Lecture 9: Finding Objects

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2005

Goals for this Lecture

- Review content of Chapter 3 from the textbook
- Discuss the process for discovering candidate objects and roles in a software system
- Review techniques that can aid this process
  - Design Stories
  - Search Strategies
  - Coming up with Names
  - Describing Candidate Objects and their relationships

© University of Colorado, Boulder, 2005

Laying a Foundation

- Wirfs-Brock and McKean compare object design to graphic design
  - Good graphic design requires careful use of color, texture, and shapes to make images “leap off the page”.
    - A bad design muddles what should be emphasized
      - Example: “chart junk”
  - A good design is more than the sum of its parts.
  - Object designs, likewise, require good abstractions, well-formed objects, and a good overall structure
    - To create these designs, however, you need a process

© University of Colorado, Boulder, 2005

A Process for Finding Objects

- Initial strategies for finding objects were a bit naïve
  - Take text that describes the requirements for the system and
    - Underline nouns → Objects!
    - Underline verbs → Methods!
  - This strategy is inadequate because finding good objects involves finding abstractions that are going to be useful for your application
    - Some of these abstractions may not have real-world counterparts
    - Although we must determine which domain concepts WILL be included and how they will fit within the overall application

© University of Colorado, Boulder, 2005

Finding Objects, continued

- However, this does not mean we can't be systematic!
  - We can "find objects" via
    - our knowledge of the application domain
    - our knowledge of "application machinery"
    - lessons learned from other designers (think patterns!)
    - our past design experiences (you'll get better with each new system)

© University of Colorado, Boulder, 2005

The Process

1. Write a brief design story; Identify what is important about your application
2. Use this story to identify several major themes that define some central concerns for your application
3. Search for candidate objects that surround and support each theme
4. Check that each candidate represents a key domain concept
5. Look for candidates that will help your application interact with these key domain concepts
6. Name, describe, and characterize each candidate

© University of Colorado, Boulder, 2005

The Process, continued

7. Organize your candidates; Look for clusters of objects that have to work together to solve a problem (use cases can help here)
8. Double check to see if each candidate is appropriate
9. Defend each candidate's reasons for inclusion
10. When discovery slows, move on to creating responsibilities (chapter 4) and collaborations (chapters 5 and 6)

© University of Colorado, Boulder, 2005

Discussion

- Again, this process is not meant to be performed in a sequential manner; you may do several steps at once, you might discard several objects at once and start over, etc.
- Wirfs-Brock and McKean recommend that you do start with a design story, however
- The goal is to come up with a core set of initial object candidate that represent the fundamental abstractions upon which your system is based
  - Many additional candidates will be created as you move forward in analysis and design

© University of Colorado, Boulder, 2005

Find Objects FIRST

- Your first candidates should be concrete objects or roles
  - These candidates should be “smart”
    - They “do things” in your system
    - They may “know things” about your system as well, but they do things in response to what they know
  - So, identify distinct objects with clear roles first; then identify their responsibilities and their relationships
- Classes and Interfaces will come later once you have enough concrete objects to understand the key relationships in your design
  - Identify what objects have attributes and behaviors in common (classes) and what objects have common responsibilities (interfaces)

© University of Colorado, Boulder, 2005

Getting Started: Design Stories

- To make finding objects easier, create a framework for searching for candidates by writing a story about your application
  - This allows you to identify candidates that “fall into place” and support various aspects of your story
  - The story should include not only functionality but your goals with respect to the software under design; what are the “cool things” about what you are trying to do and what things are you unsure about?
    - Try to coalesce information from multiple sources, like use cases, other requirements, system architecture, users, etc.

© University of Colorado, Boulder, 2005

Design Stories: How to Do It

- Write a rough story—top paragraphs, more or less
  - Don't take a lot of time revising and polishing it
  - What's notable about the application? What is it supposed to do?
  - Is it connected to a real-world example?
  - Have you done something similar in the past?
- If you are a member of a large design team
  - Write your own story first; then merge with other team members
- Try to identify important themes within each story

© University of Colorado, Boulder, 2005

Design Stories: Examples

- Lets look at the examples in the text book
  - Page 81 contains a story about Internet banking services
  - Page 82 contains a story about an Internet game, Kriegspiel
- Both stories were written quickly: one rambles while the other is more focused
- The key themes for the former are
  - Modeling online banking services, flexibly configuring behavior, sharing scare resources among thousands of users, supporting different views of accounts and access privileges
- The key themes for the latter are
  - Game modeling, a computer opponent, partitioning responsibilities across distributed components

© University of Colorado, Boulder, 2005

Search Strategies

- Themes in Design Stories can lead to candidates
- Candidates will generally fall into one of these categories
  - The work your system performs
  - Things directly affected or connected to the application
  - Information that flows through your software
  - Decision making, control and/or coordination activities
  - Structures and groups of objects
  - Domain Concepts
- What do these categories remind you of?

© University of Colorado, Boulder, 2005

Role Stereotypes!

- As discussed before, objects need to have a clear role and these roles will often match the stereotypes we covered in Lecture 4
  - If your system performs computations, look for service providers
  - If your system interacts with the outside world, look for interfacers
    - With respect to users, only include them in your object model if you need to treat different types of users in different ways
  - If your system handles lots of events, look for controllers
  - If your system manipulates lots of information, look for structurers
- Now, lets see how these are used to explore our two examples (pages 85—87) in the textbook

© University of Colorado, Boulder, 2005

What's In a Name

- Your candidates need strong names
  - When the name of an object is spoken, designers infer something about the object's role and
- So make sure an object's name fits its responsibilities
- Wirfs-Brock and McKean provide several heuristics to use while naming object candidates
  - They note that multiple naming systems (roles, patterns, domain concepts) can coexist within a single application

© University of Colorado, Boulder, 2005

Naming Heuristics

- Qualify Generic Names
  - This conveys both a general set of responsibilities and a specific type of behavior
    - Calendar vs. GregorianCalendar vs. JulianCalender
- Include only the most revealing and salient facts in a name
  - Timer vs. MillisecondTimerAccurateWithinPlusOrMinusTwoMilliseconds (!!)
- Give service providers "worker" names
  - StringTokenizer, SystemClassLoader, AppletViewer, etc.

© University of Colorado, Boulder, 2005

Naming Heuristics, continued

- Names that convey broad responsibilities may imply the need for additional objects
  - AccountingService may be useful initially but may eventually be replaced with more specific services: PaymentService or TransferFundsService
  - Keep the generic name if you can think of at least three specializations; otherwise lose the name
  - This is a “black art”; choosing names that convey enough meaning without being overly restrictive is hard!

© University of Colorado, Boulder, 2005

Naming Heuristics, continued

- Choose a name that does not limit behavior
  - Account vs. AccountRecord
    - the former can take on more responsibilities than the latter
- Choose a name that lasts for a candidate’s lifetime
  - ApplicationCoordinator vs. ApplicationInitializer
    - the latter indicates that it will only be around at the launch of a program
- Choose a name that fits your current design context
  - Names that sound reasonable for accounting applications may not make sense for other domains

© University of Colorado, Boulder, 2005

Naming Heuristics, continued

- Do not overload names
  - even though some OO languages support this
  - Example: having two objects called Processor (each in different packages) that may process different things or "process" in different ways
- Eliminate name conflicts by adding an adjective or using a synomym
  - TransactionProperties vs. Properties
    - as long as the two objects don't do radically different things
- Choose names that are readily understood
  - Account vs. Acct

© University of Colorado, Boulder, 2005

Describing Candidates

- Use CRC Cards to describe candidates
  - Record name, description, and role stereotypes
- Use patterns when describing candidates
  - See examples on page 94
- Provide examples of how a candidate will be used to clarify its purpose (these examples will probably not fit on the CRC card)

© University of Colorado, Boulder, 2005

Connecting Candidates

- Cluster candidates to help you clarify existing ones and “discover” new ones
- Feel free to rearrange your clusters to gain new insights
- Try clustering by
  - application layer
  - use case
  - stereotype role
  - object neighborhood
  - abstraction level
  - application theme

© University of Colorado, Boulder, 2005

Looking for Common Ground

- Once you have identified a bunch of distinct candidates, its time to look for commonalities
- These commonalities will help you identify classes and interfaces
- Strategies
  - Look for powerful abstractions and common roles
    - Car, Boat, Bike, Tractor → Vechicle
  - Look for the right level of abstraction
    - ChessMove vs. PawnMove, RookMove, etc.
  - Discard candidates if they can be replaced by a shared role
    - Book, CDs, DVDs, etc. → InventoryItem

© University of Colorado, Boulder, 2005

Defending Candidates

- You should be able to state why each candidate is worth keeping
- Keep a candidate if you can
  - Give it a good name
  - Define it and give it a stereotype
  - Show that it can be used in a use case
  - Assign it one or two initial responsibilities
  - Understand how other objects view it
  - Differentiate it from similar candidates

© University of Colorado, Boulder, 2005

Defending Candidates, cont.

- Discard a candidate when it
  - has responsibilities that overlap those of other candidates that you like better
  - seems vague
  - appears to be out of scope
  - doesn't add value to the design
  - seems insignificant or "too clever" or too much for what you need to accomplish

© University of Colorado, Boulder, 2005