Unified Modeling Language (UML)

Unified Modeling Language -- A system that allows a software designer to graphically layout and model a software application. It gives designers a way to literally draw a map of how a piece of software will be constructed and function.

A Brief History

Flow Charts

Prior to the mid 1970s, software design, if there was any, may have been aided by the use of Flow Charts. These were diagrams of how the various parts of a program interacted. It was primarily Procedural Oriented as opposed to Object Oriented. A Flow Chart displayed graphically the "flow of control" through a program.

Object Oriented Modeling Languages

From the mid 1970s to the late 1980s a number of new software "modeling languages" appeared. The three developers who stand out for their work during this period are: (1) **Grady Booch** of Rational Software Corporation who developed the **Booch Method**, (2) **Ivan Jacobson** of Objectory, Inc. who developed **Object Oriented Software Engineering**, and (3) **James Rumbauch** of General Electric who developed the **Object Modeling Technique**.

Unified Modeling Language

Booch, Jacobson, and Rumbaugh were independently working toward the same objectives. In the mid 1990s they began collaborating. The result was the **Unified Modeling Language** of which the first version (then called just the Unified Method) was released in October of 1995.

Why Use UML?

To be successful a software company must produce quality software, that meets requirements and is released on time and within budget.

This was easy to do when computers were very simple and slow devices and programs did little more than play solitaire, convert between English and Metric measures, and balance your checkbook.

Today when a program may be called upon to "simulate a battlefield scenario with full and accurate engineering and physics based rendering" this can be very difficult if not impossible without some means of organizing the design phase.

Modeling a software system using UML is a lot like drawing up the blueprints to build a home. Would you want the builders to just go to your site and just start "hacking together" your home?

Getting started with UML

UML is not difficult to learn. It consists of three parts.

1. The basic building blocks (the words of the language)

2. Rules controlling how the blocks are put together (the syntax)
3. Common mechanisms that apply throughout the language (use conventions or semantics)

This brief introduction to UML just covers the basic building block of UML.

Basic Building Blocks of UML

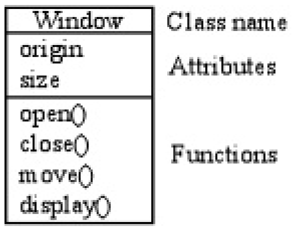
Three Types

1. **Things**
   Four Types
   - Structural things -- The nouns of UML models. These represent elements that are conceptual or physical. There are seven kinds of structural things: Class, Interface, Collaboration, Use Case, Active Class, Component, and Node. (See the examples below.)
   - Behavioral things -- Dynamic parts of UML models. The verbs which represent behavior over time and space. There are two kinds of behavioral things: Interaction, and State Machine. (See the examples below.)
   - Grouping things -- Organizational parts of UML. These are boxes into which models can be decomposed. There is only one kind of grouping thing, the Package. (See the examples below.)
   - Annotational things -- Explanatory parts of UML. Used to describe, illuminate, and remark any element of a model. There is only one kind of annotational thing, the Note. (See the examples below.)
2. **Relationships**
   Four Types
   - Dependency -- A semantic relationship in which a change on one thing (the independent thing) may cause changes in the other thing (the dependent thing). (See the examples below.)
   - Association -- A structural relationship describing links between objects. May also include labels to indicate number and role of the links. In the example shown below there may be any number of employees (*) each of which has 0 or 1 employer. (See the examples below.)
   - Generalization -- A specialization/generalization relationship. Simply put this describes the relationship of a parent class (generalization) to its subclasses (specializations). (See the examples below.)
   - Realization -- Defines a relationship in which one class specifies something that another class will perform. Example: The relationship between an interface and the class that realizes or executes that interface. (See the examples below.)
3. **Diagrams**
   Nine Types
   - Class Diagram -- A set of classes, interfaces, and collaborations and their relationships. Most often found in modeling Object Oriented systems. (See the examples below.)
   - Object Diagram -- A set of objects and their relationships. Represents static instances of things found in class diagrams. (See the examples below.)
   - Use Case Diagram -- A set of Use Cases and actors. (See the examples below.)
   - Sequence Diagram -- An interactive diagram (set of objects, relationships, and messages that may be exchanged) emphasizing the time-ordering of messages. (See the examples below.)
   - Collaboration Diagram -- An interaction diagram emphasizes the structural organization of the objects that send and receive messages. (See the examples below.)
   - Statechart Diagram -- Shows a state machine with states, transitions, events, and activities. (See the examples below.)
   - Activity Diagram -- Special type of statechart diagram that shows the flow from activity to activity within a system. (See the examples below.)
   - Component Diagram -- Shows the organizations and dependencies among a set of components. (See the examples below.)
   - Deployment Diagram -- Shows the configuration of run-time processing nodes and the components that are part of them. (See the examples below.)

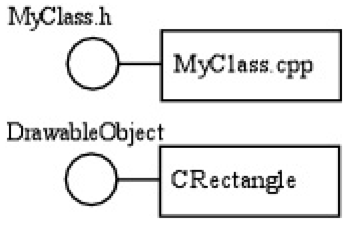
Examples of the Basic Building Blocks of UML

Seven Types of Structural Things

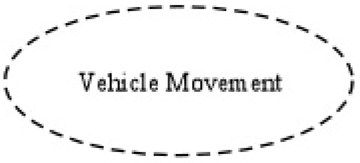
1. Class - An object with defined attributes and operations. A class in UML is very much like a class in C++.



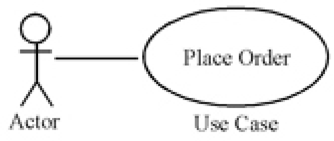
2. Interface - A collection of functions that specify a service of a class or component, i.e. externally visible behavior of that class.



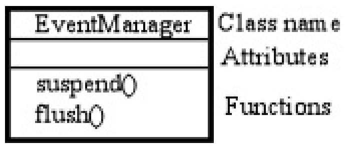
3. Collaboration - A larger pattern of behaviors and actions. Example: All classes and behaviors that create the modeling of a moving tank in a simulation.



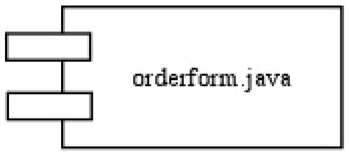
4. Use Case - A sequence of actions that a system performs that yields an observable result. Used to structure behavior in a model. Is realized by a collaboration.



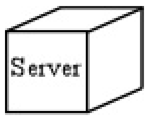
5. Active Class - Like a class but its represents behavior that runs concurrent with other behaviors, i.e. threading.



6. Component - A physical and replaceable part of a system that implements a number of interfaces. Example: a set of classes, interfaces, and collaborations.

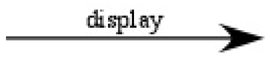


7. Node - A physical element existing at run time and represents a resource.

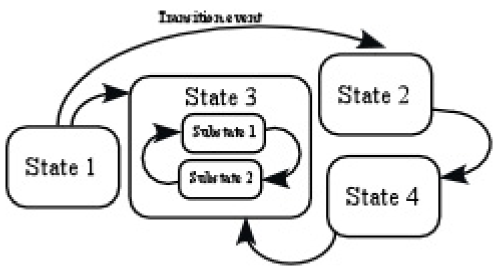


Two Types of Behavioral Things

1. Interaction - A behavior made up of a set of messages exchanged among a set of objects in a particular context to accomplish a specific purpose.

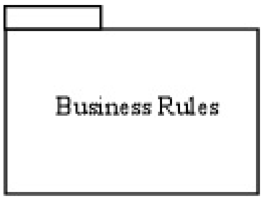


2. State Machine - A behavior that specifies the sequences of states an object or interaction goes through during its' lifetime in response to events.



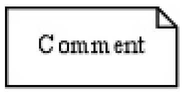
One Type of Grouping Thing

1. Package - A general purpose mechanism for organizing elements into groups.



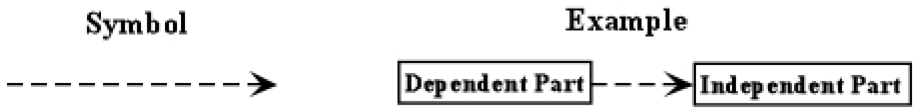
One Type of Annotation Thing

1. Note - A symbol to display comments.

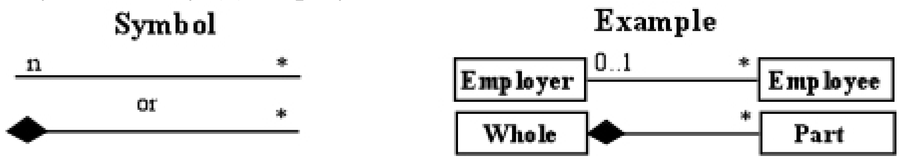


Four Types of Relationships

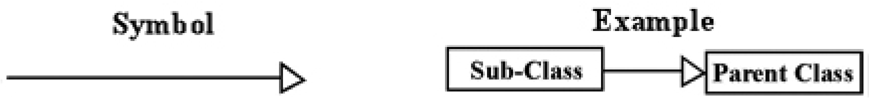
1. Dependency - A semantic relationship in which a change on one thing (the independent thing) may cause changes in the other thing (the dependent thing).



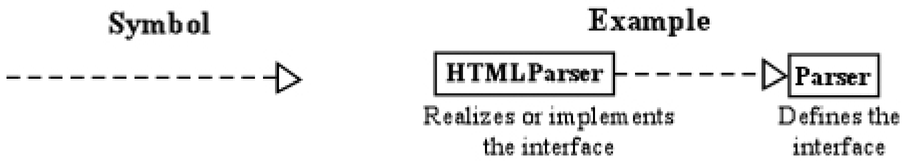
2. Association - A structural relationship describing links between objects. May also include labels to indicate number and role of the links. In the example there may be any number of employees (*) each of which has 0 or 1 employer. The double arrowhead is used to indicate a "has-a" relationship, meaning there is 1 employer who may have many (*) employees.



3. Generalization - A specialization/generalization relationship. Simply put this describes the relationship of a parent class (generalization) to its subclasses (specializations).

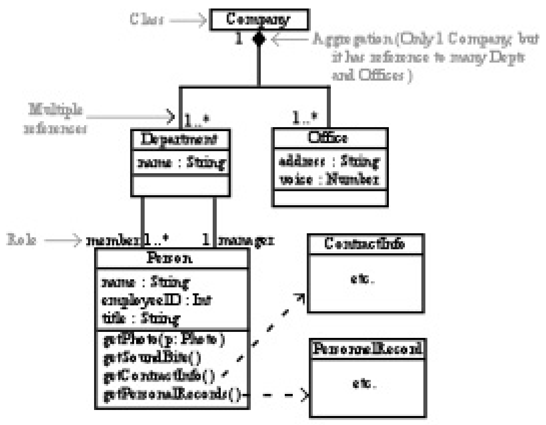


4. Realization - Defines a relationship in which one class specifies something that another class will perform. Example: The relationship between an interface and the class that realizes or executes that interface.

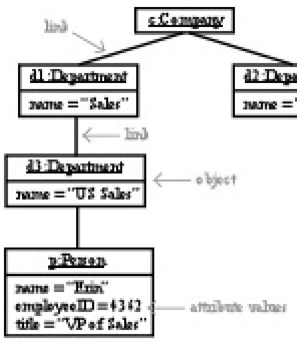


Nine Types of Diagrams

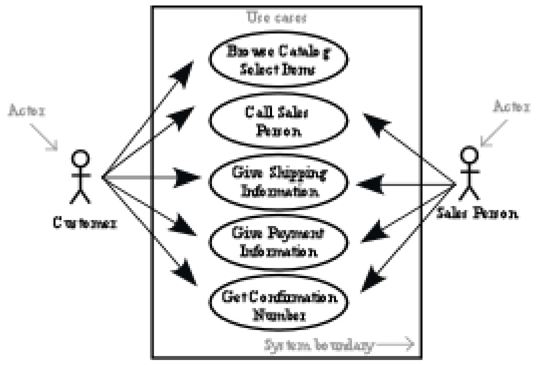
1. Class Diagram - A set of classes, interfaces, and collaborations and their relationships. Most often found in modeling OO systems.



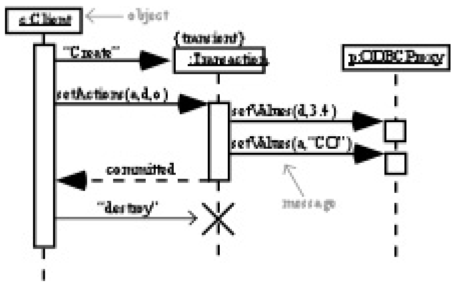
2. Object Diagram - A set of objects and their relationships. Represents static instances of things found in class diagrams.



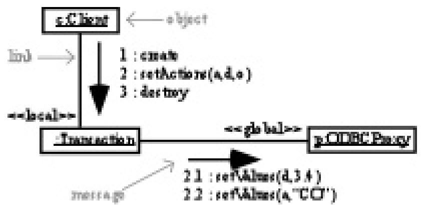
3. Use Case Diagram - A set of Use Cases and actors.



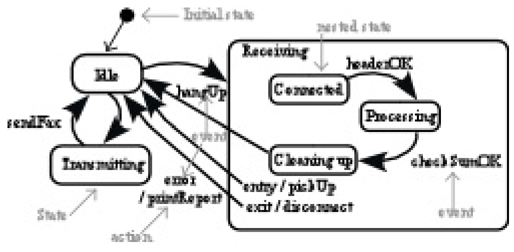
4. Sequence Diagram - An interactive diagram (set of objects, relationships, and messages that may be exchanged) emphasizing the time-ordering of messages.



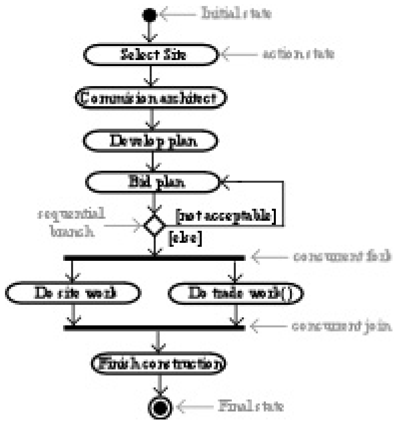
5. Collaboration Diagram - An interaction diagram emphasizes the structural organization of the objects that send and receive messages.



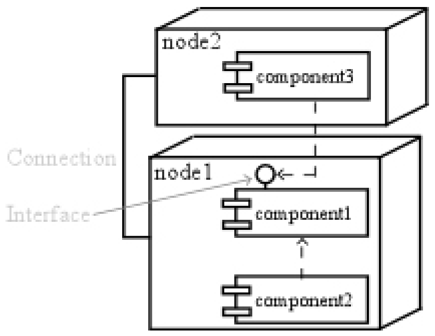
6. Statechart Diagram - Shows a state machine with states, transitions, events, and activities.



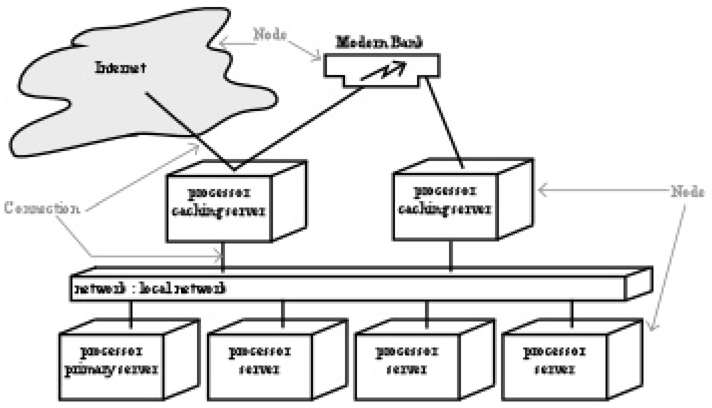
7. Activity Diagram - Special type of statechart diagram that shows the flow from activity to activity within a system.



8. Component Diagram - Shows the organizations and dependencies among a set of components.



9. Deployment Diagram - Shows the configuration of run-time processing nodes and the components that are part of them.


Internet
Node
Modem Bank
Connection
processor caching server
processor caching server
Node
network : local network
processor primary server
processor server
processor server
processor server