Lecture 10: Use Case Patterns

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2003

Goals for this Lecture

- Look at a number of Use Case Patterns
  - from the book
    - Patterns for Effective Use Cases
      - by Steve Adolph and Paul Bramble
      - Addison-Wesley and Pearson Education, Inc.
      - © 2003
      - ISBN 0-201-72184-8

What are Patterns

- The "pattern movement" has its origins in Christopher Alexander's work in the late 1970s to define pattern languages for designing cities and communities
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem…"
- Design patterns (which we cover later in the semester), thus, are useful solutions to common design problems
- Use Case patterns, then, contain solutions to problems that are related to creating and maintaining a set of use cases

Common Misconceptions

- Patterns offer a complete methodology in and of themselves
  - They only offer solutions to specific problems; they do not provide a complete picture of a given domain
- Using patterns guarantees success
  - Patterns specify a context in which they should be used; if your context does not match, then the pattern may fail
- Patterns offer new solutions to old problems
  - Patterns document "solutions that have worked in the past" for specific problems; thus, they document "tried-and-true" solutions rather than innovative or untested approaches

Parts of a Use Case Pattern

- Pattern Name
  - provides vocabulary
- Context
  - preconditions
- Problem Statement
  - what happens if the use case is not followed
- Metaphoric Story
  - case study to make the pattern easier to understand
- Forces Affecting the Problem
  - various factors that affect the problem and what trade-offs can be made between them
- The Solution
  - the technique used to solve the problem
- Examples
  - Demonstrates benefits of following pattern or consequences if you don't

Types of Use Case Patterns

- Adolph and Bramble have defined 31 use case patterns of two particular types
  - Development Patterns
    - Team Organization - use case team
    - Process - process used to write use cases
    - Editing - how to evolve existing use cases
  - Structural Patterns
    - Use case sets - involving collections of use cases
    - Use cases - involving individual use cases
    - Scenarios and steps - involving action steps
    - Use case relationships - «include», «extend», etc.

Just to give you a feel…

- SmallWritingTeam
- ParticipatingAudience
- BalancedTeam
- BreadthBeforeDepth
- SpiralDevelopment
- TwoTierReview
- QuittingTime
- RedistributeTheWealth
- CleanHouse
- CommonSubBehavior
- ActorIntentAccomplished
- InterruptsAsExtensions
- PromotedAlternative
- SharedClearVision
- VisibleBoundary
- ClearCastOfCharacters
- UserValuedTransactions
- EverUnfoldingStory
- CompleteSingleGoal
- VerbPhraseName
- ScenarioPlusFragments
- Adornments

Pattern Overview

- The Team (D)
  - SmallWritingTeam
- The Process (D)
  - BreadthBeforeDepth
  - SpiralDevelopment
  - QuittingTime
- The Use Case Set (S)
  - SharedClearVision
  - UserValuedTransactions
- The Use Case (S)
  - CompleteSingleGoal
  - VerbPhaseName
  - Adornments
- The Scenario (S)
  - LeveledSteps
- The Step (S)
  - ForwardProgress
- (S) - Structure
- (D) - Development

SmallWritingTeam

- Problem
  - Using too many people to write a use case is inefficient, and the compromises made to align the many different points of view may result in a less than satisfactory system
- Solution
  - Restrict the number of people refining any one work product to just two or three people. Use a TwoTierReview process to include more people
  - TwoTierReview says to hold two types of reviews
    - The first by a smaller team, possible held many times
    - the second by the complete group, perhaps just once

BreadthBeforeDepth

- Problem
  - You will not make timely progress or create coherent use cases if you waste energy writing detailed use cases sequentially
- Solution
  - Conserve your energy by developing an overview of your use cases first, then progressively add detail, working across a group of related use cases
    - Use the UML graphical notation for this process, since this notation only allows the specification of a use case name (and optionally extension points) within a use case oval

SpiralDevelopment

- Problem
  - Developing use cases in a single pass is difficult and can make it expensive to incorporate new information into them. Even worse, it can delay the discovery of risk factors
- Solution
  - Develop use cases in an iterative, breadth-first manner, with each iteration progressively increasing the precision and accuracy of the use case set
- One Approach
  - List actors and goals first; pause
  - Select subset and develop success scenarios; iterate perhaps adding actors, goals, and new use cases
  - Then, select subset and develop extensions, etc.

QuittingTime

- Problem
  - Developing a use case model beyond the needs of the stakeholders and developers wastes resources and delays the project
- Solution
  - Stop developing use cases once they are complete and satisfactorily meet audience needs
    - To determine if your use cases are “complete”:
      - 1. Have you identified and documented all actors/goals?
      - 2. Does the customer think the set is complete?
      - 3. Can your designers implement these use cases?

SharedClearVision

- Problem
  - The lack of a clear vision about a system can lead to indecision and contrary opinions among the stakeholders and can quickly paralyze the project
- Solution
  - Prepare a statement of purpose for the system that clearly describes the objectives of the system and supports the mission of the organization. Distribute widely.
  - In the “vision statement,” describe objectives, problems the system will solve, problems the system will NOT solve, the stakeholders, and the benefits provided to the stakeholders

UserValuedTransactions

- Problem
  - A system is deficient if it cannot deliver services that are valuable to its users and it does not support the goals and objectives specified by the SharedClearVision
- Solution
  - Identify the valuable services that the system delivers to the actors to satisfy their business purposes
- Leads to use cases like “Hire Employee” rather than “Create Employee Record”

CompleteSingleGoal

- Problem
  - Improper goals will leave writers uncertain about where one use case ends and another begins
- Solution
  - Write each use case to address one complete and well-defined goal.
- Example
  - Change Seat
    - In an airline setting, this could refer to exchanging a seat or upgrading a seat; better to make the goal more clear

VerbPhraseName

- Problem
  - Meaningless, generic names will not set reader expectations or provide a convenient reference point
    - Names should convey meaning
- Solution
  - Name the use case with an active verb phrase that represents the goal of the primary actor
- Bad Examples
  - Main Use Case, Claim Process, Use Case 2
- Better Examples
  - File Accident Claim, Approve Property Damage Claim

Adornments

- Problem
  - The inclusion of non-functional requirements in a use case can quickly clutter and obscure the details of a use case
- Solution
  - Create additional fields in the use case template that are outside the scenario text to hold supplementary information
- Example: Do not place conditional logic in a scenario that captures business rules. For example, “Is a user eligible for a seat upgrade?”
  - Simply assume that they are, then place rules for “eligibility” in a separate field of the use case

LeveledSteps

- Problem
  - Excessively large or excessively small use case steps obscure the goal and make the use case difficult to read and comprehend
    - Imagine describing the action of stepping onto a sidewalk in smaller and smaller steps
- Solution
  - Keep scenarios to three to nine steps; Ideally, the steps are all at similar levels and at a level of abstraction just below the use case goal
  - Examples will be presented in class

ForwardProgress

- Problem
  - Writers have to decide how much behavior to put into any one step. They can easily write too much detail, making the use case long and tiring to read
- Solution
  - Eliminate or merge steps that do not advance the actor. Simplify passages that distract the reader from this progress
  - Examples will be presented in class

What’s Next?

- More details about the analysis phase
  - How to find candidate classes
    - CRC cards will be used as an example
  - How to find relationships between classes
  - Advanced UML notations for classes, relationships, etc.
  - Problems encountered during analysis
- Then, the midterm…stay tuned!