Conceptual Modeling

A Short Discussion

Outline

- Conceptual modeling
  - The goal of conceptual modeling
- The OO solution
- The object model (conceptual)
  - Syntax and semantics
- Object modeling approach
  - Home Heating System

Conceptual Modeling

- Early modeling to understand the problem
- Conducted in cooperation with the customer
  - Domain experts
  - Domain engineers
- No real problem analysis if the customer is not involved
- Power of OO
  - It is simple and people can quickly participate effectively

The OO Solution

- The OO model closely resembles the problem domain
  - Base your model on the objects in the problem domain
- Iteratively refine the high-level model until you have an implementation
  - Attempt to avoid big conceptual jumps during the development process

Objects

- Person class
  - Person objects
    - Attributes: name, age, height, weight, move, change job
    - abstracts to Person class
- Card class
  - Card objects
    - Card objects
    - Attributes: height, weight, al-number
    - Operations: issue, change

Attributes and Operations
Characteristics of Objects

- Identity
  - Discrete and distinguishable entities
- Classification
  - Abstract entities with the same structure (attributes) and behavior (operations) into classes
- Polymorphism
  - The same operation may behave differently on different classes
- Inheritance
  - Sharing of attributes and operations based on a hierarchical relationship

Objects

- Something that makes sense in the application context (application domain)
  - J.O. Public
  - Joe’s Homework Assignment 1
  - J.J. Public’s driver’s license
- All objects have identity and are distinguishable
- NOT objects
  - Person
  - Drivers license

Classes

- Describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other classes, and common semantics
- Person
  - J.O. Public
  - Joe Smith
  - D.Q. Public
- Card
  - Credit card
  - Driver's license
  - Teller card

Class Diagrams

Objects have an identity
Do not explicitly list object identifiers
SSN OK!

Examples

Person
  - name: String
  - age: integer
  - height: integer
  - weight: integer
  - SSN: integer

Card
  - height: integer
  - width: integer
  - thickness: integer
  - id-number: integer
### Operations and Methods

- Transformation that can be applied to or performed by an object
- May have arguments

**Card**
- height: integer
- width: integer
- thickness: integer
- id-number: integer
- issue()
- revoke()

**Shape**
- height: integer
- width: integer
- rotate(angle: integer)
- move(x: integer, y: integer)

### Object Notation - Summary

**Class name**
- attribute-1 : data-type-1 = default-value-1
- attribute-2 : data-type-2 = default-value-2
- attribute-3 : data-type-3 = default-value-3

**operation-1(argument-list-1) : result-type-1**
**operation-2(argument-list-2) : result-type-2**
**operation-3(argument-list-3) : result-type-3**

### Associations

- Conceptual connection between classes
  - A credit card is issued by a bank
  - A person works for a company

**Class diagrams**

**Instance diagram**

### Associations are Bi-directional

- There is no direction implied in an association

**Country**
- name

**Has-capital**
- City
  - name

**Person**
- name

**Issued-by**
- Drivers-license
  - lic.-number: integer

### Associations Have Direction

- Unified adds a direction indicator
  - Inconsistently used

**Country**
- name

**Has-capital**
- City
  - name

**Person**
- name

**Issued-by**
- Drivers-license
  - lic.-number: integer

### Multiplicity

- One person holds one credit card
- One object can be related to many objects through the same association

One person can hold zero or more credit cards (* stands for many)
One person can hold zero or more credit cards (0..*)
Each card has zero or one holder (0..1)

Explicit enumeration is also possible (2, 3, 2..5, etc.)

Higher order associations

- Ternary association
  - Project, language, person
  - Seldom needed (and should be avoided)

Link Attributes

- Associations can have properties the same way objects have properties

Why not this?

Salary and job title are properties of the job not
the person

Another Approach

In this case, a link attribute is the only
solution
Role Names

- Attach names to the ends of an association to clarify its meaning

```
Person
  name: String
  address: String
  boss

Manager
  name: String
  employee

Company
  name: String
  address: String
  employee

salary: integer
job-title: String

Aggregation symbol
```

Aggregation

- A special association, the is-part-of association
  - A sentence is part of a paragraph (a paragraph consists of sentences)
  - A paragraph is part of a document (a document consists of paragraphs)

```
Document
  1..*  Paragraph

Paragraph
  1..*  Sentence

Aggregation symbol
```

Aggregation (Cont.)

- Often used in parts explosion

```
Car
  Wheel
  Body
  Gearbox
  Engine

Door
  Hood
  Trunk
  Piston
  Valve
  Crankshaft

Aggregation symbol
```

Generalization and Inheritance

- The is-a association
  - Cards have many properties in common
  - Generalize the common properties to a separate class, the base-card
  - Let all cards inherit from this class, all cards is a base-card (plus possibly something more)

```
Card
  height: integer
  length: integer
  address: String
  expiration: Date
  issued: Date
  revoked: boolean

Drivers License
  issued: Date
  expires: Date
  expire():

ID Card
  credit-limit: integer
  issued: Date
  expire():

Credit Card
  credit-limit: integer
  issued: Date
  expire():
```

Example

```
Game
  name: String
  players

Player
  name: String
  team

Team
  name: String
  games

Aggregation symbol
```

Aggregation Versus Association

- Can you use the phrase is-part-of or is-made-of
- Are operations automatically applied to the parts (for example, move) - aggregation
- Not clear what it should be......
### Aggregation Versus Inheritance

- Do not confuse the is-a relation (inheritance) with the is-part-of relation (aggregation)
- Use inheritance for special cases of a general concept
- Use aggregation for parts explosion

### Recursive Aggregates

- A recursive aggregate contains (directly or indirectly) an instance of the same kind of aggregate

### Object Modeling Summary

- Classes
  - Name
  - Attributes
  - Operations
  - Associations
    - Roles
    - Link attributes
- Aggregation
- Inheritance

### Object Modeling Approach

- Start with a problem statement
  - High-level requirements
- Define object model
  - Identify objects and classes
  - Prepare data dictionary
  - Identify associations and aggregations
  - Identify attributes of objects and links
  - Organize and simplify using inheritance
  - Iterate and refine the model
  - Group classes into modules

### The Home Heating System
Home Heating Requirements

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software should maintain the temperature within a specified range by controlling the heat flow to individual rooms.

- The software shall control the bang switch.
- The rooms shall be heated when the temperature is below the desired temperature.
- The rooms shall no longer be heated when the temperature is above the desired temperature.
- The flow of heat to each room shall be individually controlled by opening and closing in bang switches.
- The valve shall open when the room needs heat and close otherwise.
- The flow of heat shall be able to turn the heating system on or off.
- The user shall set the desired temperature on the thermostat.
- The operator shall be able to turn the heating system on and off.
- The furnace must not run when the system is off.
- When the furnace is not running and a room needs heat, the software shall turn the furnace on.
- To turn the furnace on, the software shall follow these steps:
  - Open the fuel valve.
  - Turn the burner on.
- To turn the furnace off, the software shall follow these steps:
  - Close the fuel valve.
  - Turn the burner off.

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

Identify Object Classes

Eliminate Bad Classes

- Redundant classes
  - Classes that represent the same thing with different words.
- Irrelevant classes
  - Classes we simply do not care about.
- Vague classes
  - Classes with ill-defined boundaries.
- Attributes
  - Things that describe individual objects.

- Operations
  - Sequences of actions are often mistaken for classes.
- Roles
  - The name of a class should reflect what it is, not the role it plays.
- Implementation details
  - Save that for implementation.

Classes After Elimination

Prepare Data Dictionary

- Water Tank
  - The storage tank containing the water that circulates in the system.
- Pump-1
  - The pump pumping water from the Water Tank to the radiators in the rooms.
Possible Associations

- Not much information from the prose requirements
- A lot of information from the system design

Object Model

- A room consists of a thermometer and a radiator
- A radiator consists of a valve and a radiator element
- The home heating system consists of a furnace, room, a water pump, a control panel, and a controller
- The furnace consists of a fuel pump and a burner
- The control panel consists of an on-off switch and a thermostat
- The controller controls the water pump
- The controller controls the radiator
- The controller monitors the temperature in each room
- The controller opens and closes the valves in the rooms
- The operator sets the desired temperature
- The operator turns the system on and off
- The controller gets notified of the new desired temperature

Attributes

- Temp Sensor
- Temp Sensor Temperature

Final Object Model
Iterate the Model

- Keep on doing this until you, your customer, and your engineers are happy with the model.

Operation vs Method

- **Operation**: specifies object behavior
- **Operations** of class are public **services** offered by class.
- **Service**: represented by set of operations.
- **Message**: object requests execution of an operation from another object by sending it message.
- **Method**: message is matched up with method defined by the class to which the receiving object belongs (or any of its superclasses)
- **Methods** of its classes are the implementations of these operations.