Lecture 13: Advanced Analysis (Part 1)

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2003

Credit where Credit is Due

- Some material presented in this lecture is taken from section 5 of Maciaszek's "Requirements Analysis and System Design". © Addison Wesley, 2000

Goals for this Lecture

- Cover the material presented in Section 5 of the textbook
  - Introduce
    - Advanced Class Modeling
      - plus advanced UML notation for classes and associations
    - Class Layers

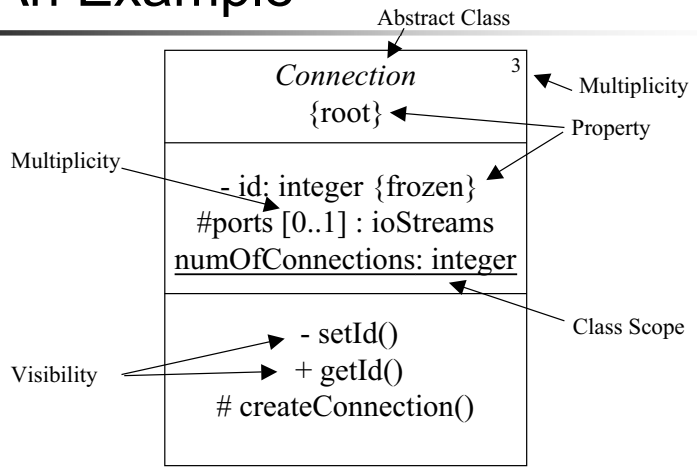
Advanced Class Modeling

- First, lets look at some advanced UML notations for classes and associations
- Then, we will review the material in section 5.1.1 to 5.1.5 of the Maciaszek textbook

Advanced UML Class Notations

- UML supports a number of advanced modeling features for classes
  - Class and Attribute Properties
  - Class and Attribute Multiplicity
  - Class-Scope Attributes and Operations
  - Visibility

An Example



Visibility

- The visibility of an attribute or operation specifies whether it can be used by other classes
- Three types
  - public (+) (This is the default)
    - Any outside class can access the feature (as long as it has a reference to the class)
  - protected (#)
    - Any descendant of the class can use the feature
  - private (-)
    - Only the host class can access the feature

Scope

- A feature (attribute or operation) can be assigned a scope
  - instance: each instance of a class has its own state for the feature
  - classifier (or class): There is only one value for this feature across all classes
    - numberOfConnections in the previous example
- Classifier scope is indicated by underlining the feature definition

Properties

- A class can be assigned two properties
  - root - the class can have no parents
  - leaf - the class can have no children
- A property is indicated by placing it below the class in brackets, e.g. {leaf}
  - attributes and operations can have properties too (covered later in this lecture)
- A class can also be abstract; which means that no instances can be created of this class
  - This is indicated by placing the class name in italics
  - This is used when the root class is meant to serve as a template for creating various subclasses

Multiplicity

- Class multiplicity constrains the number of instances that can be created for a class
  - The multiplicity for classes is indicated in the top, right corner of the class
- On attributes, it constrains the number of values an attribute can have
  - this lets you specify attributes that can be modeled as arrays: ports[2..*] : Port

Complete Attribute Syntax

- The complete syntax for attributes is
  [visibility] name [multiplicity] [: type]
  [= initial-value] [{property}]
- Example
  + ports [2..*] : Port = null {addOnly}
  id : integer = 0
- Attribute Property Values
  - *changeable*: default, freely modifiable
  - *addOnly*:may add new values;no changes allowed
  - *frozen*: the value may not change after the object is initialized

Complete Operation Syntax

- The complete syntax for operations is
  [visibility] name [(parameter-list)] [: return-type] [{property}]
- The complete syntax for a parameter is
  [direction] name : type [= default-value]
- Examples
  + set(n : Name, s : String) {sequential}
  - setId(inout id : integer)

Additional Operation Info

- Possible Direction Values
  - in : An input parameter; may not be modified
  - out : An output parameter; may be modified to communicate with caller
  - inout: An input parameter; may be modified
- Possible Operation Properties
  - isQuery: Does not change state of system
  - sequential: does not support multiple threads
  - guarded: does protect against multiple threads
  - concurrent: multiple threads can execute it at the same time

Associations

- Advanced adornments for associations include
  - navigation
  - visibility
  - qualification
- In addition, we will introduce/review the notions of (and the notations for)
  - association classes
  - association constraints

Association Navigation

- A direction can be added to an association

- in this example, you can navigate from objects of type User to objects of type Password but not the other way around

Association Visibility

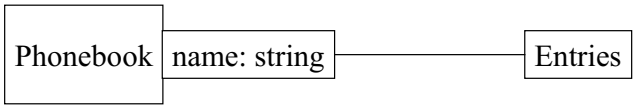
- Visibility can be assigned to an association role
  - public: objects outside the association can navigate the association
  - protected: only an object and its children can access a protected association
  - private: only the objects that participate in the association can navigate it; Below only instances of the `User` class can retrieve a `Password` via the `key` role name

Association Qualification

- Associations sometimes model relationships that involve “look up” or queries
  - That is, when navigating the relationship, you are looking for a particular object (or set of objects)
- Example
  - A phonebook consists of multiple entries
  - Given a name, we want to look up the associated phone number
- Note
  - Maciaszek talks about qualification in section 5.1.6

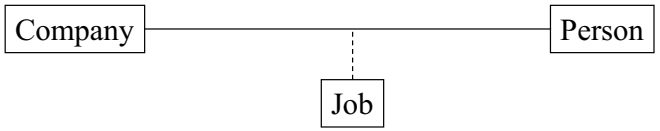
Association Qualification, cont.

- UML can model such a situation using an association qualification
  - the qualification is drawn as a rectangle extending out of its associated class
  - the rectangle contains the attributes used to perform the “look up”



Association Classes

- There are times when it becomes necessary to associate data with an association
  - Employment: should the details of a job be associated with a company or a person?
- Note: Maciaszek talks about some of the difficulties of association classes in 5.1.7
  - This information is not required for this class



Association Constraints

- UML provides five pre-defined association constraints
  - implicit: the relationship is conceptual
  - ordered: the set of objects at one end of the association are in an explicit order
  - changeable: links between objects can be modified freely
  - addOnly: new links may only be added; existing links may not change
  - frozen: a link, once added, cannot be modified
- Constraints are drawn in braces: {frozen}

Maciaszek, section 5.1.1

- Stereotypes
  - Generic extension mechanism in UML
  - Indicated with chevrons,«extend»
  - Used to change the semantics of an existing UML element
    - Such as labeling a class «Actor» to indicate that it came from the Use Case model

Maciaszek, section 5.1.2-5.1.3

- Constraints, Notes, and Tags
  - Used to indicate semantics that cannot be expressed by the UML notation
  - Indicated by curly brackets { } or the UML note symbol (page 159); Notes which indicate constraints should use the «constraint» stereotype
  - See examples of constraints on pages 157 to 159
  - A tag is any non-constraint textual information attached to the analysis model using curly brackets (see page 158)

Maciaszek, section 5.1.4

- Visibility
  - See slide 7
  - Maciaszek addresses some interesting issues with respect to visibility and inheritence and the notion of “friend”
    - most of these issues arise when your implementation language is C++
      - as such, I place less importance on these issues
      - if you develop in C++, be sure to study section 5.1.4 in depth
- For the most part, in analysis, you should apply the following heuristics
  - all attributes are private
  - all operations are public
    - subclasses must access attributes of parents via operations
    - You can change this later in implementation, when a profiler indicates that these heuristics are slowing you down (do so only in the presence of such information, e.g. do not assume that it is slowing you down without first checking!)

Maciaszek, section 5.1.5

- Derived Information
  - Derived attributes and associations indicate information that can be computed from other attributes and associations
    - As such, when notated, these elements represent redundant information
      - however, its best to specify this information explicitly to remind you later that it can be computed!
  - The notation for derived attributes and associations is a “/” in front of their name
    - See pages 166 and 167

Class Layers

- Software Systems consist of components and subsystems that can be highly interconnected
  - The complexity of such systems is defined by connection
  - Maximum connection between a set of classes is n (n - 1) / 2
    - See figure 5.16 on page 173
- Why is this a problem?
  - The 7 ± 2 rule: humans have a hard time dealing with ten or more concepts simultaneously
- Approach
  - The use of hierarchy can be beneficial in such situations; modularity is important too!

Class Layers, continued

- Using hierarchy the fully connected class graph of figure 5.16 is reduced in figure 5.17
  - Classes are layered in a hierarchy and classes in layer *n* can only be connected to classes in layers *n-1*, *n*, and *n+1*
  - UML provides the notion of a package to address these issues
    - packages are used to group classes

Packages

- Packages are indicated using a “folder icon” in the UML
  - Classes are clustered “inside”
- Oftentimes, packages represent subsystems and can be stereotyped with «subsystem»
- Packages can be also be nested

Summary

- Reviewed advanced UML notations
- Discussed UML extension mechanisms
- Introduced notion of UML Packages

- What’s Next?
  - Review of more information in Section 5
    - Advanced issues of generalization, aggregation, and a new concept: delegation