Lecture 26: OO Design Methods: Mathiassen, Part 6

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2001

Goals of Lecture

- Cover Mathiassen's method for component design (e.g. low-level design)
- Activities
  - Model Component
  - Function Component
  - Connecting Components

Component Design

- Purpose
  - To determine an implementation of requirements within an architectural framework
- Definitions
  - Component: A collection of program parts that constitutes a whole and has well-defined responsibilities
  - Connection: The implementation of a dependency relation
- Principles
  - Respect the component architecture
  - Adapt component designs to the technical possibilities
- Results
  - A description of the system's components

Component Design

- Last step before implementation
- Previous Steps
  - Problem Domain Analysis
  - Application Domain Analysis
  - Architectural Design
- Meaning of First Principle
  - Do not change the architectural design for short term advantage

Component Design

- Input
  – Architectural Specifications
- Steps (Page 232)
  – Design Components
    - Both Model and Function
  – Design Component Connections
- Output
  – Component Specifications

Model Component Design

- Purpose
  – To represent a model of a problem domain
- Definitions
  – Model Component: A part of the system that implements the problem domain model
  – Attribute: A descriptive property of a class or an event
- Principles
  – Represent Events as classes, structures, and attributes
  – Choose the simplest representation of events
- Results
  – A class diagram of the model component;
  note: component ≠ class

Background

- Central Concept: Structure
  – Model Components should reflect structure of problem domain's relevant conceptual relations
- Foundation
  – OO Model of Problem Domain Analysis
- Main Task
  – Represent problem domain events using mechanisms of OO programming languages
- Results
  – Revised problem-domain class diagram

Designing the Model Component

- Input
  – Class Diagram
  – Behavioral Patterns
  – Component Specs from Arch. Design
- Steps (page 239)
  – Represent Private Events
  – Represent Common Events
  – Restructure Classes
- Output
  – Model Component Specification
- Example: Figure 12.1, 12.2, 12.4

Background

- Key concept of problem-domain analysis returns
  - Events! (Event Tables guide process of model component design)
- Events
  - are grounded in problem domain
  - have attributes
  - cause model updates when they occur
- Behavioral Patterns
  - Specify legal traces of events
- Method: Use behavioral patterns to determine information the model components must capture

Step 1: Represent Private Events

- Private Events involve only one problem domain object
  - Use Event Table to identify private events
- Use guidelines of figure 12.5 to modify problem-domain class diagram
  - Single events: store attributes in class
  - Multiple events: create new event class

Example: Customer Class

- Has two private events (Fig. 12.2)
  - Credit Approval
    - Attributes: date, name, address
  - Change Address
    - Attributes: date, address
- Represent Events (Figure 12.6)
  - Credit Approval occurs once
    - Add attributes to customer class
  - Change Address can happen more than once
    - Create new class; each instance corresponds to one occurrence of the event

Step 2: Represent Common Events

- Common Events involve more than one problem-domain object
- Guidelines
  - Choose one object to represent the event
  - All other objects access event info via structural relationships
- Heuristic
  - Choose simplest structure
    - Use event table to guide you

Example: Customer and Account

- Open Account and Close Account
  - Occur only once for each account
  - Occur multiple times for Customer
  - Simplest Representation
    - Attributes on Account Object
      - account state, opendate, closedate
- Deposit and Withdraw
  - Occur multiple times for both customer and account
  - Need to evaluate multiple options and choose simplest structure; see Figure 12.9

Step 3: Restructure Classes

- Simplify Revised Class Diagram
  - Generalization (Figure 12.10)
    - Multiple classes might be replaced a common superclass
  - Association (Figure 12.11)
    - Some associations may be obsolete
  - Embedded Iterations (Figure 12.12 and 12.13)
    - Simple analysis models may not specify enough information to produce correct designs

Function Component Design

- Purpose
  - To determine the implementation of functions
- Definitions
  - Function Component: A part of a system that implements functional requirements
  - Operation: A process property specified in a class and activated through class objects
- Principles
  - Base the design on function types
  - Specify Complex Operations
- Results
  - A class diagram with operations and specifications of complex operations

Background

- Behavior in OO systems is described as operations on a system’s classes
  - Behavior is activated by invoking these operations that reside within objects
- Since an OO system’s interactions constitute its behavior, and functions are used to enable interactions, functions must be implemented by operations

Model Component Design

- Inputs
  - Function List, Class Diagram, Component Specs
  - Model Component Specs
- Steps (page 252)
  - Design functions as operations
    - Design not implement! Simple operations first!
  - Explore patterns
  - Specify Complex Operations
- Results
  - Modified Model Components, Function Component Specs

Step 1: Design Functions as Operations

- Design functions based on type
  - Update, Read, Compute, and Signal
- Figure 13.3 provides guidelines for each type
- In general, sequence diagrams can be used to specify operations
  - Note: I don’t like the diagrams (Figures 13.4-13.7) presented by Mathiassen in this section because they do not show legal UML

Step 2: Explore Patterns

- Model-Class Placement (Figure 13.8)
  - Operations are best placed in a model-component class with compatible attributes and operations
- Function-Class Placement (Figure 13.9)
  - If an operation involves objects from different model components, then it must be placed in a function component
- Strategy (Figure 13.10)
  - Useful in designing an operation that might be implemented in multiple ways; allows dynamic change of the operation at run-time
- Active Function (Figure 13.11)
  - Active functions reside in Active Objects

Step 3: Specify Complex Operations

- Operations can be specified in a number of ways
  - Textually (Figure 13.12)
  - Graphically
    - Sequence Diagrams
    - State Chart Diagrams
- A system’s total behavior can be represented using state charts

Connecting Components Activity

- Purpose
  - To connect system components
- Definitions
  - Coupling: A measure of how closely two classes or components are connected
  - Cohesion: A measure of how well a class or component is tied together
- Principle
  - Highly cohesive classes and loosely coupled components
- Results
  - Class Diagram

Coupling

- A negative measure, we wish to minimize it
- Four types
  - Outside coupling: Class A makes use of the public aspects of Class B
  - Inside Coupling: Operation A refers directly to private properties of its host class
  - Coupling from below: A subclass refers to private properties of its superclass
  - Sideways Coupling: A class refers directly to private properties in some other class
- Low coupling can be achieved by using outside coupling and avoiding sideways coupling

Cohesion

- A positive measure, we try to maximize it
- Properties of Class Cohesion
  - Operations constitute a functional whole
  - Attributes and object structures describe objects with well-defined states
  - Operations use each other
- Properties of Component Cohesion
  - Component classes are conceptually related
  - Structural relations among classes are primarily generalizations and aggregations
  - Key operations can be carried out within component

Connecting Components

- Input
  - Class diagram and Component Specs
- Steps (page 274)
  - Connect Classes
  - Explore Patterns
  - Evaluate Connections
- Output
  - Class diagrams and component specs

Step 1: Connect Classes

- Three types of component connections
  - Aggregating another component's classes (Figure 14.2)
  - Specializing another component's public class (Figure 14.3)
  - Calling public operations in another component's objects (Figure 14.4)
- The call connection is preferred

Step 2: Explore Patterns

- Observer
  - Basic Structure (Figure 14.5)
    - Abstract subject and observer
    - Concrete subject and observers
  - Basic Pattern of Use (Figure 14.6)
  - Example of Use (Figure 14.7)

Step 3: Evaluate Connections

- Evaluate Connections to ensure low coupling is being achieved
- Figure 14.8 presents a checklist of concerns for each type of coupling