UML

- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.

- It can be used with all processes, throughout the development life cycle, and across different implementation technologies.

"Describes the structure of objects in a system including identity, relationship to other objects, attributes, and operations…"

The model abstracts and captures relevant problem domain concepts. It is represented graphically with

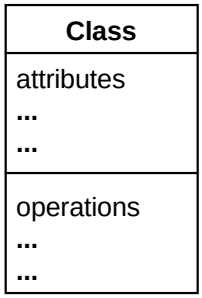
- classes arranged into hierarchies sharing common structure and behavior
- class associations (or relationships)
- class attribute values (data carried by object instances)
- class operations (behavior an object performs or undergoes)

Classes

Classes describe a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

Objects in a class share a common semantic purpose beyond common attributes and behavior.

Notation



Classes should be defined by an informative statement of purpose and scope.

Class names are critical for ease of communication. Strive for crisp, clear, and direct naming.

Attributes

Attributes should be:

- complete - capture all information relevant to the class being defined.

- fully factored - each attribute captures a separate aspect of the class abstraction

- mutually independent - attributes take on values independent of each other.

Note: Dependent attributes (or derived attributes) may be introduced to facilitate performance.

Attribute Classifications

descriptive attributes - Provide facts intrinsic to each instance of the class

naming attributes - Provide facts about the labels and names carried by an object (class instance)

classifying attributes - Capture the facts, which tie an object to another object of a different class.

Operations and Methods

All objects in a class share the same operations. A method is an implementation of an object for a class.

Each operation has a target object as an implicit argument, and optionally, input and output arguments.

The same operation may apply to different classes. This is the foundation of *polymorphism*.

To exploit polymorphism, operations on more than one class should have consistent intent (semantics) and the same the same signature (number and types of input and output arguments).

As with classes, naming is important. *Avoid* using the same name for two operations that are semantically different.

Associations and Links

"links and associations are the means for establishing relationships among objects and classes"

Links are physical or conceptual connections between object instances.

An association describes a group of links with a common structure and common semantics. Links are instances of associations.

An association describes a set of potential links in the same way that a class describes a set of potential objects.

Associations are inherently bi-directional.

Associations are typically binary, but may be of any higher order.

Associations are often left unnamed when they can be easily identified by their classes. This can, however, lead to ambiguity.

Roles

A role name is a name that uniquely identifies one end of an association. Each role on an association identifies an object or set of objects associated with an object at the other end.

The role name clarifies the purpose an object serves in an association.

Role names are necessary for associations between two objects of the same class.

For example, a directory can contain other directories. Roles help clarify this association.

Multiplicity

Multiplicity refers to how many instances of one class may relate to a single instance of an associated class.

Determine classes and associations before deciding on multiplicity.

Multiplicity is represented by symbols at the end of an association. Here are the various possibilities.

An instance of A is associated with ***zero or more*** instances of B

An instance of A is associated with ***zero or one*** instances of B

In special cases, the multiplicity can be specified with a number, interval, a set of numbers, or whatever illustrates the point.

An instance of A is associated with two or more instances of B

An instance of A is associated with 1 through 5 instances of B

Link Attributes

An attribute is a property of the objects in a class. A *link attribute* is a property of a link in an association. Each link attribute has a value for each link.

The link attribute arises because the attribute is not actually an attribute of either class in the association. It is a result of the association.

Test for a link attribute by determining if the attribute exists only when the two objects exist.

As a rule, link attributes should not be folded into a class because future flexibility is reduced if the multiplicity of the association should change.

Qualification

A qualified association relates two object classes and a qualifier. It is an attribute that reduces the effective multiplicity of an association by identifying a single target object

Qualified associations improve semantic accuracy and increase the visibility of navigation paths.

A qualified association is equivalent to the concepts known as associative arrays, maps, and dictionaries.

Relationships

- A is a logical/physical part of B
- A is logically/physically contained in B
- A has a knowledge/awareness of B with persistence beyond a single encounter
- A is a member of B
- A communicates with B (on a regular basis)
- A owns or manages B

Aggregation

Aggregation is typically described as a “is part of” relationship, in which objects representing the components of an object are associated with the object containing the entire assembly.

Aggregation implies that an object owns or is responsible for a group of other objects. A classical example is a parts explosion diagram.

In comparison, an association link implies that an object knows of another object in the sense that they must interact in some capacity.

The objects in an aggregation are independent, but are more tightly coupled than through an ordinary association.

Aggregation is transitive. That is, A is part of B, and B is part of C, implies A is part of C. Aggregation is not symmetric. This is A is part of B does not imply B is part of A.

The semantics of aggregation are not precise. For example, can the aggregate exist only if all the aggregated objects exist? How coupled are the lifetimes of the two objects? Can a part be contained in multiple aggregates?

When in doubt, model with associations.

Use a consistent interpretation of aggregation in your models.

Generalization and Inheritance

Generalization and inheritance are techniques that allow sharing of similarities between objects while preserving their differences.

Generalization defines a relationship between a class and one or more refined versions of it. The class being refined is called the *superclass* and the refined version is called the *subclass*. Attributes and operations of the superclass are shared by the subclasses

Inheritance is described as a “is a” relationship because each subclass is an instance of the superclass as well. An instance of the subclass is a simultaneous instance of the superclass.

A subclass instance includes a value for every attribute of the superclass. Any operation of the superclass can be applied to the subclass. The subclass may add its own attributes and operations.

Generalization and inheritance are transitive across an arbitrary number of levels.

Overriding

A subclass may override default values of attributes or implementations of methods in the superclass. The overriding (subclass) feature replaces the overridden (superclass) feature.

A feature may be overridden to specify behavior tailored for the subclass.

Never change the semantics or signature of an overridden feature. A subclass is an instance of the superclass. Changing a feature or signature either breaks the interface contract.

Aggregation and Composition

Both represent part-whole, element-collection, and component-container type relationships.

In composition the component object can only be part of one container object.

In composition the parts and the whole have the same persistence in time.