Analysis: Filling Out the Object Model

Overview

- Remember that the analysis *object model* attempts to describe not only object types, but also important attributes, operations, and associations
- This type of model will be much more detailed in the design phase
- In the analysis phase, only include attributes and operations that are relevant and important to the understanding of the problem at hand (i.e. the application domain)
- Now that we've seen techniques for identifying objects/classes, we fill out the object model by
    - Identifying associations
    - Identifying attributes
    - Identifying responsibilities (operations, attributes)
- We can, of course, use UML class diagrams as a notation for capturing this information and illustrating it clearly
- Analysis also includes building the dynamic model -- this will be discussed separately

Associations

As seen in the basic discussion of UML class diagrams, an *association* shows a relationship between two or more classes.

Properties of associations

- **name** -- an association can have a name, which describes the nature of the association
    - The name is optional, but is a good idea when the nature of the association is not clear
    - When the association is one of our special types -- aggregation or inheritance -- the name is usually not needed, as the nature of the association is clear already
    - With other general associations, the name is a good idea to include
- A **role** at each end
    - A role name identifies the purpose of the class at a given end of an association
    - Again, role names are best to include when the relationship is not clear
    - Aggregation and inheritance are, again, usually clear on their own
    - When there's only one association between a pair of distinct classes, the class names usually serve as good role names already (so no need to come up with new ones)
    - Good places to use role names:
        - An association between two objects of the same class type
        - Multiple associations between same pair of classes
- A **multiplicity** at each end
    - a multiplicity identifies the possible number of instances of one class relating to an instance of the class at the other end

    - Not usually needed with inheritance relationships
    - WIth aggregation, the assumed multiplicity at the diamond end would be 1 (unless specified otherwise)
    - *Qualification* can be used to reduce multiplicities (like "0 or more").
        - Many-to-many relationships are complex. Reducing multiplicities is a good idea
        - Qualification uses the concept of keys
        - Often, objects on a "many" side can be identified with a unique name (or key)
        - Use this to *qualify* the association -- reducing the multiplicity
        - Example: A Directory contains many Files. A File lives in one Directory. So there's a 1-to-many association between Directory and File. But each File in a Directory has a unique filename. So we can qualify the relationship by saying "Each possible filename in a Directory associates with at most one File. (And we still have 1 File exists in 1 Directory). So with qualification, it's a 1-to-1 relationship). See textbook page 57 for diagram

Inheritance associations

- With inheritance associations, the terms *generalization* and *specialization* are sometimes used
- *Generalization* is the activity that starts with specific classes and identifies commonalities that can be factored out (into base categories)
- *Specialization* is the activity that starts with a higher-level concept and identifies more specific concepts and subcategories from there
- Both activities can help identify appropriate places to set up inheritance relationships

Identiying Associations

- Examine verb phrases
- Add association and role names where appropriate
- Use qualifiers to reduce multiplicity and identify key attributes
- Eliminate redundant associations (i.e. too many paths between two classes
- Use generalization and specialization to identify appropriate inheritance relationships

Identifying Attributes

- Examine possessive phrases and adjectives
- Entity objects usually have specific information to be stored. Represent the stored state with attributes. Can ask the following:
    - How is the object described?
    - What parts of the description are relevant and applicable to the problem domain?
    - What is the *minimal* description needed?
- If an attribute is also being modeled as an object, don't list as an attribute -- use an aggregation relationship between classes instead

    - Is the independent existence of the attribute important? i.e. might it be manipulated as an entity by itself? Then make it an object
    - Or is only the value of the attribute important in the context of the object it's in? In this case, better to just make an attribute
    - Example: A Person object, with an address attribute. Will the address be an entity to be manipulated sometimes by itself? If so, make it an object, with an association to the Person. If you only manipulate the address in the context of the Person, make it an attribute. Can you give an example of a context for each of these situations?
- Remember, this is analysis, not design. Don't waste time on fine details until the overall structure is stable

Identifying Responsibilities

- This already being done when using the CRC card technique
- A responsibility is usually either what a class *knows* (data) or what a class *does* (behavior)
- Attributes qualify as responsibilities if the class' job is to store specific data (e.g. a message queue storing saved e-mail messages)
- For behavior, look for verbs in the requirements, use cases, and scenarios