Lecture 19: OO Design Methods: Mathiassen, Part 1

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

Goals of Lecture

• Begin to examine the OO Analysis and Design Method described in the Mathiassen, et al., textbook
• In particular, we will look at their take on problem domain analysis
  – This involves classes, structure, and behavior

Problem Domain Analysis

• In Mathiassen
  – Begin with a class activity
    • identify a candidate set of classes and events
  – Followed by a structure activity
    • define the structural relationships between classes
  – End with a behavior activity
    • specify the behavior of each class

Classes

• The Mathiassen method begins problem domain analysis using classes
  – Trying to answer the question
    • Which objects and events should we include in the model and which should we leave out?
  – Steps
    • We abstract problem domain phenomena by seeing them as objects and events
    • We classify objects and events and select which classes and events the system will maintain information on
  • Classes are the first means to define and limit a problem domain; e.g. Mathiassen uses classes as designations; a means for setting our scope
Classification

• Classify objects in the problem domain
  – Challenges
    • Formalizing Existing Concepts
      – Humans may use a term, such as “course”, to refer to many different things; we may need to disambiguate between “course”, “seminar”, and “lab”
    • Different Interpretations
      – In a business context, accounting, production, and sales may all use the term “order” to mean different things
  – Approach: identify phenomena as objects, classify these objects and the events they can produce

Objects and Events

• For Mathiassen, objects are entities with identity, state, and behavior
• Objects are characterized by events
  – events are defined as “instantaneous incidents involving one or more objects”
• Example
  – Bank customer
  – Possible Events: Deposit, Withdraw, Apply for Loan, Buy Bonds, etc.

More on Events

• An event is an abstraction of an activity
  – We can abstract the activity of a “withdrawal” in order to describe the behavior of a bank customer
• Activities have duration, events do not
  – “loan approved”
• Events tie objects together
  – A “deposit” involves a customer and an account; the event is assigned to both objects
• Events have unique names (they live in a global namespace)

Problem Domain Analysis, cont.

• Having identified a set of objects
  – We find a set of classes to model them
  – Mathiassen recommends brainstorming as many different classes as possible, at first
    • you will later evaluate this list to identify the core set of classes that will be needed to model the system
  – Mathiassen also recommends that this process be performed with the user
Generating Potential Classes

• nouns
  – and noun phrases
  – as given by users
• general types
  – physical things
  – people and roles
  – organizations
  – places, concepts
  – descriptions
  – resources
  – devices
  – systems

  • Remember
    – brainstorm
    – do not (yet) evaluate
  • Chose
    – simple names
    – that originate in the problem domain
    – indicate a single instance

Problem Domain Analysis, cont.

• Having a set of potential classes
  – We now must identify events
  – Start with the verbs that your users use
  – Draw on general event types
    • work and production, transportation, consumption, life cycle, career and education, contracting and exchange, monitoring and control, planning and management, decision making and communication
  – Choose event names that are simple, originate in the problem domain and indicate a single event

Beware Verb Tense

• In choosing the verb form for event names, Mathiassen identifies three choices
  – present tense, past tense, present participle
    • reserve, reserved, reserving
  – Potential Problems
    • present tense verbs are difficult to distinguish from method names
    • past tense verbs are difficult to distinguish from the state reached after the event has occurred
    • the third form contradicts the fundamental property of an event as being instantaneous

Problem Domain Analysis, cont.

• Evaluate classes/events systematically
  – General evaluation criteria
    • Is the class or event within the system definition
    • Is the class or event relevant for the problem domain
    • «Note the similarity to the principle of domain relevance»
  – Classes and Events should concern only the problem domain at this point, not the application domain
Evaluation Criteria for Classes

• Can you identify objects from the class?
  – Is there a recognition rule?
• Does the class contain unique information?
  – If the class contains information that can be derived from other classes, then you are modeling functionality and not classes
• Does the class encompass multiple objects?
  – Singleton classes are rare
• Does the class have a suitable and manageable number of events?
  – A class with few events may be too simple; too many events and it may be better to split the class into smaller, more simple, classes

Evaluation Criteria for Events

• Is the event instantaneous?
  – If you want to model multiple events throughout an activity, include start, stop, and interval events; we want to know that an event has occurred
• Is the event atomic?
  – If you have an event that can be broken down into sub-events; include the sub-events directly and discard the composite event
• Can the event be identified when it occurs?
  – Would you be able to implement a system that can observe the event?

Assigning Events to Classes

• The class activity ends by creating an event table that relates events to classes; see Figure 3.1 on page 50
• Guidelines for creating the event table
  – Which events is this class involved in?
  – What classes are involved in this event?
• Effective for evaluating the cohesion and coupling of your classes and events

Structure Activity

• Goal
  – Produce a class diagram
• Purpose
  – Model abstract, general relationships between classes and concrete, specific relationships between objects
• Benefit
  – provides a coherent problem domain overview by describing all structural relations between classes and objects in our model
Starting the structure activity

• What are the specific relations between objects in the problem domain?
  – Identify two types of object relationships
    • aggregation structures
    • associations
  • What is the conceptual relationships between two or more classes in the problem domain?
    – Identify two types of class relationships
      • generalization
      • clusters (e.g. a collection of related classes)

Important Point

• Class structures are static, conceptual relationships
  – they do not change, unless we somehow change the class descriptions themselves
• Object structures are concrete, dynamic relationships
  – They can freely change at runtime without impacting our class description

Steps of the Structure Activity

• Find candidate structures
• Evaluate Patterns
• Evaluate candidate structures and select the relevant relationships
• (See Figure 4.3 on page 72)
• Note: this process is iterative and may require backtracking to the class activity

Find Candidate Structures

• Find “is-a” relationships
  – A taxi is a car…
  – Remember that subclasses are mutually exclusive
• Find clusters
  – A cluster is a collection of classes that helps us achieve a problem-domain overview
  – See, for example, page 75
• Find associations and aggregations between objects; specify as class relationships
More on clusters

• Clusters (denoted using a folder symbol) enable an understanding of the problem domain by breaking it down into sub-domains
• Within a cluster, classes are related using generalization and aggregation
  – between clusters, classes are related using associations

Identifying Generalizations

• Approach 1
  – Examine every pair of selected classes and determine if a generalization structure exists
    • if so, the superclass’s events must be a subset of the subclasses’s events
• Approach 2
  – Determine if a relevant generalization exists for pairs of selected classes
    • this may introduce new classes
• Approach 3
  – examine each class and attempt to define a relevant generalization or specialization; may also add new classes

Identify Aggregations

• Approach One
  – Examine each pair of classes to see if the objects of one are decompositions of the objects of the other
• Approach Two
  – Determine if it is relevant to aggregate the objects from each pair of classes into objects from a newly created class
• Approach Three
  – Examine each class to see if new classes can be added that represent relevant “parts” or “wholes”

Types of Aggregation

• Whole-Part
  – the whole is the sum of the parts; if we add or remove any part, we change the whole fundamentally
    – delete the whole; delete the parts
• Container-Content
  – the whole is a container for the parts; the whole does not change fundamentally if we add or remove parts
• Union-Member
  – the whole is an organized union of members; similar to container-content except there is a lower bound on the number of members
Identify Associations/Clusters

- Add associations whenever you need to administrate, monitor, or control relations between objects that are not otherwise related
- Add clusters to identify specific sub-domains
  - Note: classes cannot belong to more than one cluster

Explore Patterns

- Object Oriented Patterns are generalized descriptions of a problem and a related solution
  - We will cover patterns, in more detail, later in the semester
- For now, we look at four patterns particularly concerned with structure
  - Role, Relation, Hierarchy, Item-Descriptor

Role Pattern

- Used to model a situation where a single person can have multiple roles in a problem domain
- Solution: have a Person object aggregate one or more Role objects; each Role object can be a different subclass of Role

Relation Pattern

- A means for relating two objects, where the relation itself has properties
  - what we called association classes earlier
- A “party” to the relation aggregates a number of Relation objects; each Relation object is associated with some other “party”
Hierarchy Pattern

- Used to organize elements into a series of layers
- Have each layer, aggregate instances of the layer below it; the bottom layer is some relevant element

```
Level N --*-- Level N-1 --*-- ... --*-- Level 1
```

Item-Descriptor Pattern

- Helps to distinguish between an item and its description
  - books and copies
  - each copy has its own identity; but shares properties described by the book object

```
Descriptor --*-- Item
```

Evaluate Systematically

- Principle
  - Model only the necessary structural relationships

- Criteria
  - Structures must be used correctly
  - Structures must be conceptually true
    - Do the structures represent the problem domain for a future user of the system
  - Structures must be simple

Structures Must be Used Correctly

- Do not mix generalization and aggregation
  - “is-a” versus “has-a” and “is-part-of”
- Use aggregation to capture fundamental, definitive relations; use associations for more fluid relations
  - Can the objects exist independently of each other?
  - Are the objects equally ranked?
  - Can the connection from an object from the one class change to other objects from the other class
- If you answer “yes” to two or more, use association
  - otherwise use aggregation