Lecture 25: OO Design Methods: Mathiassen, Part 5

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2001

Goals of Lecture

- Cover Mathiassen's method for architectural design (e.g. high-level design)
- Activities
  - Criteria
  - Components
  - Processes

Architectural Design

- Otherwise known as high-level design
  - What are the subsystems?
    - What are their interfaces?
    - What are their components?
  - How are they arranged?
    - What processes does the system support?
- Purpose
  - To structure a software system

Definitions & Principles

- Definitions
  - Criterion
    - A preferred property of an architecture
  - Component Architecture
    - A system structure composed of interconnected components
  - Process Architecture
    - A system execution structure composed of interdependent processes
- Principles
  - Define and prioritize criteria
  - Bridge criteria and technical platform
  - Evaluate designs early

Component and Process Architecture

- Component Architecture focuses on the stable aspects of a system
  - Mathiassen identifies classes as the stable element in his method
- Process Architecture focuses on the dynamic aspects of a system
  - Mathiassen identifies objects as the dynamic element in his method
- See page 174

Architectural Design

- Inputs
  - Results of Analysis
    - Problem Domain and Application Domain
- Steps (page 176)
  - Define Criteria for the Design
  - Design a component architecture
  - Design a process architecture
- Outputs
  - Architectural Specification

The Criteria Step

- Purpose
  - To set design priorities
- Definitions
  - Criterion: A preferred property of an architecture
  - Conditions: The technical, organizational, and human opportunities and limits involved in performing a task
- Principles
  - A good design has no major weaknesses
  - A good design balances several criteria
  - A good design is usable, flexible, and comprehensible

More on the Criteria Step

- Inputs
  - System Definition
- Steps
  - Consider General Criteria
  - Analyze Specific Conditions
  - Prioritize
- Outputs
  - Criteria for Design

Classical Criterion

- Usable
- Secure
- Efficient
- Correct
- Reliable
- Maintainable
- Testable
- Flexible
- Comprehensible
- Reusable
- Portable
- Interoperable

Step 1: Consider General Criteria

- Mathiassen focus on three criteria in particular (because they have universal validity)
  - Usable
    - Does the design satisfy users' needs?
    - Does the design fit the technical platform?
  - Flexibility
    - Modularity is a critical tool (Lego example, pg. 181)
  - Comprehensibility
    - abstraction is a key tool
    - design patterns (learn pattern once; use it many times)

Step 2: Analyze Specific Conditions

- The conditions of the environment that the system will be placed in, influence design
  - Credit Card System, page 182-183
    - Criteria: Security, Scalability, Performance
- Traditional conditions
  - Technical, Organizational, Human
  - figure 9.3, page 184

Step 3: Prioritize

- After you have identified the criteria important for your system, you must arrange them according to priority
- Figure 9.4 shows one form that can be used to help this process (page 185)

The Component Step

- Purpose
  - To create a comprehensible and flexible system structure
- Definitions
  - Component Architecture: A system structure of interconnected components
  - Component: A collection of program parts (classes) that constitutes a whole and has well-defined responsibilities
- Principles
  - Reduce complexity by separating concerns
  - Reflect stable context structures
  - Reuse existing components

More on Principles

- Reduce complexity by separating concerns
  - Separate components should address separate concerns; increase comprehensibility and flexibility
- Reflect stable context structures
  - Architectural design attempts to bridge requirements to technical options
    - Therefore the architecture must have a sound relationship to a system's context; which we identified during analysis; therefore our architecture should reflect the structures identified in analysis (UI Example, page 191)
- Reuse existing components
  - From analysis and from architectural patterns

The Component Step

- Inputs
  - Criteria (and results of analysis)
- Steps (page 192)
  - Explore architectural patterns
  - Define subsystems
  - Identify components (create class diagram)
  - Specify complex components
- Outputs
  - Component Specification

Step 1: Explore Architectural Patterns

- The Layered Architecture Pattern
  - Pages 193 and 194
- The Generic Architecture Pattern
  - Page 196
- The Client-Server Pattern
  - Page 197

Step 2: Define Subsystems

- Large systems need to be divided into subsystems
  - Think of it as partitioning the interface, model, and functions of the whole system into logical parts
  - Page 198 and 199
- Clients and Servers can be thought of as subsystems; different partitions of interface, model, and function lead to different types of client-server systems (See page 200 and 201)

Step 3: Identify Components

- Figure 10.11 lists design concerns for identifying components that deal with issues of model, function, and interface
  - Model components are tied to the problem domain; if an event occurs in the problem domain, some model component must change state
  - Function components provide the functionality required by the model
  - Interface components facilitate interactions between actors and the system
- Consider using existing components and/or extending the technical platform with new components (e.g. creating a new widget)

Step 4: Specify Relevant Components

- Mathiassen's recommendations are not too useful!
  - See figures 10.13 and 10.14 on page 206
- In general, the discussion from section 7.3 applies
  - again we are identifying components, not specifying them
    - we will specify details in low-level design

The Process Step

- Purpose
  - To define the physical structuring of a system
- Definitions
  - Process Architecture: A system-execution structure composed of interdependent processes
  - Processor: A piece of equipment that can execute a program
  - Program Component: A physical module of program code
  - Active Object: An object that has been assigned a process
- Principles
  - Aim at an architecture without bottlenecks
  - Distribute components on processors
  - Coordinate resource sharing with active objects

Background

- The process architecture brings us closer to the system's physical level
  - Our goal is to produce a deployment diagram that shows how our system's components will be distributed across the processors in the environment
- The process step is structured according to two levels of abstraction
  - overall distribution of components
  - processes that facilitate collaboration among objects

The Process Step

- Inputs
  - Class Diagram and Component Specs.
- Steps (page 212)
  - Explore Distribution Patterns
  - Distribute Program Components
  - Identify Shared Resources
  - Explore coordination patterns
  - Select Coordination Mechanisms
- Output
  - Deployment Diagram (page 210)

Step 1: Explore Distribution Patterns

- Mathiassen presents three patterns related to client-server systems
  - Centralized (page 216)
  - Distributed (page 217)
  - Decentralized (page 219)

Step 2: Distribute Program Components

- Begins with output of the component step, with the goal being to distribute these components across all processors
  - Can be delayed until the component architecture and the components themselves are designed, or earlier when it has a chance to influence the components used
- Sub-steps
  - Step 1: Separate program components and active objects
    - Components with some active operations need to be split
  - Step 2: Determine Available Processors
  - Step 3: Distribute program components and active objects
    - Layered systems may all be on one processor
    - Client-Server systems will, of course, be distributed

Step 3: Identify Shared Resources

- Purpose
  - To identify bottlenecks which can arise from extensive or shared use of resources
    - Processor
      - Examine fine grain object interactions (Figure 11.8)
    - Program-Component Sharing
    - External-Device Sharing
- To find bottleneck, ask
  - Do the active objects assigned to a processor exceed it capacity?
  - What is the accessibility, capacity, and load of the shared external devices?
  - Where is model information stored? How is it accessed?
  - What is the capacity and load of the system's (architectural) connections?
- In response, you must either change design or modify hardware

Step 4: Explore Coordination Patterns

- Two primary mechanisms
  - synchronization
  - data exchange
- Patterns
  - dedicated monitor
  - centralized task dispatcher
  - subscription to state changes
  - asynchronous data exchange

Step 5: Select Coordination Mechanisms

- For each shared resource, consider the use of an active object to coordinate access to the resource