ACTIVITY DIAGRAMS

- An activity diagram shows the flow from activity to activity
- an activity diagram shows the flow of an object, how its role, state and attribute values changes
- activity diagrams is used to model the dynamic aspects of a system
- Activities result in some action (Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, such as evaluating an expression)
- Figure 1: activity diagram
- an activity diagram is a collection of vertices and arcs
- Activity diagrams commonly contain Activity states and action states, Transitions, Objects
- activity diagrams may contain simple and composite states, branches, forks, and joins
- the initial state is represented as a solid ball and stop state as a solid ball inside a circle

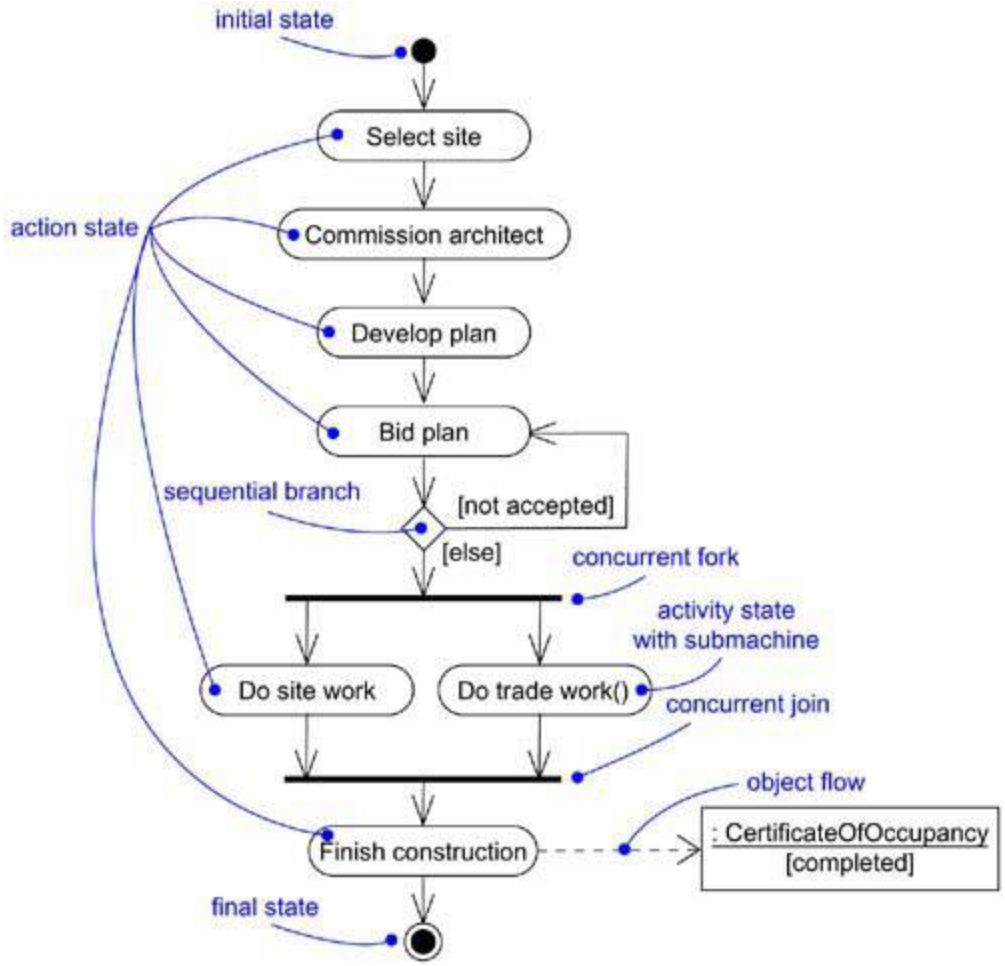

Figure 1: activity diagram

Action States

- The executable, atomic computations are called action states because they are states of the system, each representing the execution of an action
- Figure 2 Action States
- action states can't be decomposed
- action states are atomic, meaning that events may occur, but the work of the action state is not interrupted
- action state is considered to take insignificant execution time
- action states are special kinds of states in a state machine

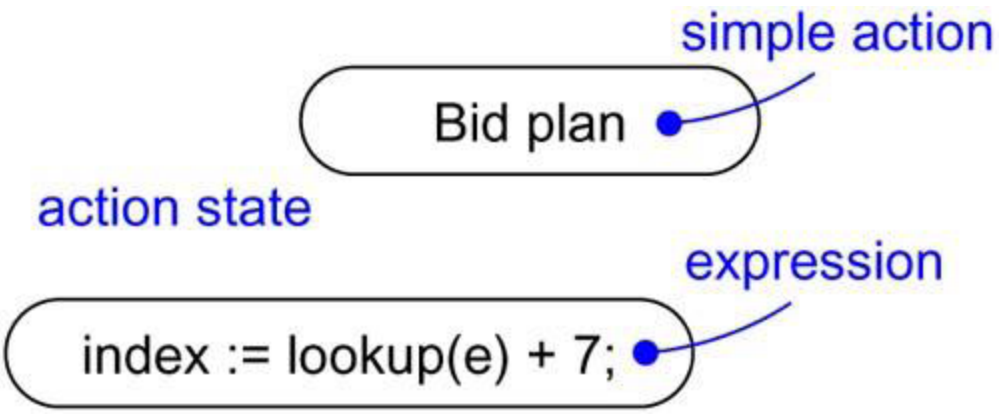

Figure 2: Action States

Activity States

- activity states can be further decomposed
- activity states are not atomic, meaning that they may be interrupted
- they take some duration to complete
- Figure 3 Activity States
- are just special kinds of states in a state machine

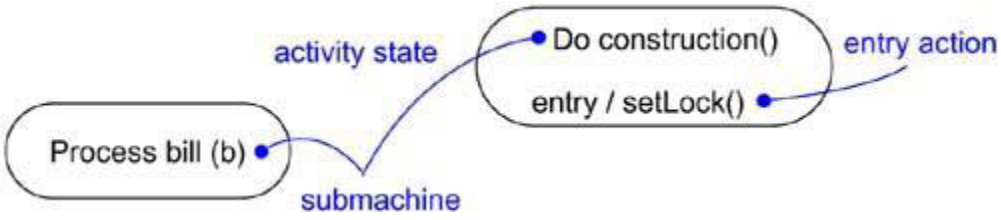

Figure 3: Activity States

Transitions

- transitions shows the path from one action or activity state to the next action or activity state
- a transition is represented as a simple directed line

Triggerless Transitions

- Figure 4
- Triggerless Transitions are transitions where control passes immediately once the work of the source state is done

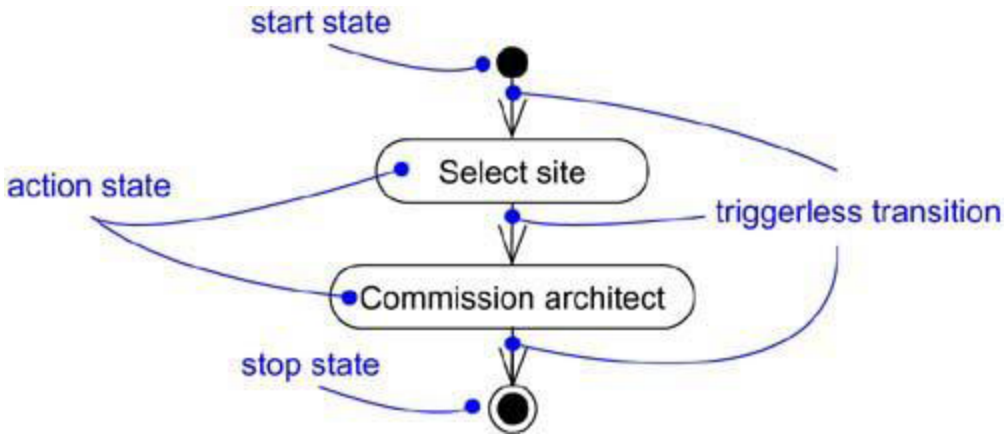

Figure 4: Triggerless Transitions

Branching

- represent a branch as a diamond
- A branch may have one incoming transition and two or more outgoing ones
- each outgoing transition contains a guard expression, which is evaluated only once on entering the branch
- Figure 5: Branching

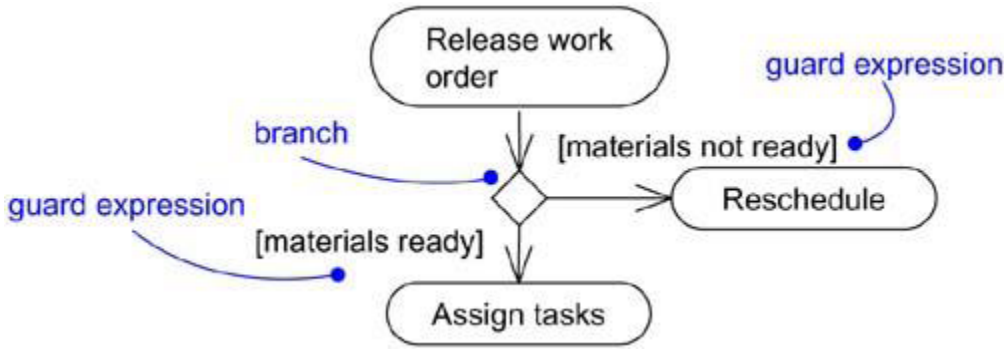

Figure 5: Branching

Forking and Joining

- A *fork* may have one incoming transition and two or more outgoing transitions each of which represents an independent flow of control
- a fork represents the splitting of a single flow of control into two or more concurrent flows of control
- Below the fork, the activities associated with each of these paths continues in parallel
- A *join* may have two or more incoming transitions and one outgoing transition

- Above the join, the activities associated with each of these paths continues in parallel
- At the join, the concurrent flows synchronize, meaning that each waits until all incoming flows have reached the join, at which point one flow of control continues on below the join
- the forking and joining of the parallel flows of control are specified by a *synchronization bar*
- A synchronization bar is rendered as a thick horizontal or vertical line
- Figure 6: Forking and Joining

  Joins and forks should balance, meaning that the number of flows that leave a fork should match the number of flows that enter its corresponding join.

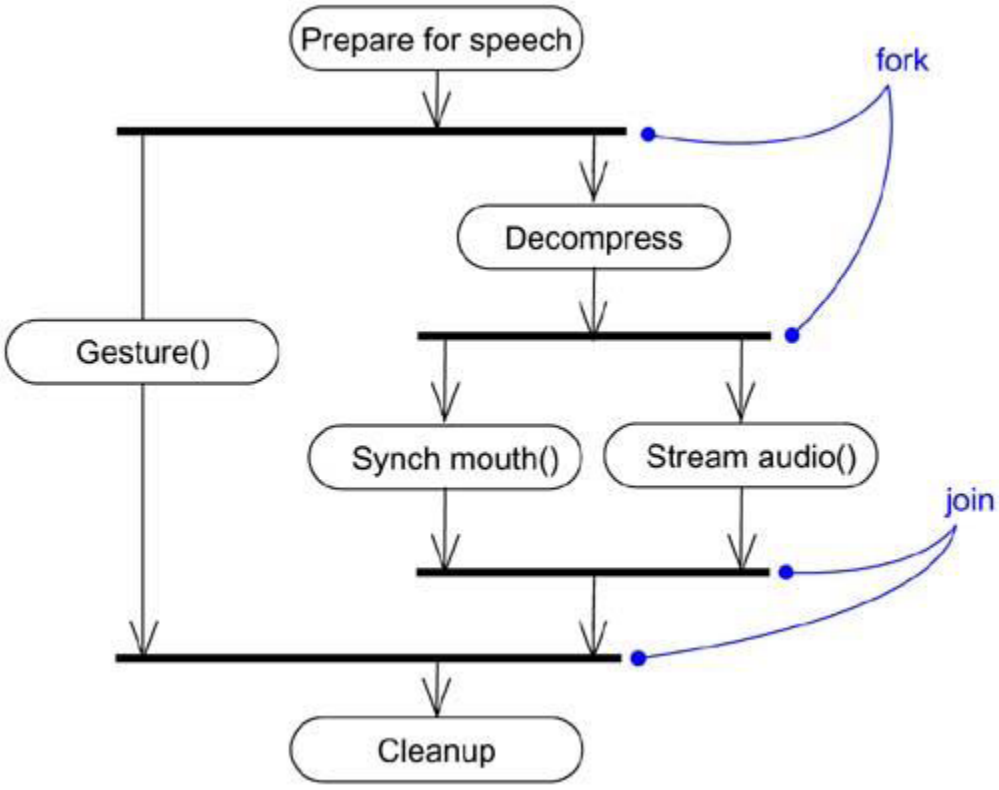

Figure 6: Forking and Joining

Swimlanes

- swimlanes partitions activity diagrams into groups having activity states where each group represents the business organization responsible for those activities
- Figure 7:Swimlanes
- Each swimlane has a name unique within its diagram

- swimlane represents a high-level responsibility for part of the overall activity of an activity diagram
- each swimlane is implemented by one or more classes

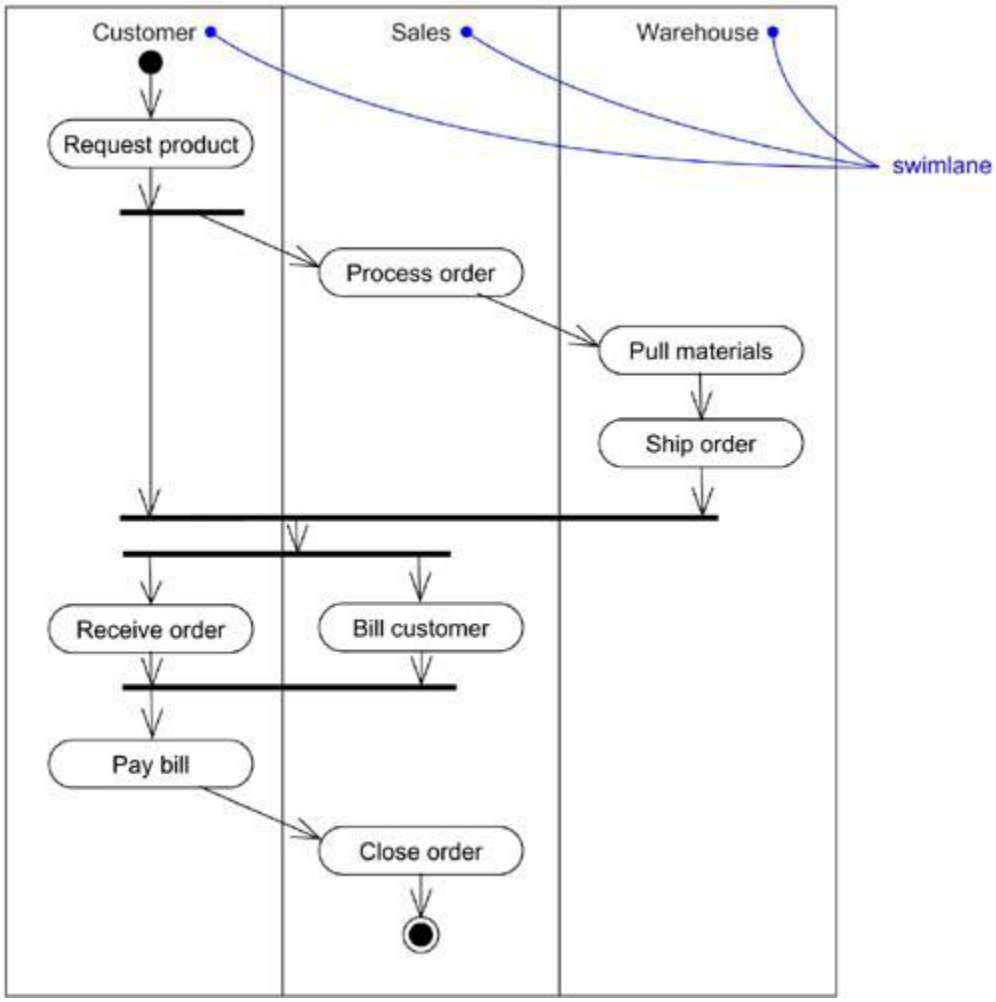

Figure 7: Swimlanes

Object Flow

- object flow indicates the participation of an object in a flow of control, it is represented with the help of dependency relationships.
- Figure 8: Object Flow

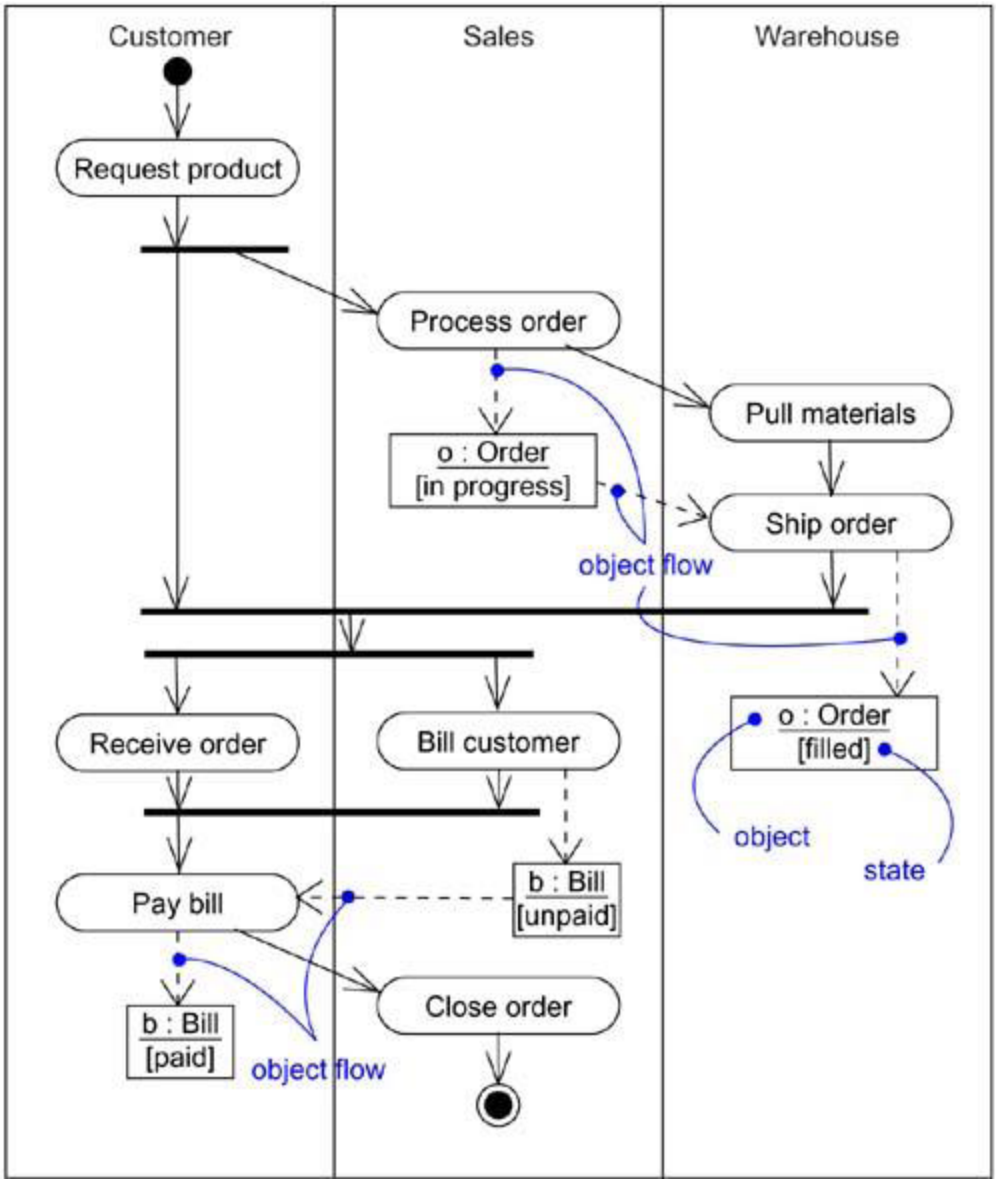

Figure 8: Object Flow

Activity diagrams are used to model dynamic aspects of a system and for this we need activity diagrams,

To model a workflow

To model an operation

Modeling a Workflow

To model a workflow,

- Establish a focus for the workflow. For nontrivial systems, it’s impossible to show all interesting workflows in one diagram.
- Select the business objects that have the high-level responsibilities for parts of the overall workflow. These may be real things from the vocabulary of the system, or they may be more abstract. In either case, create a swimlane for each important business object.
- Identify the preconditions of the workflow’s initial state and the postconditions of the workflow’s final state. This is important in helping you model the boundaries of the workflow.
- Beginning at the workflow’s initial state, specify the activities and actions that take place over time and render them in the activity diagram as either activity states or action states.
- For complicated actions, or for sets of actions that appear multiple times, collapse these into activity states, and provide a separate activity diagram that expands on each.
- Render the transitions that connect these activity and action states. Start with the sequential flows in the workflow first, next consider branching, and only then consider forking and joining.
- If there are important objects that are involved in the workflow, render them in the activity diagram, as well. Show their changing values and state as necessary to communicate the intent of the object flow.

  For example, Figure 9 shows an activity diagram for a retail business, which specifies the workflow involved when a customer returns an item from a mail order.

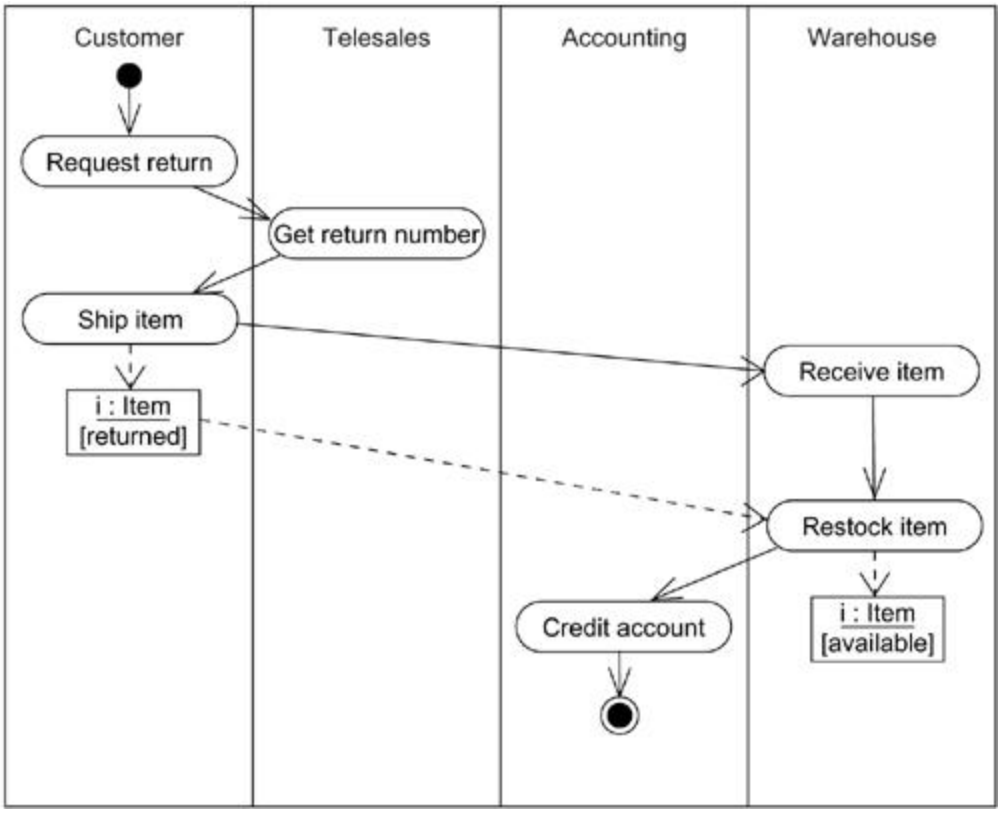

Figure 9: Modeling a Workflow

Modeling an Operation

To model an operation,

- Collect the abstractions that are involved in this operation. This includes the operation's parameters (including its return type, if any), the attributes of the enclosing class, and certain neighboring classes.
- Identify the preconditions at the operation's initial state and the post conditions at the operation's final state. Also identify any invariants of the enclosing class that must hold during the execution of the operation.
- Beginning at the operation's initial state, specify the activities and actions that take place over time and render them in the activity diagram as either activity states or action states.
- Use branching as necessary to specify conditional paths and iteration.
- Only if this operation is owned by an active class, use forking and joining as necessary to specify parallel flows of control.

Figure 10 shows an activity diagram that specifies the algorithm of the operation intersection b/w lines.

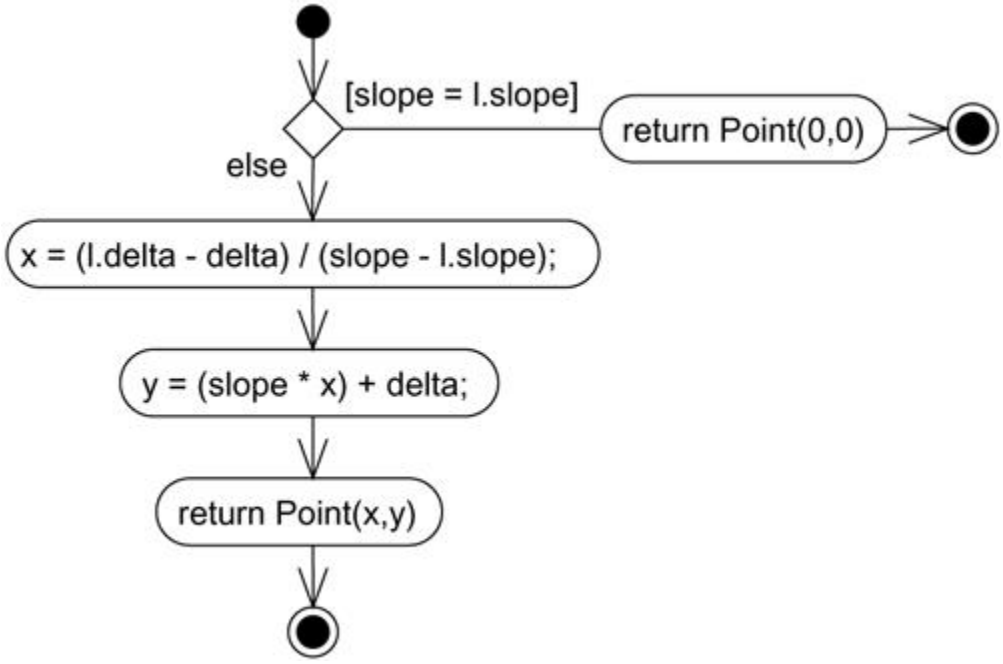

Figure 10: Modeling an Operation

Source : http://praveenthomasln.wordpress.com/2012/04/05/activity-diagrams-s8-cs/