect objects in the system
  - they facilitate collaboration between objects
- Specifying associations involves
  - naming them
  - naming the roles
    - especially useful in self associations
    - note, a role name becomes an attribute in the class on the opposite end of the association
  - determining multiplicity
- What associations might we put into the University example?

Specifying Aggregation/Composition

- "Whole-part" relationships between composite and component classes
  - UML models aggregation as a constrained form of association
- Maciaszek suggests additional power
  - ExclusiveOwns and Owns
  - Has and Member
- Litmus test: "has" or "is-part-of" is needed to explain relationship

Example, continued

- Aggregations for the University example is shown in figure 4.6 on page 129
- Student and AcademicRecord participate in a composition relationship
- A Course aggregates its various CourseOfferings

Specifying Generalizations

- Looking for common features among classes
  - Move common features up a class hierarchy and specialized features down
- Apart from inheritance, generalization has two objectives
  - substitutability and polymorphism
- Litmus test: "can be" and "is-a-kind-of" required to explain relationship
- Are there any generalizations that we can make in the University example?

Summary

- Requirements Specification
  - Involves creating state, behavior, and state change models
  - We looked at state models today in depth
  - How do we find classes in the first place?
    - Looked at CRC Cards in depth
    - We will be returning to their use in design later this semester
  - How do we then find attributes and associations
    - Associations have many types, including composition, aggregation and generalization