UML Tutorial

The Unified Modeling Language has quickly become the de-facto standard for building Object-Oriented software.

The **OMG** specification states: *"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."*

The important point to note here is that UML is a 'language' for specifying and not a method or procedure. The UML is used to define a software system; to detail the artifacts in the system, to document and construct - it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology' but in itself it does not specify that methodology or process.

UML defines the notation and semantics for the following domains:

- The User Interaction or Use Case Model - describes the boundary and interaction between the system and users. Corresponds in some respects to a requirements model.
- The Interaction or Communication Model - describes how objects in the system will interact with each other to get work done.
- The State or Dynamic Model - State charts describe the states or conditions that classes assume over time. Activity graphs describe the workflow's the system will implement.
- The Logical or Class Model - describes the classes and objects that will make up the system.
- The Physical Component Model - describes the software (and sometimes hardware components) that make up the system.
- The Physical Deployment Model - describes the physical architecture and the deployment of components on that hardware architecture.

UML 2.0

UML 2 builds on the already highly successfull UML 1.x standard, which has become an industry standard for modeling, design and construction of software systems as well as more generalized business and scientific processes. UML 2 defines 13 basic diagram types, divided into two general sets:

1. Structural Modeling Diagrams

Structure diagrams define the static architecture of a model. They are used to model the 'things' that make up a model - the classes, objects, interfaces and physical components. In addition they are used to model the relationships and dependencies between elements.

- Package diagrams are used to divide the model into logical containers or 'packages' and describe the interactions between them at a high level
- Class or Structural diagrams define the basic building blocks of a model: the types, classes and general materials that are used to construct a full model
- Object diagrams show how instances of structural elements are related and used at run-time.
- Composite Structure diagrams provide a means of layering an element's structure and focusing on inner detail, construction and relationships
- Component diagrams are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface
- Deployment diagrams show the physical disposition of significant artefacts within a real-world setting.

2. Behavioral Modeling Diagrams

Behavior diagrams capture the varieties of interaction and instantaneous state within a model as it 'executes' over time.

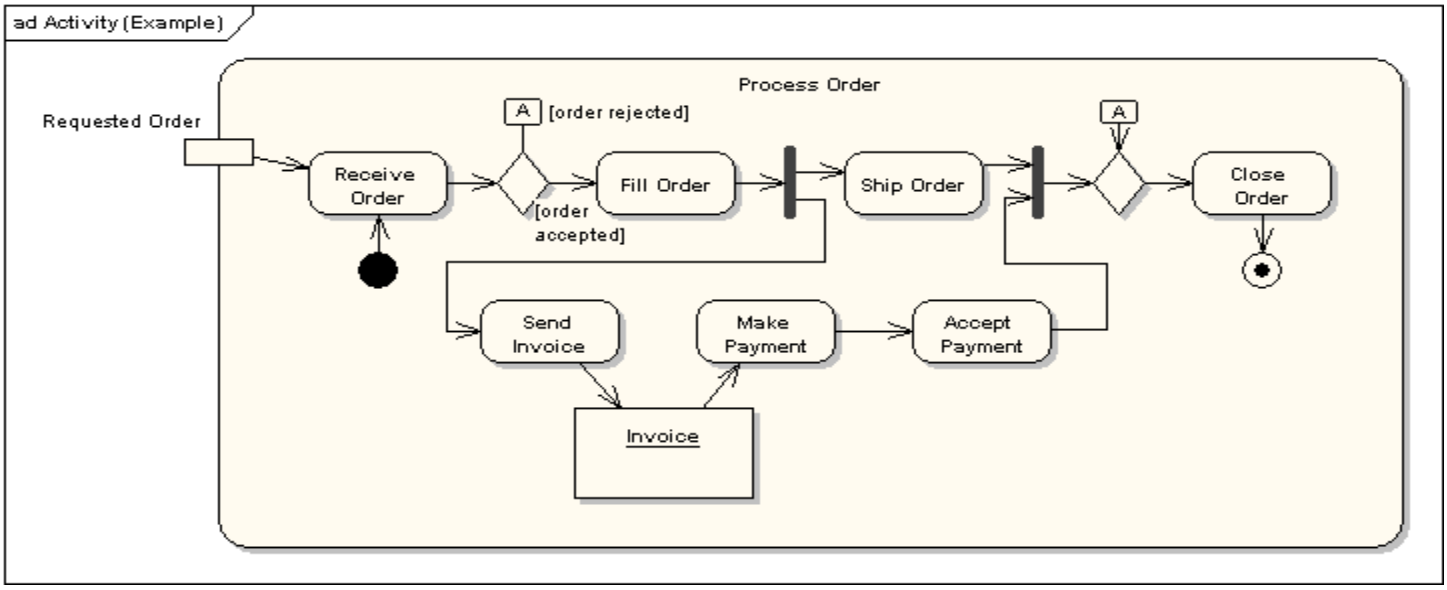
- Use Case diagrams are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios
- Activity diagrams have a wide number of uses, from defining basic program flow, to capturing the decision points and actions within any generalized process
- State Machine diagrams are essential to understanding the instant to instant condition or "run state" of a model when it executes
- Communication diagrams show the network and sequence of messages or communications between objects at run-time durin a collaboration instance
- Sequence diagrams are closely related to Communication diagrams and show the sequence of messages passed between objects using a vertical timeline
- Timing diagrams fuse Sequence and State diagrams to provide a view of an object's state over time and messages which modify that state
- Interaction Overview diagrams fuse Activity and Sequence diagrams to provide allow interaction fragments to be easily combined with decision points and flows

UML 2 Activity Diagram

Activity Diagrams

In UML an activity diagram is used to display the sequence of activities. Activity Diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be use to detail situations where parallel processing may occur in the execution of some activities. Activity Diagrams are useful for Business Modelling where they are used for detailing the processes involved in business activities.

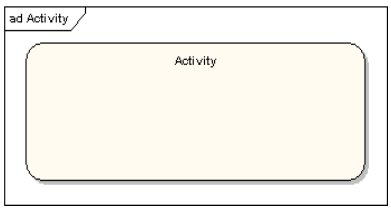
An Example of an Activity Diagram is shown here



The following sections describe the elements that constitute an Activity diagram.

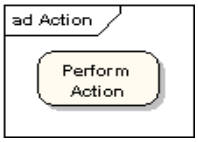
Activities

An activity is the specification of a parameterized sequence of behaviour. An activity is shown as a round-cornered rectangle enclosing all the actions, control flows and other elements that make up the activity.



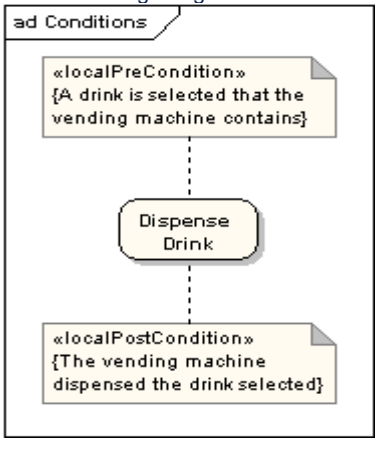
Actions

An action represents a single step within an activity. Actions are denoted by round-cornered rectangles.



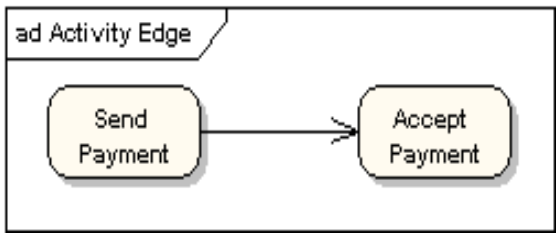
Action Constraints

Constraints can be attached to an action. The following diagram shows an action with local pre- and post-conditions.



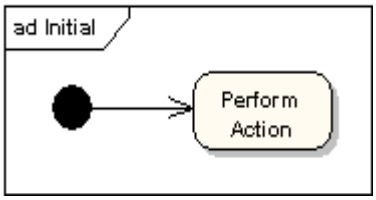
Control Flow

A control flow shows the flow of control from one action to the next. Its notation is a line with an arrowhead.



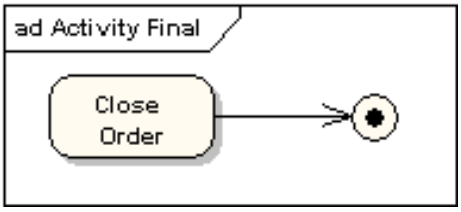
Initial Node

An initial or start node is depicted by a large black spot, as depicted below.

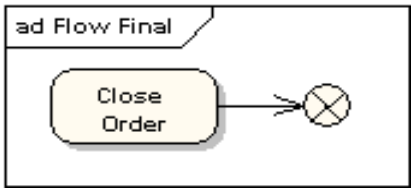


Final Node

There are two types of final node: activity and flow final nodes. The activity final node is depicted as a circle with a dot inside.



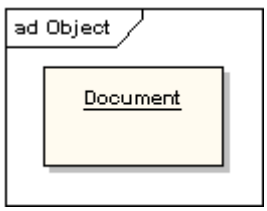
The flow final node is depicted as a circle with a cross inside.



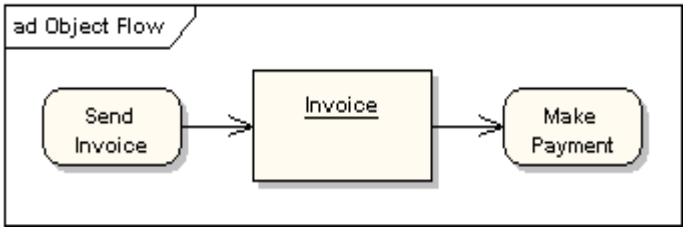
The difference between the two node types is that the flow final node denotes the end of a single control flow; the activity final node denotes the end of all control flows within the activity.

Objects and Object Flows

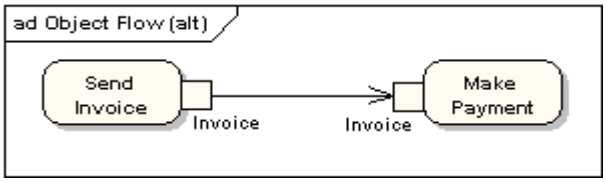
An object flow is a path along which objects or data can pass. An object is shown as a rectangle.



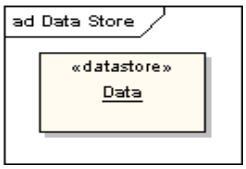
An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.



An object flow must have an object on at least one of its ends. A shorthand notation for the above diagram would be to use input and output pins.

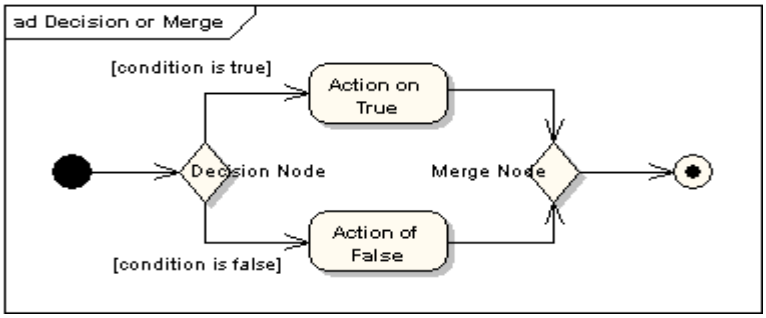


A data store is shown as an object with the «datastore» keyword.



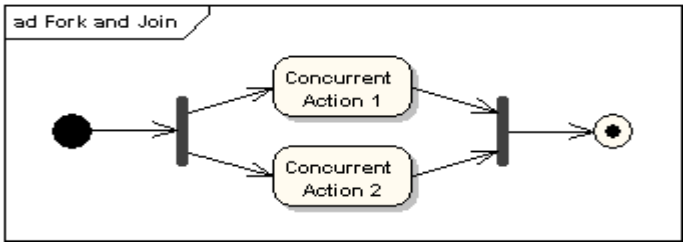
Decision and Merge Nodes

Decision nodes and merge nodes have the same notation: a diamond shape. They can both be named. The control flows coming away from a decision node will have guard conditions which will allow control to flow if the guard condition is met. The following diagram shows use of a decision node and a merge node.



Fork and Join Nodes

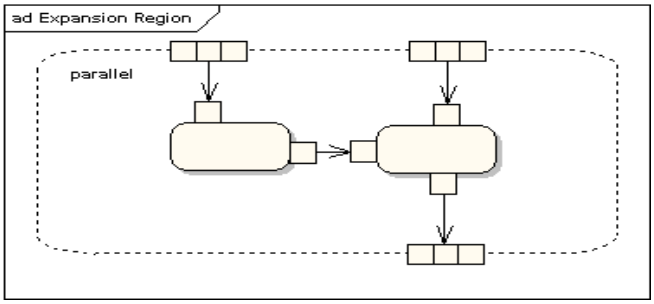
Forks and joins have the same notation: either a horizontal or vertical bar (the orientation is dependent on whether the control flo is running left to right or top to bottom). They indicate the start and end of concurrent threads of control. The following diagram shows an example of their use.



A join is different from a merge in that the join synchronises two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received. A merge passes any control flows straight through it. If two or more inflows ar received by a merge symbol, the action pointed to by its outflow is executed two or more times.

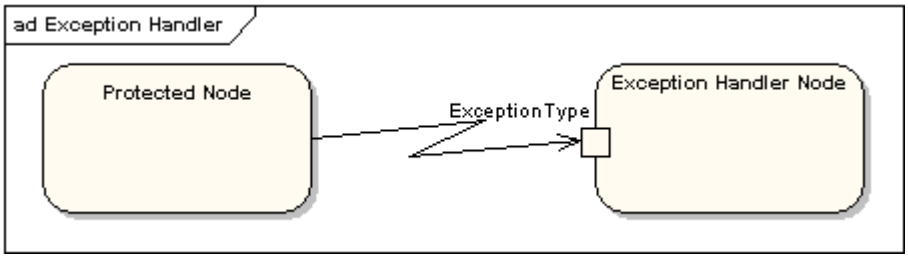
Expansion Region

An expansion region is a structured activity region that executes multiple times. Input and output expansion nodes are drawn as a group of three boxes representing a multiple selection of items. The keyword iterative, parallel or stream is shown in the top left corner of the region.



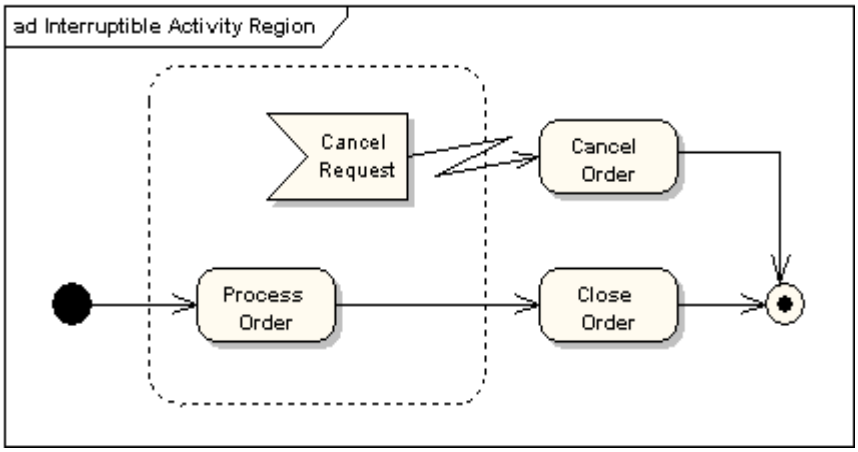
Exception Handlers

Exception Handlers can be modelled on activity diagrams as in the example below.



Interruptible Activity Region

An interruptible activity region surrounds a group of actions that can be interrupted. In the very simple example below, the Proces Order action will execute until completion, when it will pass control to the Close Order action, unless a Cancel Request interrupt is received which will pass control to the Cancel Order action.



Partition

An activity partition is shown as either horizontal or vertical swimlanes. In the following diagram, the partitions are used to separate actions within an activity into those performed by the accounting department and those performed by the customer.