Lecture 20: OO Design Methods: Mathiassen, Part 2

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

Goals of Lecture

- Continue our look at Mathiassen's method for problem domain analysis
- Review of Steps
  - Class Activity (last lecture)
  - Structure Activity (last lecture)
  - Behavior Activity (this lecture)

Motivation

- In problem domain analysis, we need to understand what happens in a problem domain over time
- Our system's fundamental purpose (according to Mathiassen) is to register, store, and produce information about problem domain events

The Behavior Activity

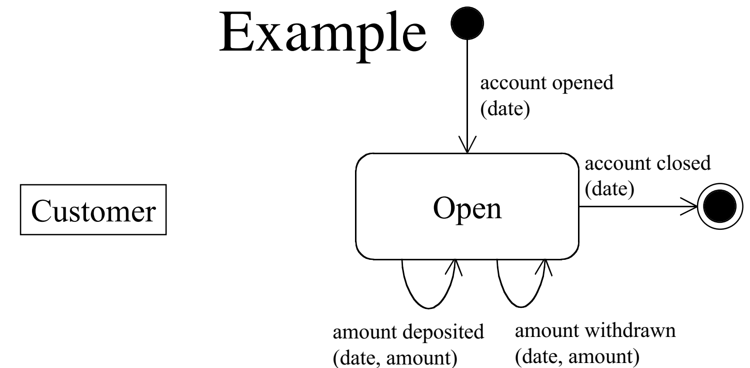
- The behavior activity extends
  - the class definitions of the class and structure activities
  - with information about attributes and (what Mathiassen calls) the behavioral pattern for each class
- The behavioral pattern allows us to specify the possible event traces for an object; we need to do this, since some problem domains impose limitations on the order of events

Definitions

- Event Trace: A sequence of events involving a specific object
- Behavioral Pattern: A description of possible event traces for all objects in a class
  - e.g. a state diagram!

© Kenneth M. Anderson, 2001

Example



This behavioral pattern asserts that a Customer object is created when a real customer opens an account in the bank. The customer can then deposit and withdraw money. The customer object is deleted when the account is closed.

© Kenneth M. Anderson, 2001

Interesting Take on Attributes

- Mathiassen has an interesting take on determining class attributes
  - In particular, attributes are derived for a class by examining its behavioral pattern!
  - Attributes are the data that a system must store; use the events to which an object responds to determine the data it must store

© Kenneth M. Anderson, 2001

The Behavior Activity, in detail (page 92)

- Inputs
  - Event Table and Class Diagram
  - Design Patterns
- Steps
  - Create behavioral pattern for each class
  - Consider changes to class diagram; repeat
  - When done, assign attributes for each class
- Outputs
  - Modified class diagram and (possibly) event table
  - Behavioral Patterns with attributes

© Kenneth M. Anderson, 2001

Creating Behavioral Patterns

- Start by defining the first and last event in an object's life
  - If you do this for each object, you will end with a set of object creation events and object "disappearance" events
  - Not "created" but "account opened"
  - Why "disappearance", the system may still need to examine an object after it has "died" - so an object is not automatically deleted when its last event occurs

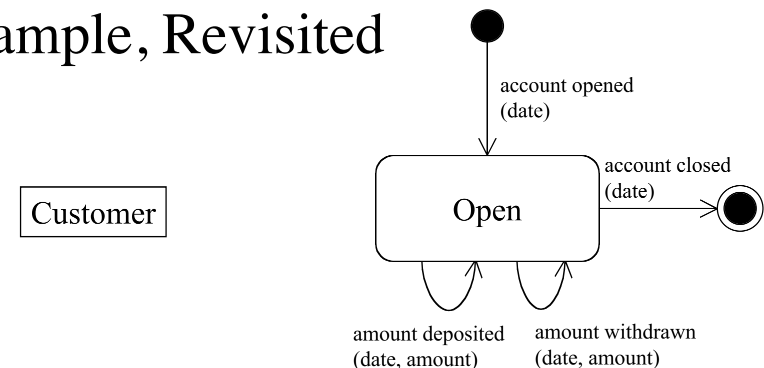
Creating Behavioral Patterns

- Continue by examining the event sequences for an object
  - Is the overall form structured or unstructured
    - structured is indicated by a sequence of events that occur in a specified order
    - unstructured is indicated by a collection of intermediate events that can occur in any order any number of times
  - Which events occur together in a sequence
  - Are there any alternative events?
  - Can a given event occur more than once?
- Use the answers to these questions to create the associated state diagrams

Evaluation Criteria

- The behavioral pattern should be sufficiently precise to describe all legal, and thus all illegal, event traces
- The behavioral pattern should provide an overview and thus be as simple as possible
- These criteria may be conflicting!
  - You can avoid this conflict by describing "typical" behavior in the "main" diagram and create additional diagrams to specify "specialized" behavior

Example, Revisited



What happens if the same customer re-opens an account at a later date? In the current state diagram, we imply that we will have to create a new customer object and thus potentially store two customer objects for the same "real world" customer. This implies the need for a new state "closed": see page 98 of Mathiassen for details

Updating your Event Table

- As you create behavioral patterns, you will typically learn more about your events
  – in particular, how often they might occur
  – be sure to update your event table
    - use a "*" to indicate an event that can occur zero or more times; a "+" indicates an event that can occur zero or one time
    - Note: I do not like this notation since it does not correspond to "standard" regular expression syntax; normally a "?" is used to indicate "zero or one"
    - Use whatever you like, just be sure to document your symbols!

© Kenneth M. Anderson, 2001

Inheritance of Behavioral Patterns

- The behavioral pattern of a super class is inherited by all of its subclasses
  – An individual subclass will typically expand the inherited behavior by adding new states and events unique to that class
  – Note: multiple inheritance can cause problems if two superclasses specify incompatible behaviors; we are safe with respect to events, however, since event names are unique

© Kenneth M. Anderson, 2001

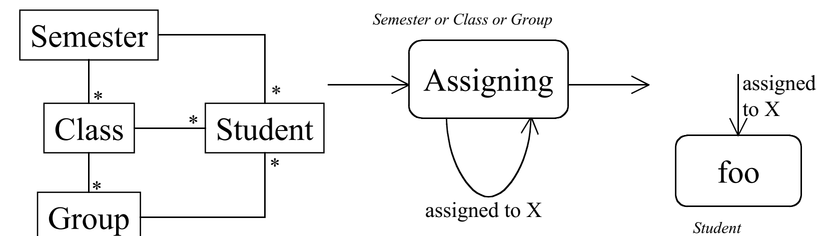
Explore Patterns

- Three Patterns arise with respect to behaviors
  – The Stepwise Relation Pattern
  – The Stepwise Role Pattern
  – The Composite Pattern

© Kenneth M. Anderson, 2001

Stepwise Relation Pattern

- Used to model situations where a problem-domain object is related to the elements of a hierarchy in a stepwise fashion (page 103)



© Kenneth M. Anderson, 2001

Stepwise Role Pattern

- Used to describe how the behavior of a whole changes as its parts become active
- It is stepwise since typically the final event for one part, is typically the first event for a subsequent part
- See page 104 for details

The Composite Pattern

- Used to describe the creation (and destruction) of a hierarchy whose detailed structure is unknown at design time
- Behavioral patterns are recursive
  - top level behavior requires some behavior beneath it
- See page 105

Changing the Class Diagram

- Aggregation and Association
  - If two or more objects have common events, consider adding an aggregation or association structure between them
- Flip Side
  - If two classes are related by an aggregation or association, at least one common event should be shared between them
  - e.g. the event that establishes or removes the link between the objects that participate in this structural relationship

Changing the Class Diagram

- Generalization
  - If the same event is tied to two classes, consider whether one class is a generalization of the other
  - If two classes, share many events, consider whether they are different specializations of a third class

Changing the Class Diagram

- Adding New Classes
  - In some cases, new classes will be suggested by "ambiguous" behavioral patterns
  - The behavioral pattern of the new class, removes the ambiguity by sharing the events and implying limitations to the legal event sequences
  - (See example page 109)

Describe Attributes

- Three classes of attributes
  - information connected to events that must be recorded by the system
    - date and amount of bank withdrawal
  - information related to the object as a whole
    - customer name and address
    - Note: treat attributes as atomic in analysis
      - e.g. customer name, not customer first and last name
  - attributes that can be derived from other attributes

Evaluation Criteria

- What are the general characteristics of the class?
- How is the class described in the problem domain?
- What basic data must be captured about objects from this class?
- What results from an event trace must be captured?

Another Interesting Take

- Mathiassen asserts that an attribute should only be included in your description if it is used by at least one system function
- However, system functions are not defined in problem domain analysis
- So, you may end problem domain analysis with attributes that will not make it through the next phase of Mathiassen's OO Design Method