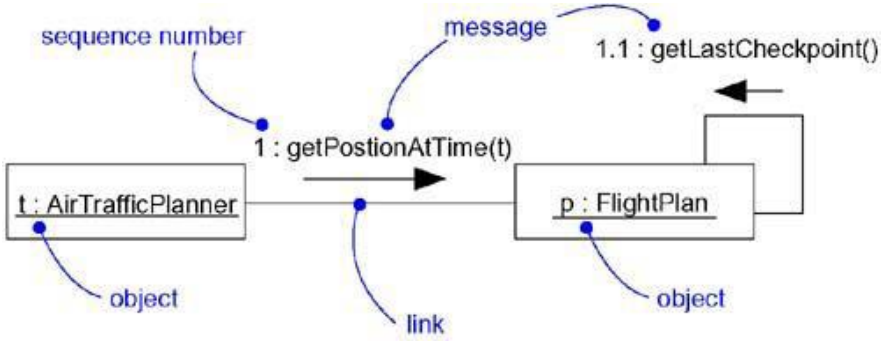
Interactions

- An interaction is a behavior that is composed of a set of messages exchanged among a set of objects within a context to accomplish a purpose.
- A message specifies the communication between objects for an activity to happen. It has following parts: its name, parameters (if any), and sequence number.
- Objects in an interaction can be concrete things or prototypical things.



- ***A link*** is a semantic connection(path) among objects through which a message/s can be send. A link is an instance of an association. The semantics of link can be enhanced by using following prototypes as adornments

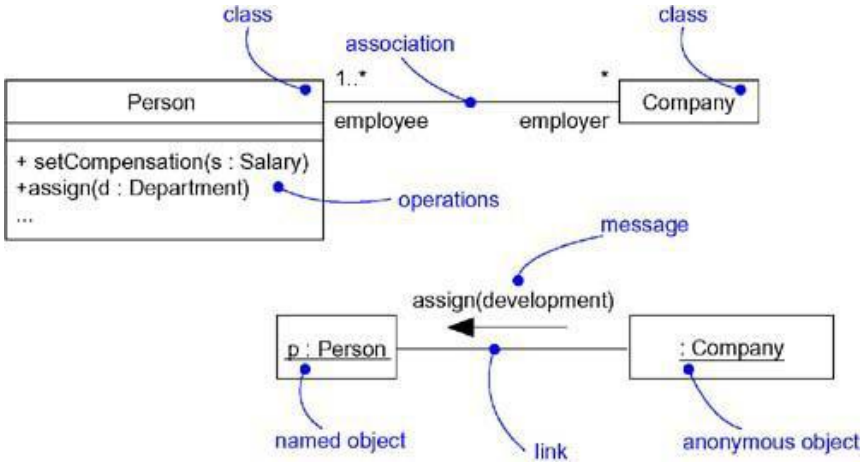
  <<association>> – Specifies that the corresponding object is visible by association

  <<self>> – Specifies that the corresponding object is visible because it is the dispatcher of the operation

  <<global>> – Specifies that the corresponding object is visible because it is in an enclosing scope

  <<local>> – Specifies that the corresponding object is visible because it is in a local scope

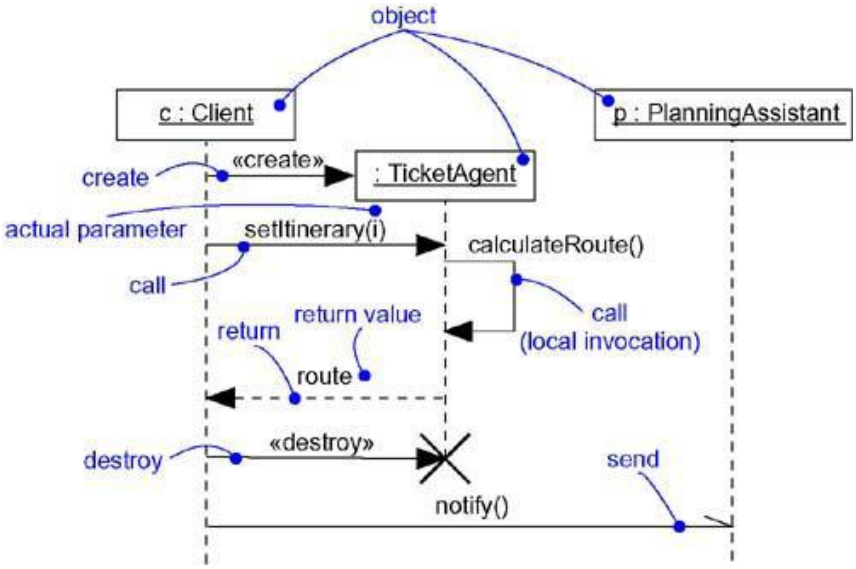
  <<parameter>> – Specifies that the corresponding object is visible because it is a parameter



message – indicates an action to be done. Complex expressions can be written on arbitrary string of message.Different types of messages are:

- Call -Invokes an operation on an object; an object may send a message to itself, resulting in the local invocation of an operation
- Return – Returns a value to the caller

- Send – Sends a signal to an object
- Create – Creates an object
- Destroy – Destroys an object; an object may commit suicide by destroying itself



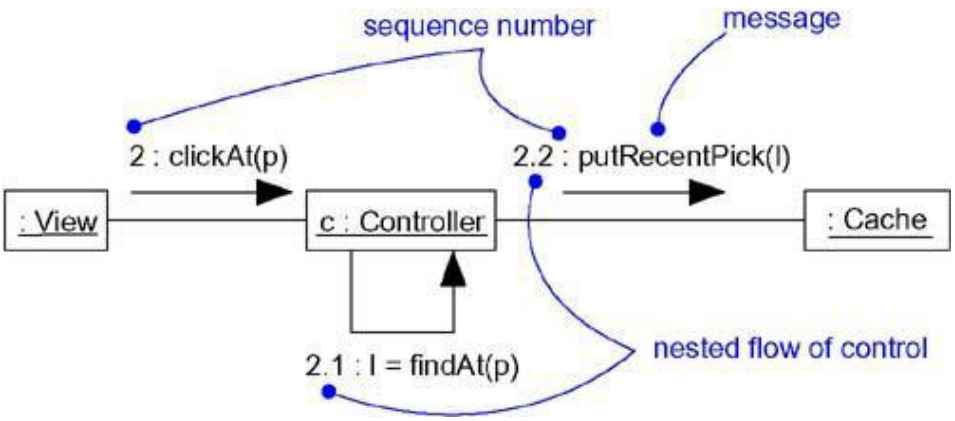
Sequencing

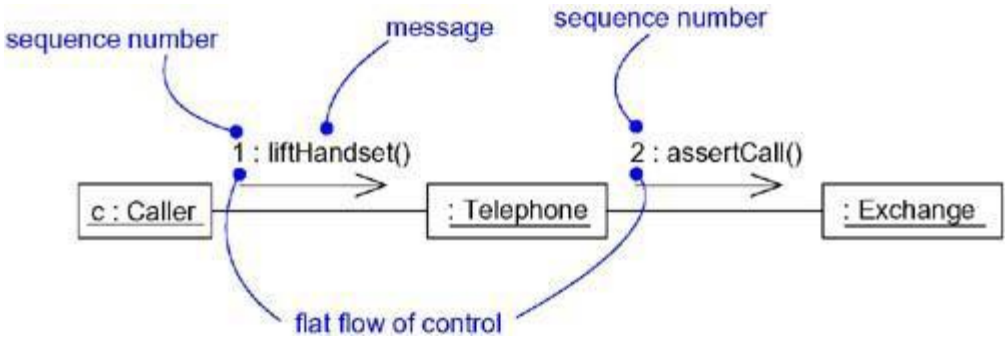
- a sequence is a stream of messages exchange between objects
- sequence must have a beginning and is rooted in some process or thread
- sequence will continue as long as the process or thread that owns it lives

Flow of control (2 types)

In each flow of control, messages are ordered in sequence by time and are visualized by prefixing the message with a sequence number set apart by a colon separator

- A *procedural or nested flow of control* is rendered by using a filled solid arrowhead,
- A *flat flow of control* is rendered by using a stick arrowhead
- *Distinguishing one flow of control from another* by prefixing a message's sequence number with the name of the process or thread that sits at the root of the sequence.
- more-complex forms of sequencing, such as iteration, branching, and guarded messages can be modeled in UML.





Creation, Modification, and Destruction of links

Enabled by *adding the following constraints to the element*

- new – Specifies that the instance or link is created during execution of the enclosing interaction
- destroyed – Specifies that the instance or link is destroyed prior to completion of execution of the enclosing interaction
- transient – Specifies that the instance or link is created during execution of the enclosing interaction but is destroyed before completion of execution

Representation of interactions

interaction goes together with objects and messages.

represented by time ordering of its messages (sequence diagram), and by emphasizing the structural organization of these objects that send and receive messages (collaboration diagram)

Common Modeling Techniques

Modeling a Flow of Control

To model a flow of control,

- Set the context for the interaction, whether it is the system as a whole, a class, or an individual operation
- Set the stage for the interaction by identifying which objects play a role; set their initial properties, including their attribute values, state, and role
- If model emphasizes the structural organization of these objects, identify the links that connect them, relevant to the paths of communication that take place in this interaction
- In time order, specify the messages that pass from object to object As necessary, distinguish the different kinds of messages; include parameters and return values to convey the necessary detail of this interaction
- Also to convey the necessary detail of this interaction, adorn each object at every moment in time with its state and role

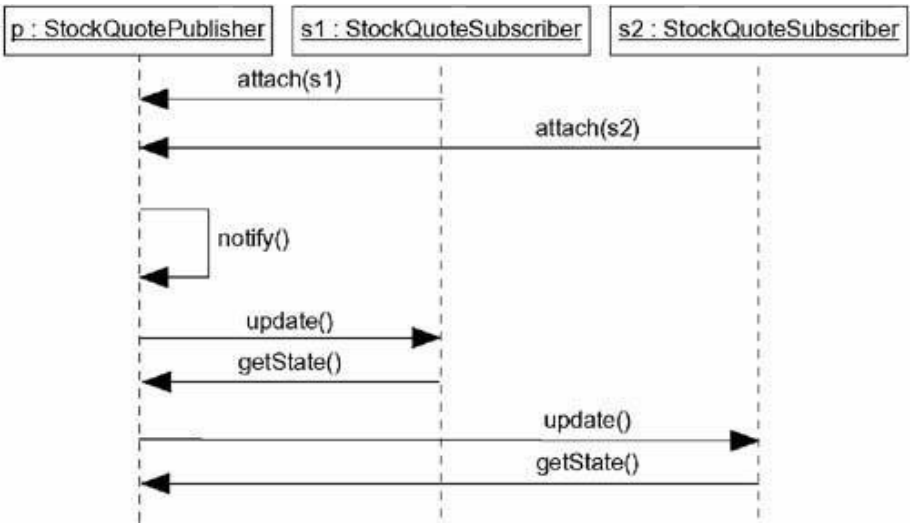


Figure 6: Flow of Control by time

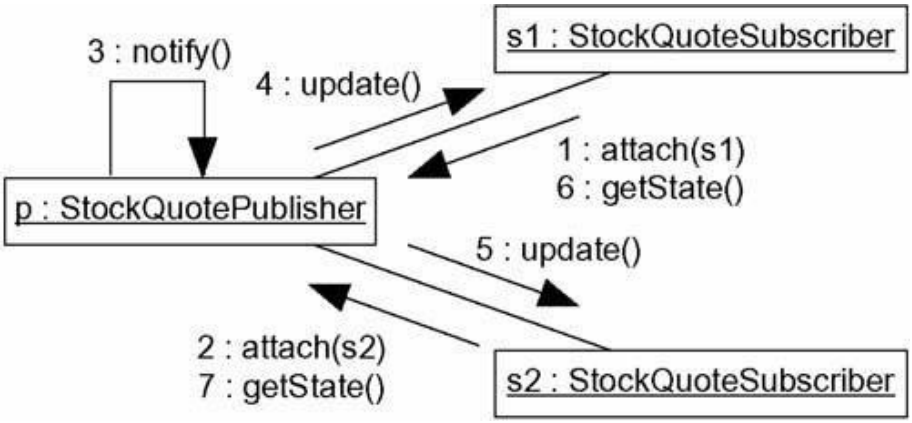


Figure 7: Flow of Control by Organization

Interaction Diagrams

- An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them
- Interaction diagrams commonly contain Objects, Links, Messages
- interaction diagrams are used to model the dynamic aspects of a system
- An interaction diagram is basically a projection of the elements found in an interaction.
- It may contain notes and constraints

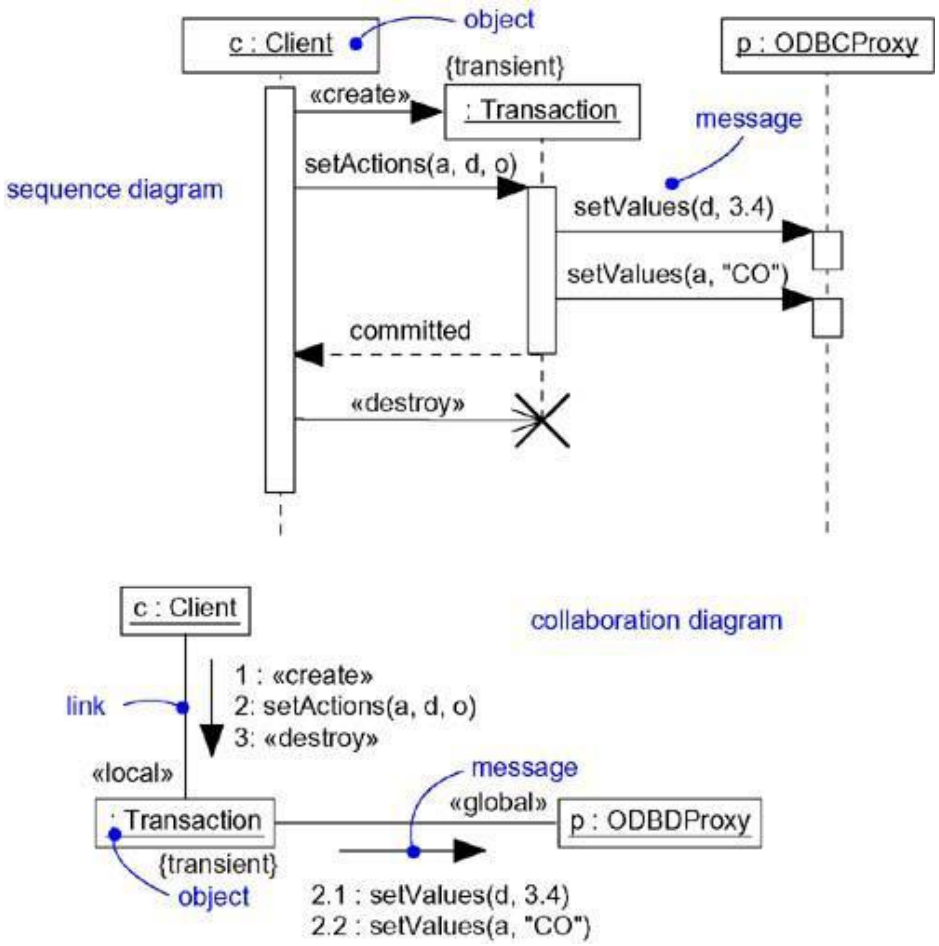


Fig:Interaction Diagrams

Sequence Diagrams

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages
- Graphically it is a table that shows objects arranged along the X axis and messages ordered in increasing time along the Y axis
- place the objects that participate in the interaction at the top of your diagram, across the X axis, object that initiates the interaction at the left, and increasingly more subordinate objects to the right
- place the messages that these objects send and receive along the Y axis, in order of increasing time from top to bottom

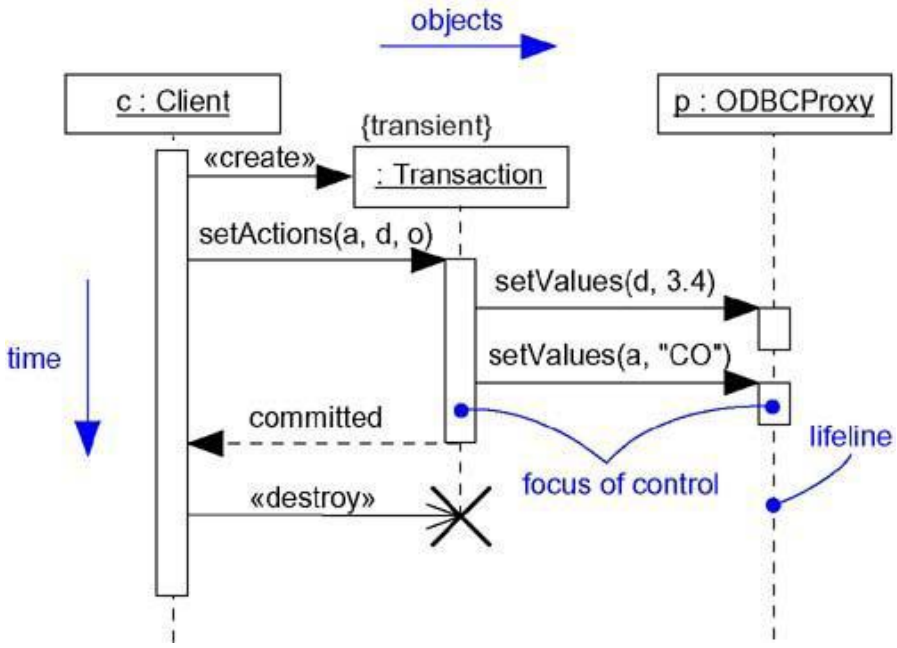


Figure: Sequence Diagram

Sequence diagrams* have *two features that distinguish* them from *collaboration diagrams

- First, there is the *object lifeline* which is a vertical dashed line that represents the existence of an object over a period of time
- Second, there is the *focus of control* which is a tall, thin rectangle that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure

Collaboration Diagrams

- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages
- Graphically it is a collection of vertices and arcs
- more-complex flows, involving iterations and branching are modeled as [i := 1n] (or just), [x > 0]

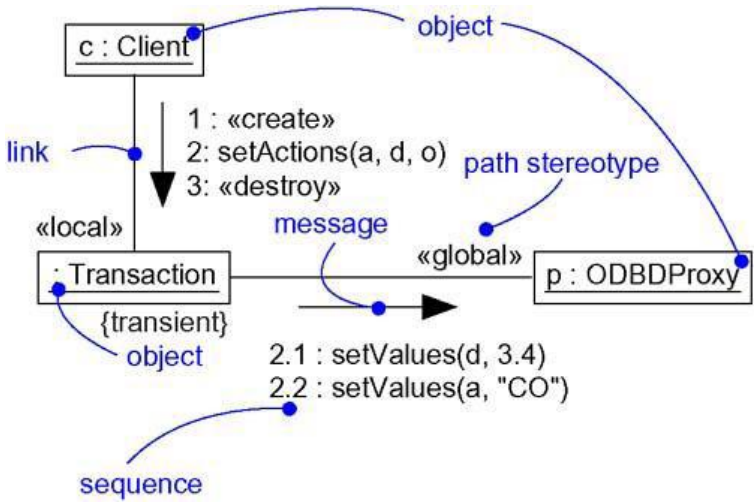


Figure : Collaboration Diagram

Collaboration diagrams* have two *features that distinguish* them from *sequence diagrams

- First, there is the *path* to indicate how one object is linked to another, attach a path stereotype to the far end of a link such as local, parameter, global, and self
- Second, there is the *sequence number* to indicate the time order of a message denoted by prefixing the message with a number, nesting is indicated by Dewey decimal numbering (eg:- 1 is the first message; 1.1 is the first message nested in message 1.)

Semantic Equivalence

sequence diagrams and collaboration diagrams are semantically equivalent that means conversion to the other is possible without any loss of information.

Common Modeling Techniques

Modeling Flows of Control by Time Ordering

To model a flow of control by time ordering,

- Set the context for the interaction, whether it is a system, subsystem, operation, or class or one scenario of a use case or collaboration
- Set the stage for the interaction by identifying which objects play a role in the interaction.
- Set the lifeline for each object. Objects will persist through the entire interaction. For those objects that are created and destroyed during the interaction, set their lifelines, as appropriate, and explicitly indicate their birth and death with appropriately stereotyped messages
- Starting with the message that initiates this interaction, lay out each subsequent message from top to bottom between the lifelines, showing each message's properties .
- If you need to visualize the nesting of messages or the points in time when actual computation is taking place, adorn each object's lifeline with its focus of control
- If you need to specify time or space constraints, adorn each message with a timing mark and attach suitable time or space constraints
- If you need to specify this flow of control more formally, attach pre- conditions and post-conditions to each message

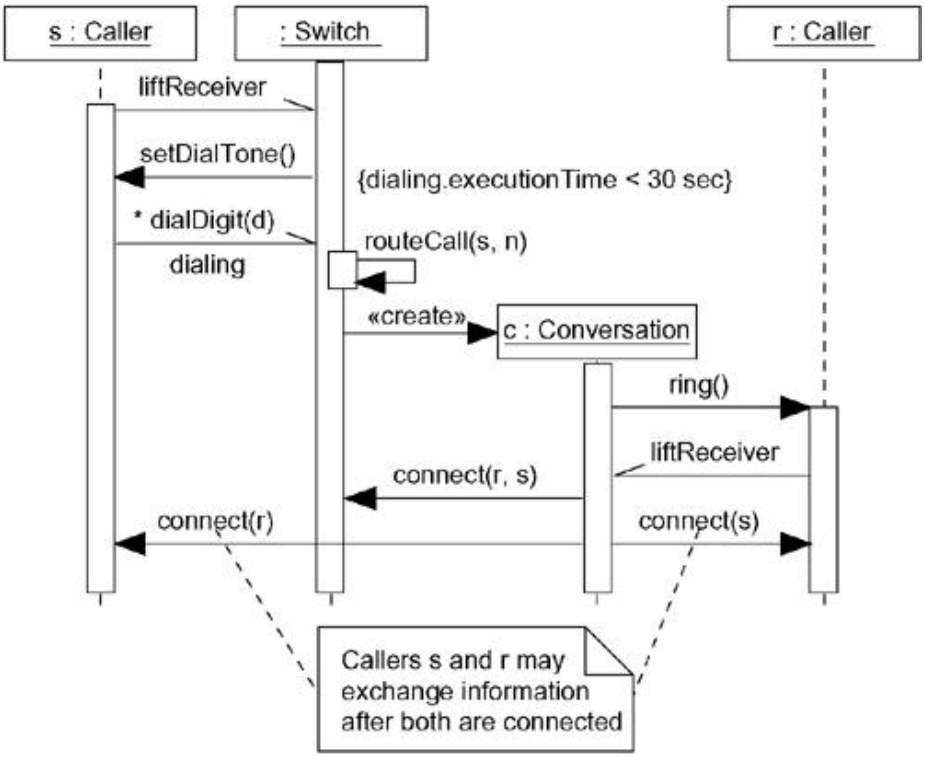


Figure : Modeling Flows of Control by Time Ordering

Modeling Flows of Control by Organization

To model a flow of control by organization

- Set the context for the interaction, whether it is a system, subsystem, operation, or class or one scenario of a use case or collaboration
- Set the stage for the interaction by identifying which objects play a role in the interaction .
- Set the initial properties of each of these objects If the attribute values, tagged values, state or role of any object changes in significant ways over the duration of the interaction, place a duplicate object on the diagram, update it with these new values, and connect them by a message stereotyped as become or copy .
- Specify the links among these objects, along which messages may pass
  1. Lay out the association links first; these are the most important ones, because they represent structural connections
  2. Lay out other links next, and adorn them with suitable path stereotypes (such as global and local) to explicitly specify how these objects are related to one another
- Starting with the message that initiates this interaction, attach each subsequent message to the appropriate link, setting its sequence number, as appropriate Show nesting by using Dewey decimal numbering.
- If you need to specify time or space constraints, adorn each message with a timing mark and attach suitable time or space constraints.
- If you need to specify this flow of control more formally, attach pre- and post-conditions to each message

The Figure shows a collaboration diagram that specifies the flow of control involved in registering a new student at a school, with an emphasis on the structural relationships among these objects

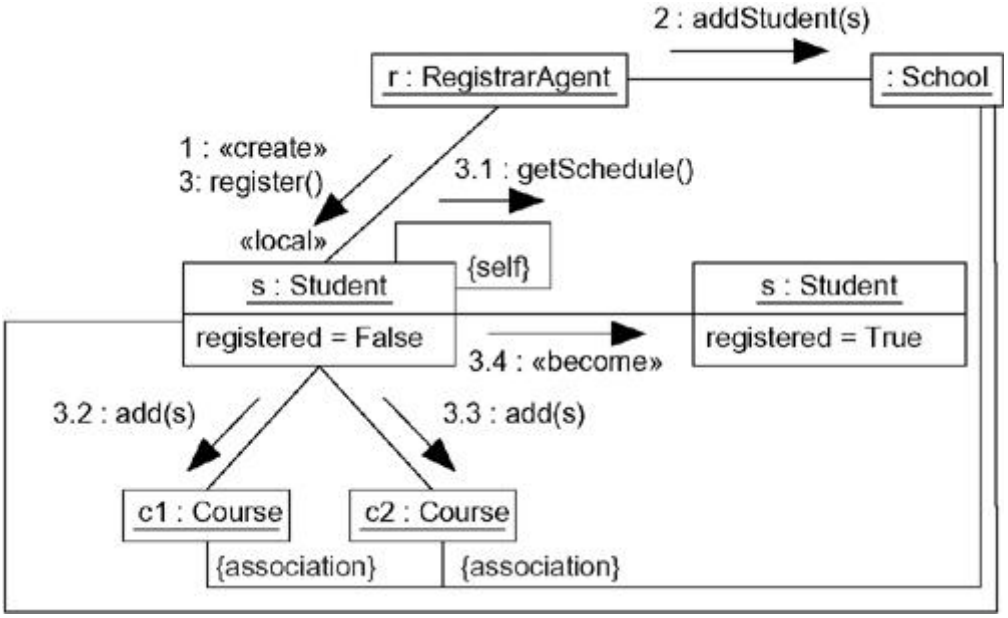


Figure 5: Modeling Flows of Control by Organization

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

So the purposes of interaction diagram can be describes as:

- To capture dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe structural organization of the objects.
- To describe interaction among objects.

Use Cases

- use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor
- use case captures the intended behavior of the system (or subsystem, class, or interface) without having to specify how that behavior is implemented
- a use case is represented as an ellipse
- Every use case must have a name that distinguishes it from other use cases: simple name and path name Figure 2: Simple and Path Names

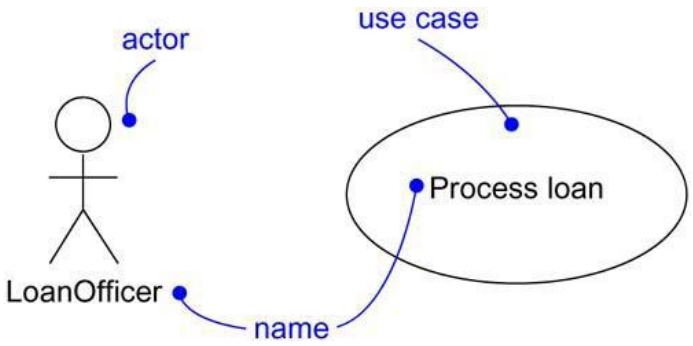


Figure 1: Actors and Use Cases

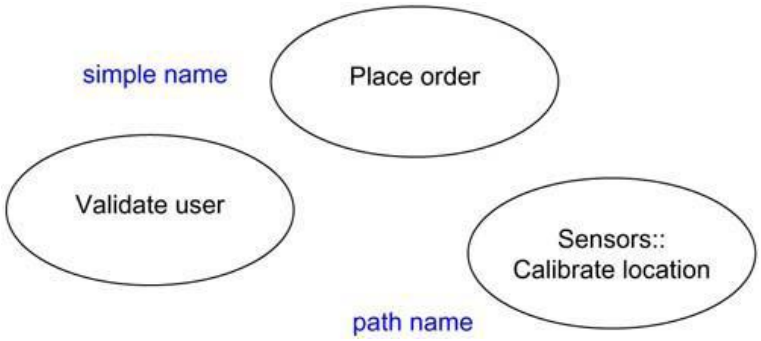


Figure 2: Simple and Path Names

Actors

- actor represents a coherent set of roles that users of use cases play when interacting with these use cases
- an actor represents a role that a human, a hardware device, or even another system plays with a system Figure 3: Actors

Actors may be connected to use cases by association

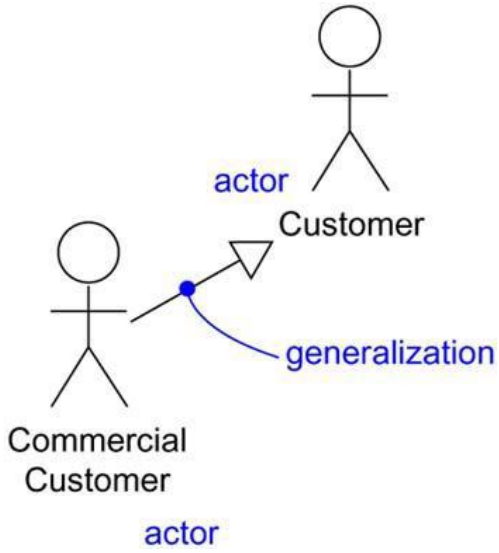


Figure 3: Actors

Use Cases & Flow of Events

flow of events include how and when the use case starts and ends when the use case interacts with the actors and what objects are exchanged, and the basic flow and alternative flows of the behavior.

The behavior of a use case can be specified by describing a flow of events in text.

There can be Main flow of events and one or more Exceptional flow of events.

Use Cases and Scenarios

- A scenario is a specific sequence of actions that illustrates behavior
- Scenarios are to use cases, as instances are to classes means that scenario is basically one instance of a use case
- for each use case, there will be primary scenarios and secondary scenarios.

Use Cases and Collaborations

- Collaborations are used to implement the behavior of use cases with society of classes and other elements that work together
- It includes static and dynamic structure

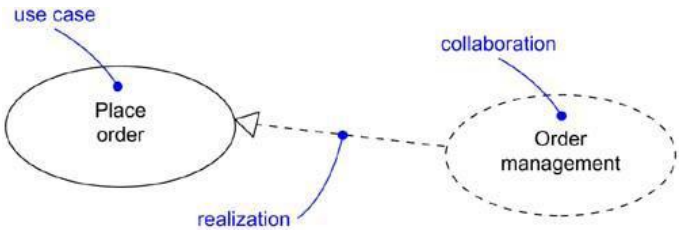


Figure 4: Use Cases and Collaborations

Organizing Use Cases

- organize use cases by grouping them in packages
- organize use cases by specifying generalization, include, and extend relationships among them

Generalization, Include and Extend

An *include relationship* between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.

An include relationship can be rendered as a dependency, stereotyped as include

An *extend relationship* between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case

An extend relationship can be rendered as a dependency, stereotyped as extend.

extension points are just labels that may appear in the flow of the base use case

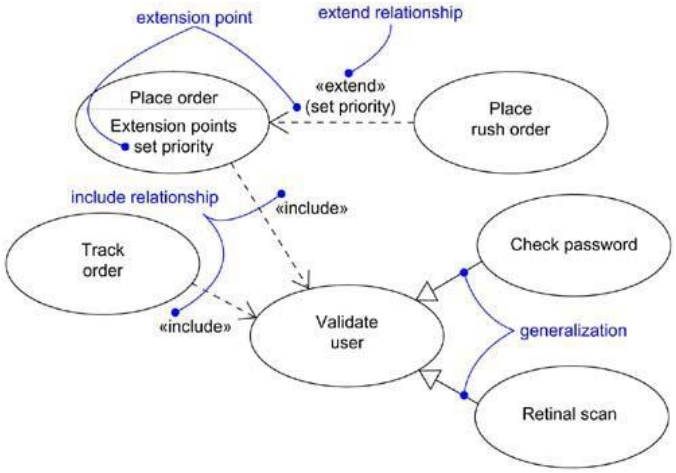


Figure 5: Generalization, Include and Extend

Modeling the Behavior of an Element

To model the behavior of an element,

- Identify the actors that interact with the element Candidate actors include groups that require certain behavior to perform their tasks or that are needed directly or indirectly to perform the element's functions
- Organize actors by identifying general and more specialized roles
- For each actor, consider the primary ways in which that actor interacts with the element Consider also interactions that change the state of the element or its environment or that involve a response to some event
- Consider also the exceptional ways in which each actor interacts with the element
- Organize these behaviors as use cases, applying include and extend

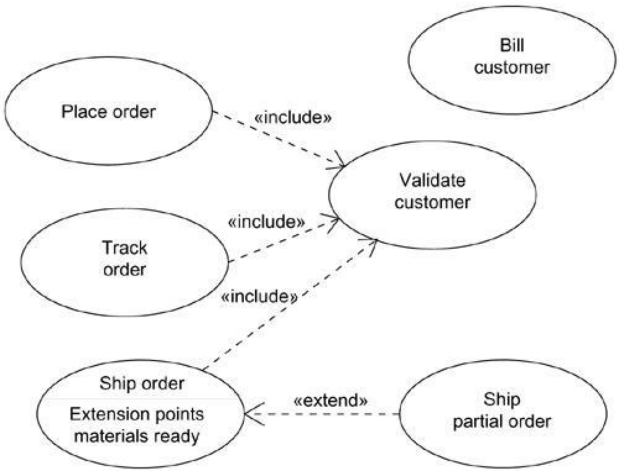


Figure 6: Modeling the Behavior of an Element

Use Case Diagrams

- A use case diagram is a diagram that shows a set of use cases and actors and their relationships
- Use case diagrams commonly contain Use cases, Actors, Dependency, generalization, and association relationships
- use case diagrams may contain packages, certain times instances of use cases, notes and constraints
- apply use case diagrams to model the static use case view of a system by modeling the context of a system and by modeling the requirements of a system

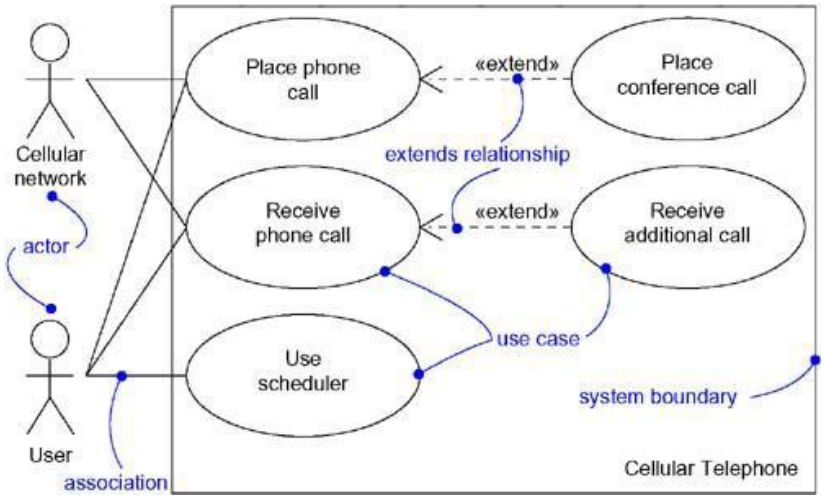


Figure 1: A Use Case Diagram

Modeling the Context of a System

To model the context of a system,

- Identify the actors that surround the system by considering which groups require help from the system to perform their tasks; which groups are needed to execute the system's functions; which groups interact with external hardware or other software systems; and which groups perform secondary functions for administration and maintenance

- Organize actors that are similar to one another in a generalization / specialization hierarchy
- provide a stereotype for each such actor
- Populate a use case diagram with these actors and specify the paths of communication from each actor to the system's use cases

Figure 2 shows the context of a credit card validation system, with an emphasis on the actors that surround the system

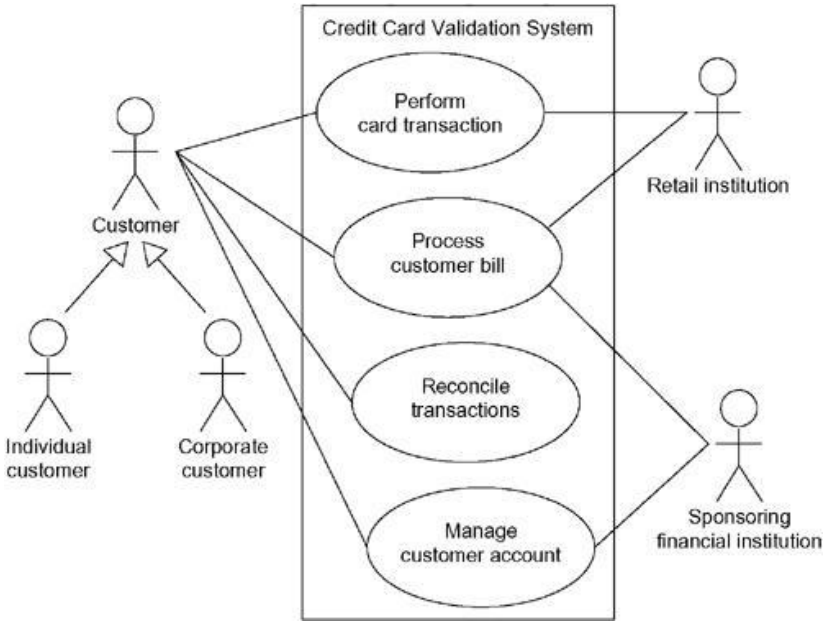


Figure 2: Modeling the Context of a System

Common Modeling Techniques

Modeling the Requirements of a System

To model the requirements of a system,

- Establish the context of the system by identifying the actors that surround it
- For each actor, consider the behavior that each expects or requires the system to provide
- Name these common behaviors as use cases
- Factor common behavior into new use cases that are used by others; factor variant behavior into new use cases that extend more main line flows
- Model these use cases, actors, and their relationships in a use case diagram
- Adorn these use cases with notes that assert nonfunctional requirements; you may have to attach some of these to the whole system

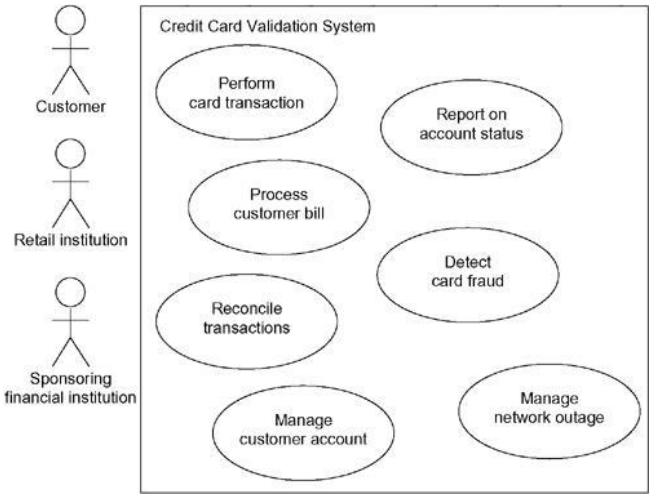


Figure 3: Modeling the Requirements of a System

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)
- 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

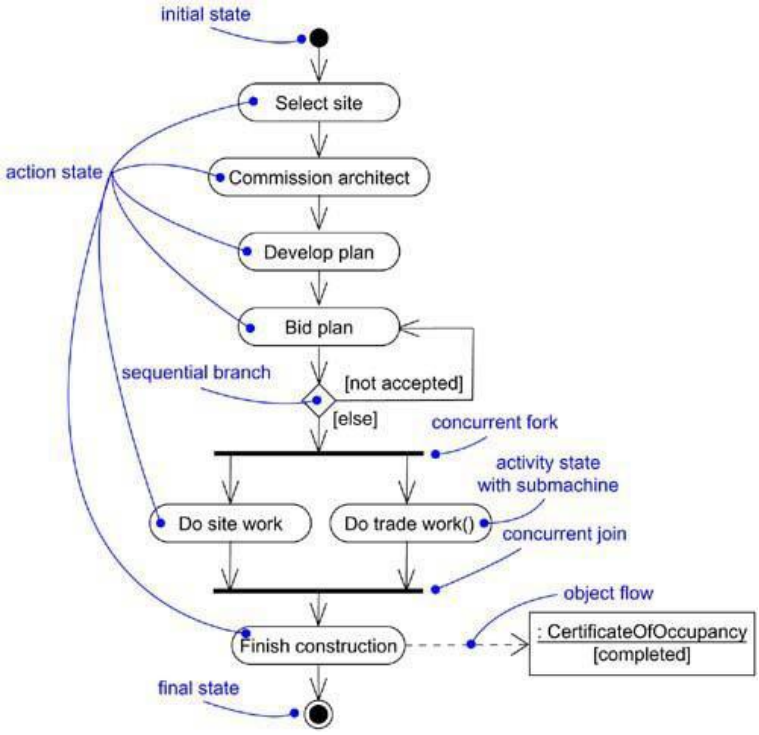


Figure 1: activity diagram

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

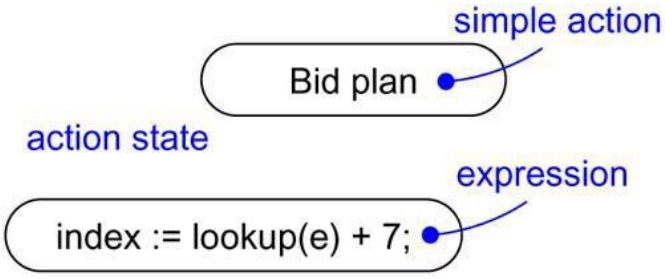


Figure 2: Action States

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
- are just special kinds of states in a state machine

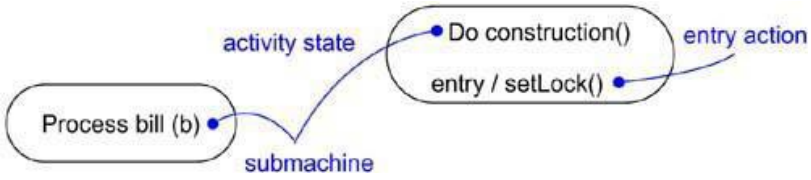


Figure 3: Activity States

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

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

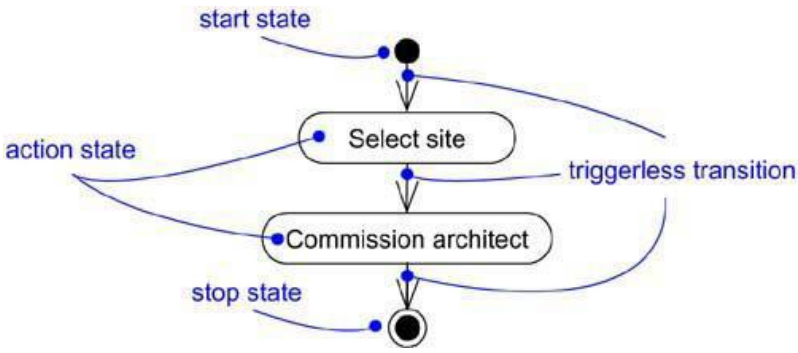


Figure 4: Triggerless Transitions

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

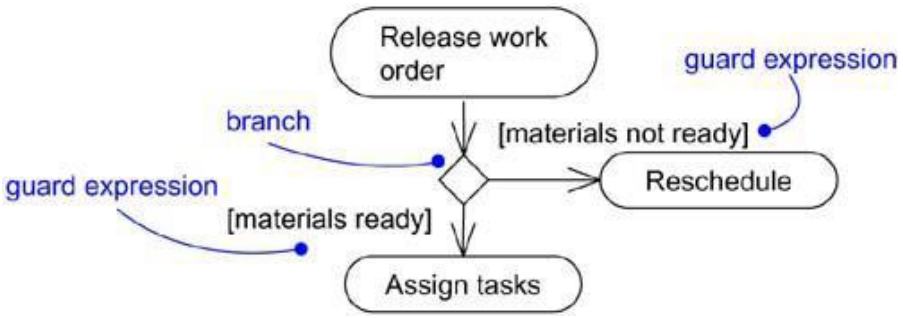


Figure 5: Branching

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

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.

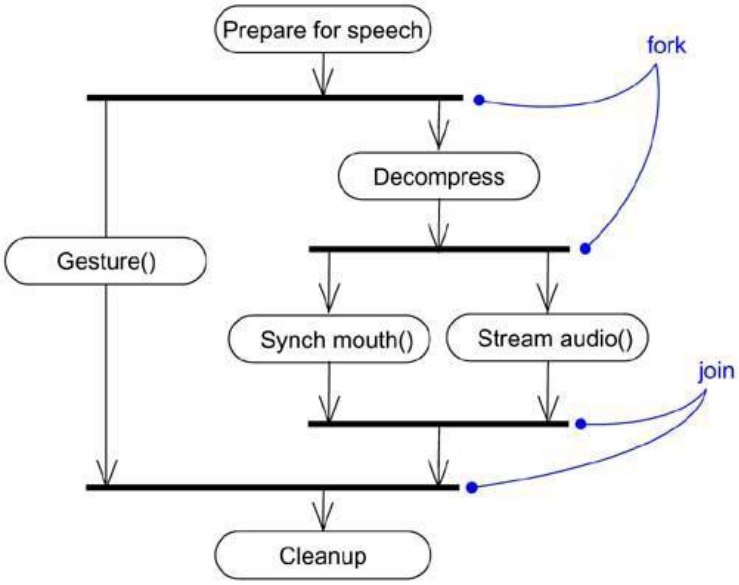


Figure 6: Forking and Joining

Swimlanes

- swimlanes partitions activity diagrams into groups having activity states where each group represents the business organization responsible for those activities
- 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

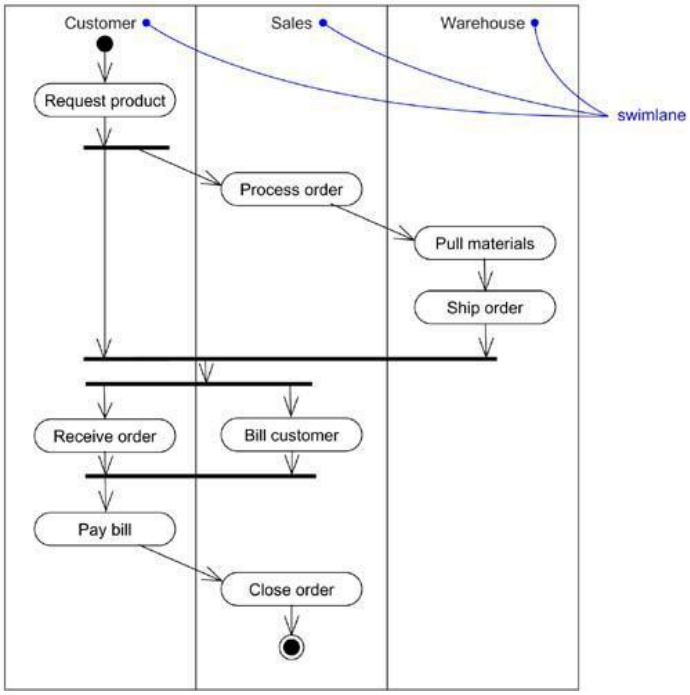


Figure 7: Swimlanes

Object Flow

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

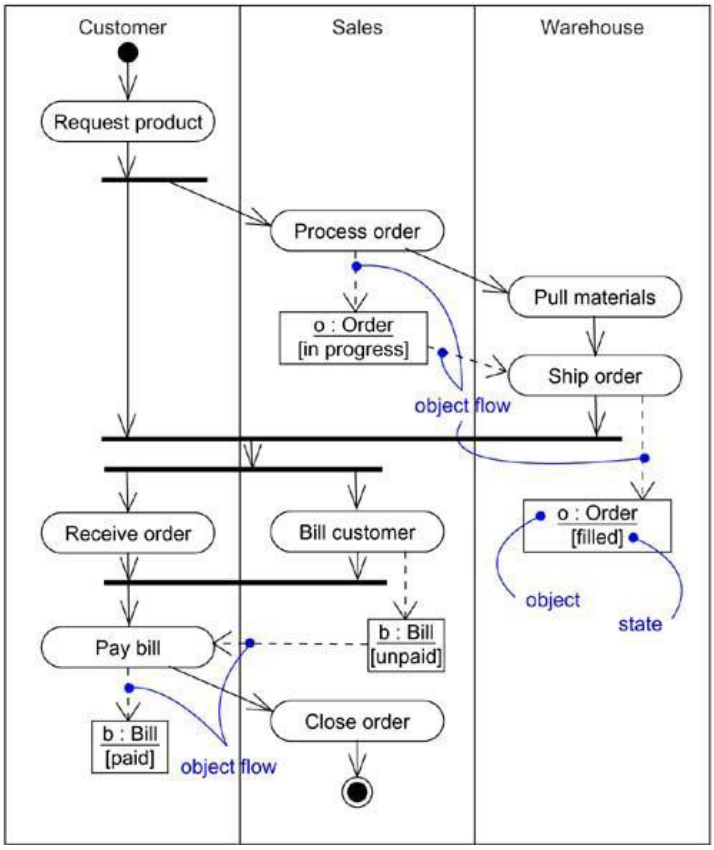


Figure 8: Object Flow

Common Modeling Techniques

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

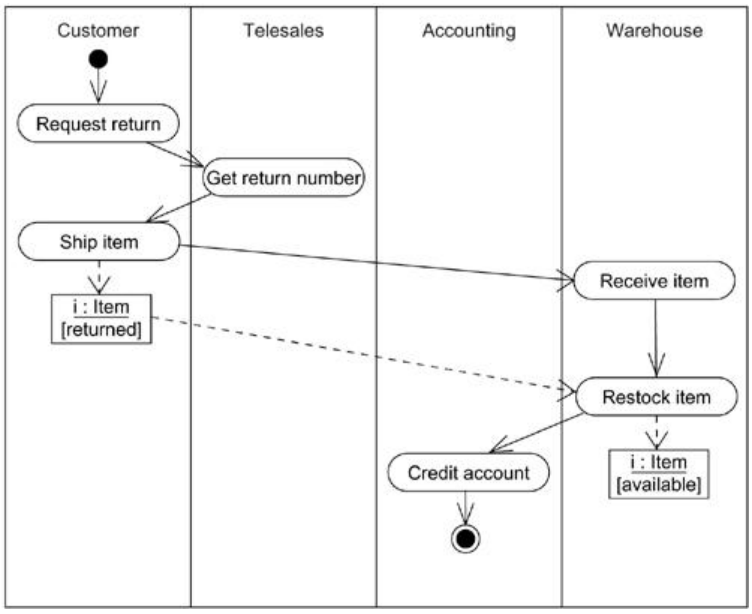


Figure : Modeling a Workflow

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

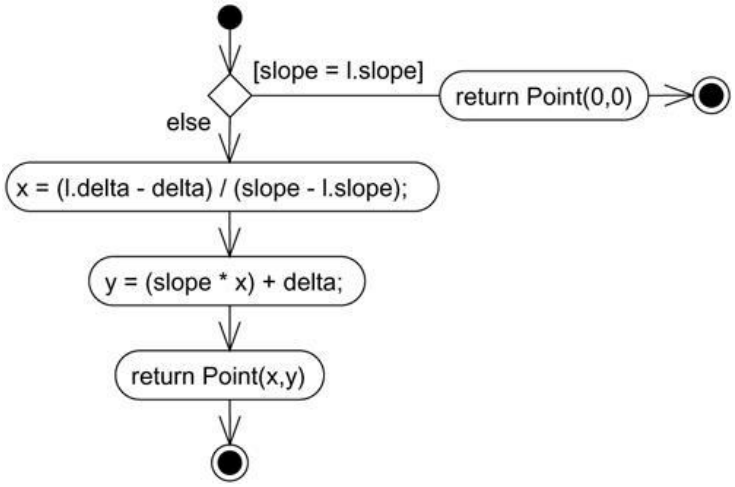


Figure 10: Modeling an Operation