CIS W338

Introduction to UML
Classes and Class Diagrams
State Diagrams and Class States

What is UML

• UML is the unification of the notations developed by Booch, Jacobson, and Rumbaugh. ("The Three Amegos.")
• It is a language – it has syntax and semantics
• Provides a framework for organizing design constructs then defining each construct with appropriate text.
Classes

- name
- attributes
- operations

Objects

- object:Class
- attributes
- operations
Attributes

- Information about an object.
- Not necessarily a variable (but most of the time it will be).
- Sometimes there is a different internal and external representation. (E.G, date).
- Sometimes a function of several variables.
- A ‘/’ indicates a read-only attribute.

Get/Set Operations

- Some authors advocate get and set operations for every attribute
  - Separates external from internal view
  - Helpful for debugging
  - Provides for validation
- A simple get/set pair, with no validation, is equivalent to making the attribute public.
Overloaded Operations

• An operation’s name includes its parameter types.
• Each variant is listed separately on the class symbol in the class diagram.

Visibility

+ Public visible to any object
– Private visible only to self (and friends in C++)
# Protected visible to self and subclasses.
Visibility in Rational Rose

Rational Rose, and some other tools, use:

<table>
<thead>
<tr>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Attribute</td>
</tr>
<tr>
<td>Protected Attribute</td>
</tr>
<tr>
<td>Public Attribute</td>
</tr>
<tr>
<td>Private Operation</td>
</tr>
<tr>
<td>Protected Operation</td>
</tr>
<tr>
<td>Public Operation</td>
</tr>
</tbody>
</table>

Class Attributes and Operations

- A class attribute or class operation corresponds to a C++ static member.
- Indicated in UML by underlying the name.
Abstract Operations/Classes

• Abstract operation corresponds to a pure virtual function in C++
• Uses the keyword abstract in Java.
• Abstract class corresponds to a class with at least one pure virtual function in C++
• Corresponds to Java’s interface class.
• Indicated by the class/operation name in italics and the word abstract in braces (“{abstract}”) after the name.

Utility Class

• Used to group operations.
• Individual objects never instantiated.
• Indicated by the word utility enclosed in guillemets (“«utility»”) preceding the name of the class.
Parameterized Class

```
Parameterized Class

set

fleet
```

Single Inheritance

```
Single Inheritance

PoweredVehicle

Car  Truck

PoweredVehicle

Car  Truck
```
Multiple Inheritance

Subclass Partitioning

- **Disjoint**: No object of the base class can belong to more than one subclass
- **Overlapping**: Some objects of the base class can belong to more than one subclass
- **Incomplete**: Not all possible objects of base class accounted for
- **Complete**: All possible objects of base class in one of the subclasses
Guidelines for Class Refinement

- If you have a completely partitioned class, the base class can be an abstract base class.
- If you have an overlapped partitioning, the common elements should be grouped into a superclass.
Associations

- An association in UML represents a varying population of relationship links between instances of classes.
Associations (example 2)

Association as a Class
Note on Associations

- Association names, role names, cardinality are all optional.
- An association implies the ability to navigate from one class to the other (i.e., one class can send the other class a message).

Bi-directional

```
class Person
{
    :
    Dog* owns;
    :
};
```

```
class Dog
{
    :
    Person* owner;
    :
};
```
People know their Dogs

```
class Person
{
    ::;
    Dog* owns;
    ::;
};
```

Dogs know their Owner

```
class Person
{
    ::;
    Person* owner;
    ::;
};
```
Association defined by a Class

```
Ownership

Person owner 1..1 owns Dog 0..*
```

Association defined by a Class (2)

```c
class Person
{
    :
    Ownership the_O;
    :
};

class Dog
{
    :
    Ownership the_O;
    :
};
```
class Ownership

class Ownership
{
    ...
    Dog* owns;
    Person* owner;
    ...
};

Composition
class Glider

class Glider
{
    :
    Fuselage fuselage;
    Tail tail;
    Wing leftWing;
    Wing rightWing;
    :
};

Composition Characteristics

- The composite object does not exist without its components.
- A given component object may be part of only one composite.
- Components are generally of different types.
Aggregation

```
class MgntReport
{
    ..
    list<Paragraph*> textPart;
    ..
};
```
Aggregation Characteristics

• The aggregate object may potentially exist without its constituent objects.
• At any time, each object may be a constituent of more than one aggregate.
• The constituents of an aggregate will generally belong to the same type.

Interaction Diagrams

• Show the interaction between objects.
• Two types
  – Collaboration Diagram
  – Sequence Diagram
• Both contain the (almost) same information.
• Software tools automatically convert one to the other.
Collaboration Diagram

Layout of diagram is free-form (similar to class diagram). Sequence number (optional) shown the order in which the messages are sent. Direction of the arrow indicates from sender to receiver. Reverse data flow arrow (optional) indicates information returned from a message.

Sequence Diagram

Time flows from top to bottom.
Model Consistency

- For object A to send a message to object B
  - There must be an association between the classes.
  - There must be an operation in class B that corresponds to the message.
- During design you iteratively work with the class diagram(s) and the interaction diagrams.
- Software tools help maintain the synchronization by offering menus of the available operations and adding operations.
State Diagrams

• Shows the states that an object may assume and the transitions the objects make from state to state.
• A state is indicated by the values of the attributes.
  – Useful to consider attributes that can take a few values.
  – Can also use ranges.
• States can be nested.

Current Inspection Status

- received
  - inspectorSelectsItem
- inInspection
  - inspectorAcceptsItem
    - accepted
  - inspectorRejectsItem
    - rejected
Operating State

```
standBy

operatorStartsMachine
[ serviceStat=inService ] /
  safetyBrake.turnOff;
  mainMotor.turnOn

  [ when(actualSpeed < epsilon) ] /
  safetyBrake.turnOn;
  operator.notifyStopped

accelerating

  [ when(actualSpeed >= runSpeed) ] /
  operator.notifyRunning

running

  operatorStopsMachine /
  mainMotor.turnOff

decellerating
```

Service State

```
inService

reportInService

beginMaintenance

removeFromService

waitingForRepair

inRepair
```

9/27/00 ©2000 P. Wolfgang

45

46
Nested States

Nested State with History
Authorization Status

Customer submits order

Tentative

Confirmed

Order complete / shipped=true

Customer cancels order

Cancelled

[after 30 days]
[Not shipped]

Transient States

Standby

TurningOn

[OK]
[Not OK]

MainMotor.turnOn(OK)

MainMotor.turnOff

Operator/Standby

MainMotor.turnOff

MainMotor.turnOn

Accelerating
Continuous Variable Attributes

Keep It Simple

- State diagrams are abstractions of class behavior.
- Should not capture every possible facet and algorithm.
- If diagram becomes a hodgepodge, may want to re-think design.
- May want to split class to reduce complexity.
Not for Everyone

- Many classes will not have a meaningful state diagram.
- Class behavior may be better captured through class invariant and definition of operations and attributes.

State-Space and Behavior

- A class should represent a uniform abstraction of the properties of the individual objects that belong to that class.
- The two properties of a class are its state-space and its allowed behavior.
Chess Knight

- **State Space**
  - Can occupy any space on the board

- **Behavior**
  - Moves one square straight in any direction followed by one move diagonal either left or right.

Chess Queen

- **State-Space**
  - Can occupy any space on the board

- **Behavior**
  - Can move any number of squares straight or along a diagonal
Chess Bishop

- State-Space
  - Can only occupy spaces of one color.
- Behavior
  - Moves along diagonals any number of squares.

Modified Knight

- State-Space
  - Can occupy all but the center four squares.
- Behavior
  - Same as regular knight.
State-Space Definition

- The *state-space* of a class C is the ensemble of all the permitted states of any object C.
- The *dimensions* of a state-space are the coordinates needed to specify the state of a given object (I.E., number of orthogonal attributes.)
- The *state* of an object is its value. (I.E., the value of the attributes.)

State-Space of Subclasses

- If B is a subclass of A, then the state-space of B must be entirely contained within the state-space of A.
- B’s state space is *confined* by A’s state space.
State-Space of Subclasses (2)

- If \( B \) is a subclass of \( A \), then \( B \)’s state-space must comprise of at least the dimensions of \( A \)’s – but it may comprise more.
- \( B \)’s state-space extends from \( A \)’s.

The Behavior of a Class

- The allowed behavior of a class \( C \) is the set of transitions that an object of class \( C \) is permitted to make between states in \( C \)’s state space.
Class Invariant

- A class invariant is a condition that every object of that class must satisfy at all times (when the object is in equilibrium).

Class Triangle

- Has sides a, b, c.
  - Triangle
    • \( a + b > c \)
    • \( b + c > a \)
    • \( c + a > b \)
  - Isosceles Triangle
    • \( a = b \) or \( b = c \) or \( c = a \)
  - Right Triangle
    • \( a^2 + b^2 = c^2 \)
Caution

• The class hierarchy shown for the triangles is only valid provided the base class does not contain an operation that permits violation of the invariant of all possible derived classes.
• For example an operation to set the value of a side to an arbitrary value.

Pre- and Post-conditions

• Every operation has a pre-condition and a post-condition.
• The pre-condition must be true when the operation begins to execute.
• The post-condition must be true when the operation ends its execution.
Contract

• If the sender of a message can guarantee that the pre-condition is true, then the target operation will guarantee that the post-condition will be true after execution.

• If, on the other hand, the sender cannot guarantee the pre-condition, then the whole deal is off: the operation is neither obligated to execute nor to guarantee the post-condition.

Design of Rectangle

A) Four points (the four corners)
   – Specifies all possible quadrilaterals.

B) Two points (top-left, bottom-right)
   – Specifies any size and position, but not orientation. (Used by Windows.)

C) Center point, Height, Width, Orientation.
   – Can specify any size, position, and orientation.
States To Avoid

• Illegal States
• Incomplete States
• Inappropriate States

Illegal States

• An illegal state is one that violates the class invariant.
• Example: Rectangle::movePoint
  – This operation moves one of the points that defines the rectangle to an arbitrary location.
  – May change the rectangle into some other shape.
Incomplete States

- The set of operations does not permit an object to reach all valid states.
- Example:
  - Constrain Width > Height.
  - Cannot produce tall rectangles.
  - Cannot produce squares.
  - Note: If we have an orientation, perhaps Width ≥ Height may be reasonable.

Inappropriate States

- Providing interfaces outside the class abstraction.
- Example:
  - A stack pointer that access a point in the middle of the stack.
  - This may be an appropriate when modeling a stack frame, but not a pure stack.
Behaviors to Avoid

- Illegal behavior
- Dangerous behavior
- Irrelevant behavior
- Incomplete behavior
- Awkward behavior
- Replicated behavior

Illegal Behavior

- An operation that allows an object to make illegal transitions between states. I.E. a transition not on the state diagram.

- Example:
  - CustomerOrder going from unapproved to filled.
Dangerous Behavior

- Operations which permit a class to enter illegal states (even temporarily).
- Example: to go from approved to backOrdered:
  - Set the number of items ordered to negative of the value.
  - Send a fill message to self.
    - Fill message recognized the negative number as a flag to set the item to backordered.

Irrelevant behavior

- Operations that are not relevant to the class.
- Example:
  - computeLoanRepayment as a member of CustomerOrder.
Incomplete Behavior

• All required transitions not supported.
• Example
  – Go from approved to unapproved when the customer files bankruptcy.

Awkward Behavior

• Multiple messages to accomplish a single transition.
• Example
  – Send a message to transition back to approved.
  – Send a message to set the shipment and move the order back to filled.
Replicated behavior

- Multiple ways to do the same thing.
- EXAMPLE:
  - turnRight();
  - turnClockwise(90);
- May creep in as a consequent of evolutionary design. Removal of replicated behavior may be more trouble than it is worth.

Strive For

- Ideal States
  - Can only reach all legal states and nothing but legal states.
- Ideal Behavior
  - Only move to legal states
  - Only move in legal ways
  - Only one interface for any given piece of behavior.