UNIT – IV

BEHAVIOURAL MODELING

Interactions:

→ An interaction is a behaviour that comprises a set of messages exchanged among a set of objects within a context to accomplish a purp

Message :-

→ A message is a specification of a communication between objects that conveys information with the expectation that activity will ensue.

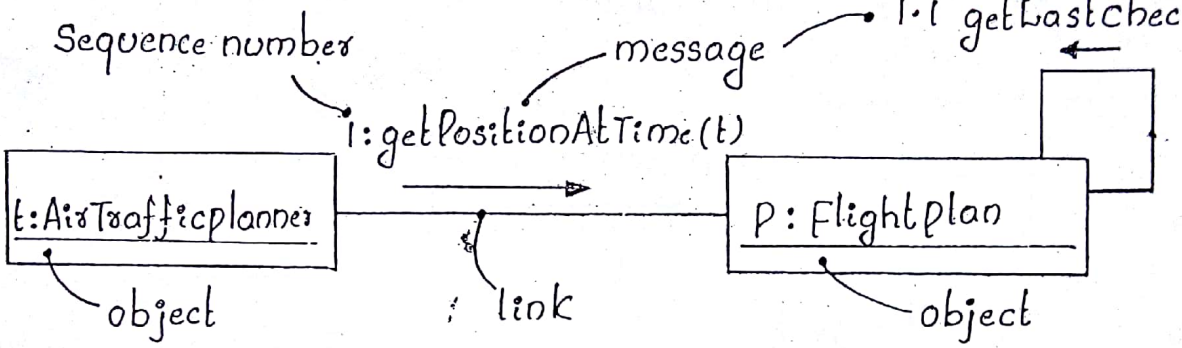


Figure : Messages, links and sequencing

Context:

→ You may find an interaction whereever objects are linked to one anoth
→ You will find interactions in the collaboration of objects that exist in t context of your system or subsystem.
→ You will also find interactions in the context of an operation.
→ Finally, you will find interactions in the context of a class.

Objects and Roles:

→ The objects that participate in an interaction are either concrete thin (or real) or prototypical things.
→ As a concrete thing, an object represents something in the real wor
→ For example, P, an instance of the class Person, might denote a partic Human.
→ Alternately, as a prototypical thing, P might represent any instance o Person.
→ In the context of an interaction, you may find instances of classes, co ponents, nodes and usecases.

Links: and Associations :-

→ A link is a semantic connection among objects.

As shown in figure, whenever a class has an association to another class, there may be a link between the instances of the two classes. Wherever there is a link between two objects, one object can send a message to the other object.

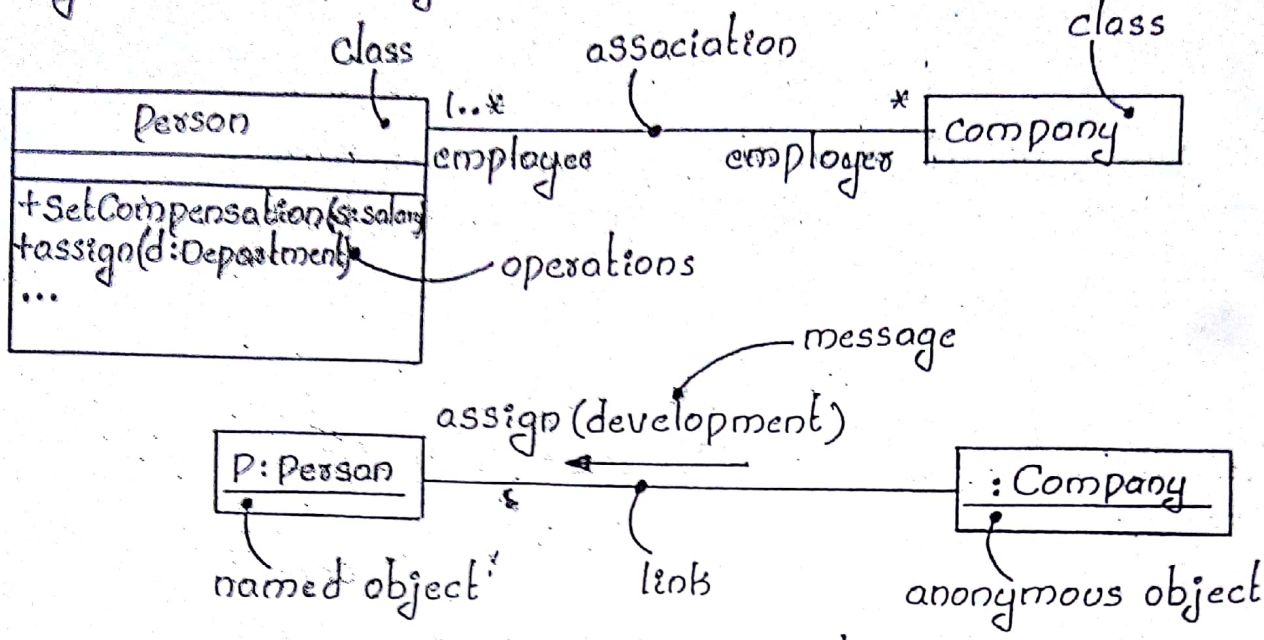
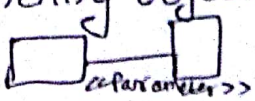


Figure : Links and Associations

- A link specifies a path along which one object can dispatch a message to another (or the same) object.
- Most of the time, it is sufficient to specify that such a path exists.

following are 5 standard stereotypes, They are

- ✓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. 

Messages:

- Suppose you have a set of objects and a set of links that connect those objects.

A message is the specification of a communication among objects that

conveys information with the expectation that activity will ensue (2
The receipt of a message instance may be considered an instance of an
event.

When you pass a message, the action that results is an executable sta
ment that forms an abstraction of a computational procedure.

An action may result in a change in state. e.g:- Alarm, wakeup

In the UML, you can model several kinds of actions

- ✓ 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. notify
- ✓ Create → Creates an object →□
- ✓ Destroy → Destroys an object; an object may commit suicide by destroying itself. →✕

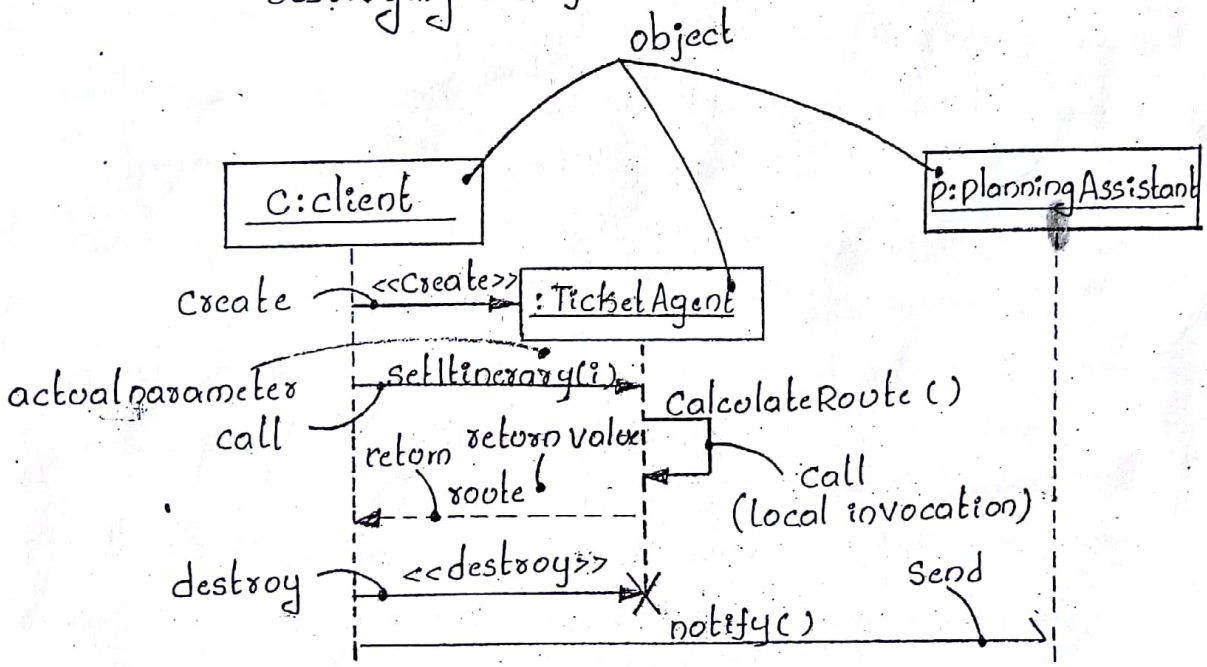


Figure : Messages

Sequencing:

→ When an object passes a message to another object (in effect, delegating some action to the receiver), the receiving object might in turn send a message to another object, which might send a message to yet a different object, and so on.

→ This stream of messages forms a sequence.

→ Any sequence must have a beginning, the start of every sequence is root

in some process or thread.

- Most commonly, you can specify a procedural or nested flow of control rendered using a filled solid arrowhead as shown in figure.

1. Procedured sequence (nested sequence) :→

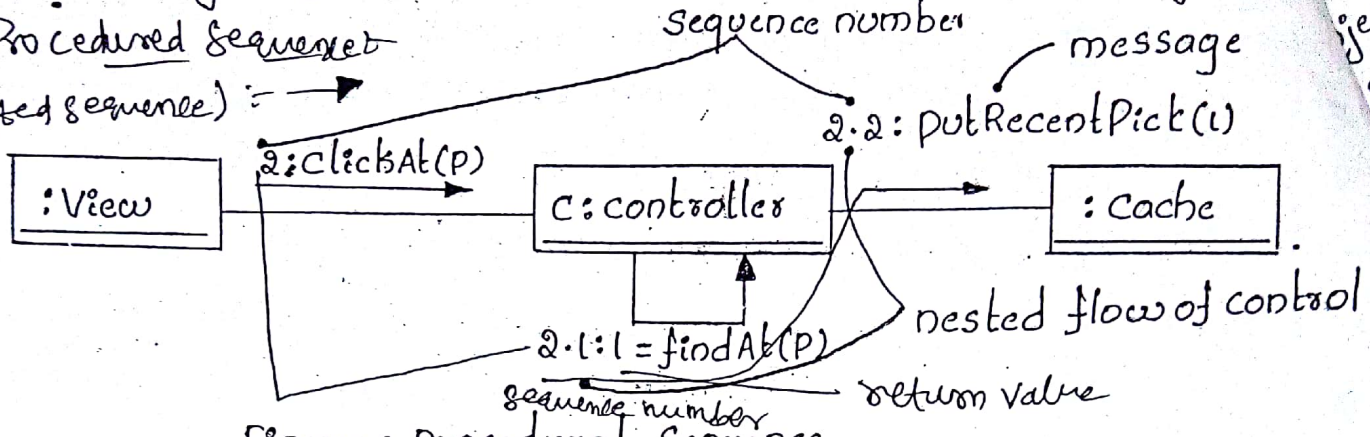


Figure : procedural sequence

2. Non-Procedured sequence

- you can specify a flat flow of control, rendered using a stick arrow head, to model the nonprocedural progression of control from step to ste

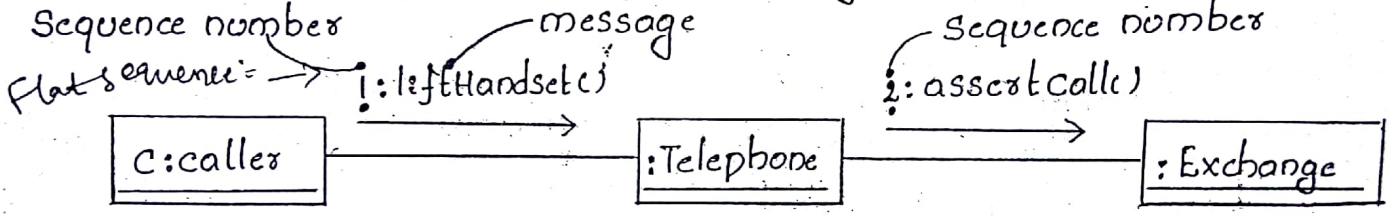


Figure: Flat Sequence

- To specify if an object or link enters and/or leaves during an interaction you can attach one of 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.

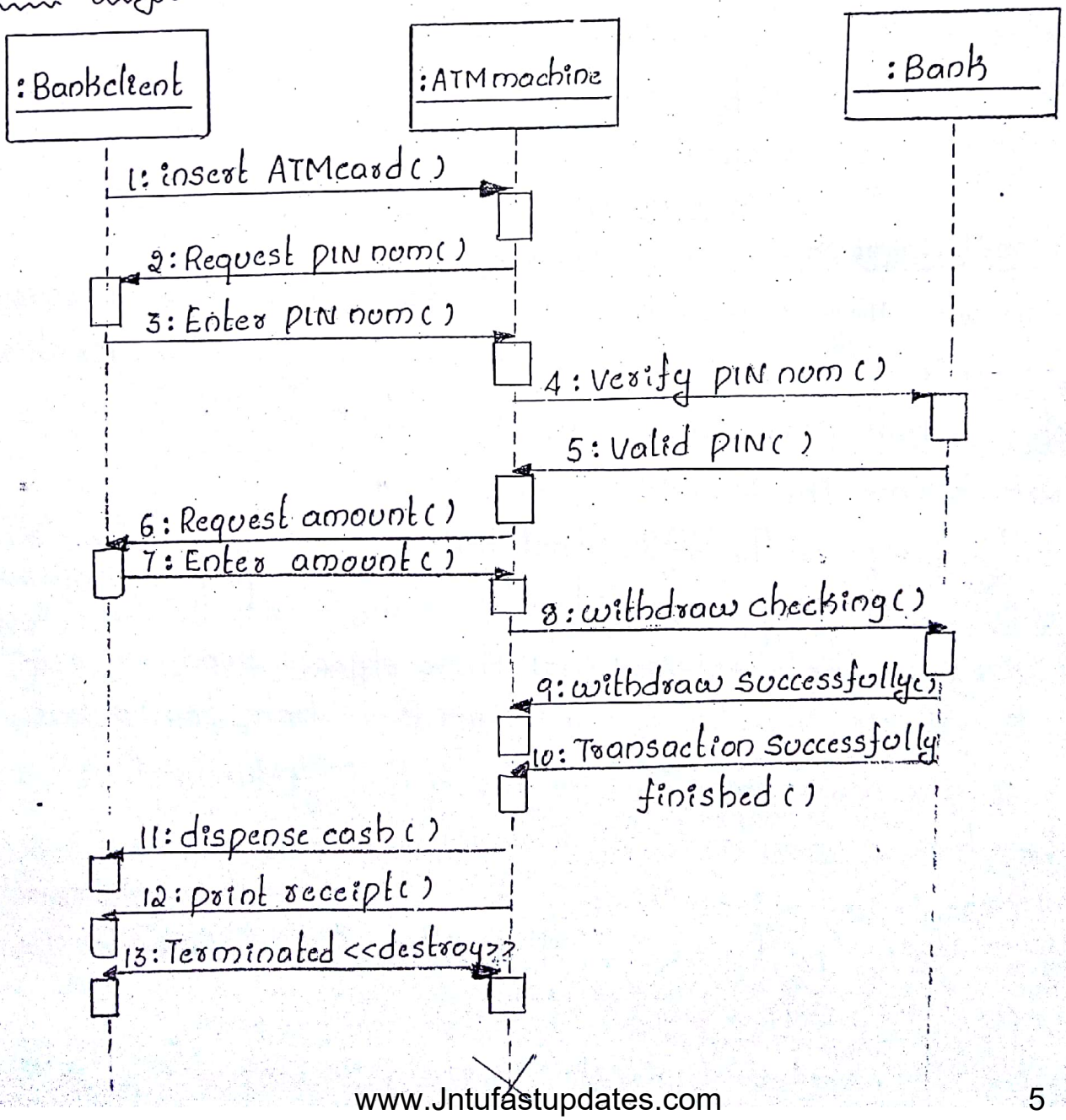
✓ transient → specifies that the instance or link is created during execution of the enclosing interaction but is destroyed before completion of execution

Interaction Diagrams:

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may b dispatched among them.

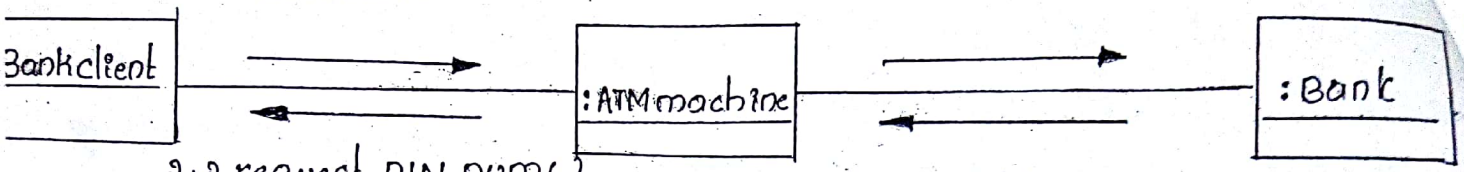
- A sequence diagram is an interaction diagram that emphasizes the lin ordering of messages.
- Graphically, a sequence diagram is a table that shows objects arrang along the x axis and messages, ordered in increasing time along th Y axis.
- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messag
- Graphically, a collaboration diagram is a collection of vertices and arc

Sequence diagram:



Collaboration Diagram:

1:1 insert ATM card()
3:3 enter PIN number()
7:7 enter Amount()
13:13 Terminated



2.2 request PIN num()
6.6 Request Amount()
11.11 dispense cash()
12.12 print receipt()
5.5 Valid pin()
9.9 withdraw successfully()
10.10 Transaction Successfully finished

4.4 : verify pin num()
8.8 : withdraw checking()

- Interaction diagrams commonly contain
  - ✓ objects
  - ✓ Links
  - ✓ Messages

Sequence Diagrams: dia in note

- A Sequence Diagram emphasizes the time ordering of messages.
- As shown in figure, you form a sequence diagram by first placing the objects that participate in the interaction at the top of your diagram, across the x axis.
- Typically, you place the object that initiates the interaction at the left, and increasingly more subordinate objects to the right.
- Next, you place the messages that these objects send and receive along the y axis, in order of increasing time from top to bottom
- Sequence diagrams have two features that distinguish them from collaboration diagrams.

First, there is the object lifeline.
An object lifeline is the vertical dashed line that represents the existence of an object over a period of time.
Their lifelines start with the receipt of the message stereotyped as created

objects may be destroyed during the interaction.
second, there is the focus of control.
The focus of control is a tall, thin rectangle is aligned with the st of the action, the bottom is aligned with its completion (and can be marked by a return message, that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

- The top of the rectangle is aligned with the start of the action, the bottom is aligned with its completion (and can be marked by a retu message).

Collaboration Diagrams: dia in notes

- A collaboration diagram emphasizes the organization of the object that participate in an interaction.
- As shown in figure, you form a collaboration diagram by first placir the objects that participate in the interaction as the vertices in a grap
- Next, you render the links that connect these objects as the arcs of thi graph.
- collaboration diagrams have two features that distinguish them fror sequence diagrams.
- First, there is the path.
- To indicate how one object is linked to another, you can attach a pat stereotype to the far end of a link (such as <<local>>, indicating the the designated object is local to the sender).
- second, there is the sequence number.
- To indicate the time order of a message you prefix the message wit a number (starting with the message numbered 1), increasing monol nically for each new message in the flow of control (2, 3, and so on)
- To show nesting, you use Dewey decimal numbering (1 is the first message; 1-1 is the first message nested in message 1; 1-2 is the second message nested in message 1; and so on).

Use Cases.

- In the UML, all such behaviors are modeled as usecases that are specified independent of their realization.
- A usecase 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.
- A usecase describes a set of sequences, in which each sequence represents the interaction of the things outside the system (its actors) with the system itself (and its key abstractions).
- A usecase involves the interaction of actors and the system.

An actor represents a coherent set of roles that users of use cases play when interacting with these usecases.
Actors can be human or they can be automated systems

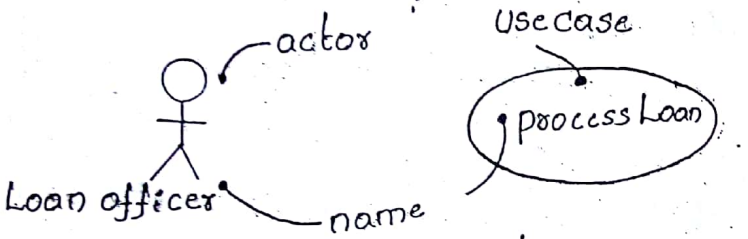


Figure: Actors and use cases

Terms and concepts:

- A usecase 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.

Graphically, a usecase is rendered as an ellipse.

Names:

- Every usecase must have a name that distinguishes it from other usecases.
- A name is a textual string.

That name alone is known as a simple name a path name is the use case named prefixed by the name of the package in which that usecase lives.

- A usecase is typically drawn showing only its name as shown in figure.

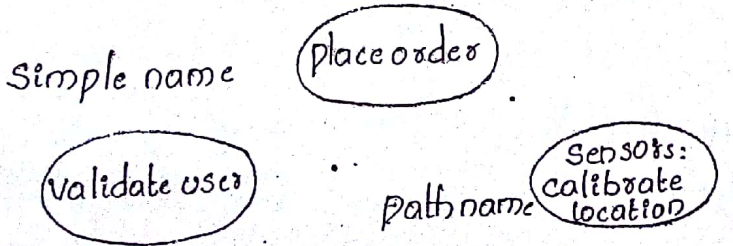


Figure: Simple and path Names

cases and Actors:

An actor represents a coherent set of roles that users of usecases play when interacting with these usecases.

Typically, an actor represents a role that a human, a hardware device, or even another system plays with a system.

- For example, if you work for a bank, you might be a loanofficer.
- If you do your personal banking there, as well, you will also play the role of customer.
- As shown in figure, actors are rendered as stick figures.
- you can define general kinds of actors (such as customer) and specialize them (such as commercialCustomer) using generalization relationships.

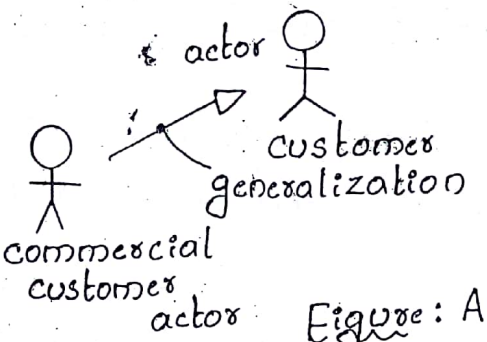


Figure: Actors

- Actors may be connected to usecases only by Association.

2. Use Cases and flow of Events:

- A usecase describes what a system (or a subsystem, class, or interface) does but it does not specify how it does it.
- When you model, it's important that you keep clear the separation of concerns between this outside and inside view.
- you can specify the behavior of a usecase by describing a flow of ever in text clearly enough for an outsider to understand it easily.
- For example, in the context of an ATM system, you might describe t usecase ValidateUser in the following way.

3. Main flow of events: The usecase starts when the system prompts the customer for a PIN number. The customer can now enter a PIN numbe via the keypad. The customer commits the entry by pressing the enter button. The system then checks this PIN number to see if it is valid. I the PIN number is valid, the system acknowledges the entry, thus end the usecase.

4. Exception flow of events: The customer can cancel a transaction at an [illegible] button, thus restarting the usecase. No

changes are made to the customers account.

Exception flow of events: The customer can clear a PIN number before committing it and reenter a new PIN number.

Exception flow of events: If the customer enters a invalid PIN no the usecase restarts. If this happen three times in a row, the system cancels the entire transaction, preventing the customer from interacting with the ATM for 60 seconds.

Usecases and collaborations:

- A usecase captures the intended behavior of the system (or subsystem class, or interface) you are developing, without having to specify how that behavior is implemented.
- As shown in figure, you can explicitly specify the realization of a use case by a collaboration.
- Most of the time, though, a given usecase is realized by exactly one collaboration, so you will not need to model this relationship explicitly

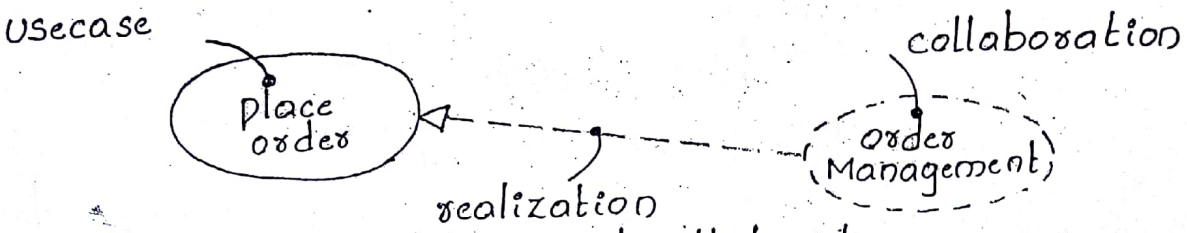


Figure: usecases and collaborations.

Organizing usecases:

- You can organize usecases by grouping them in packages in the same manner in which you can organize classes.
- You can also organize usecases by specifying generalization, include and extend relationships among them.

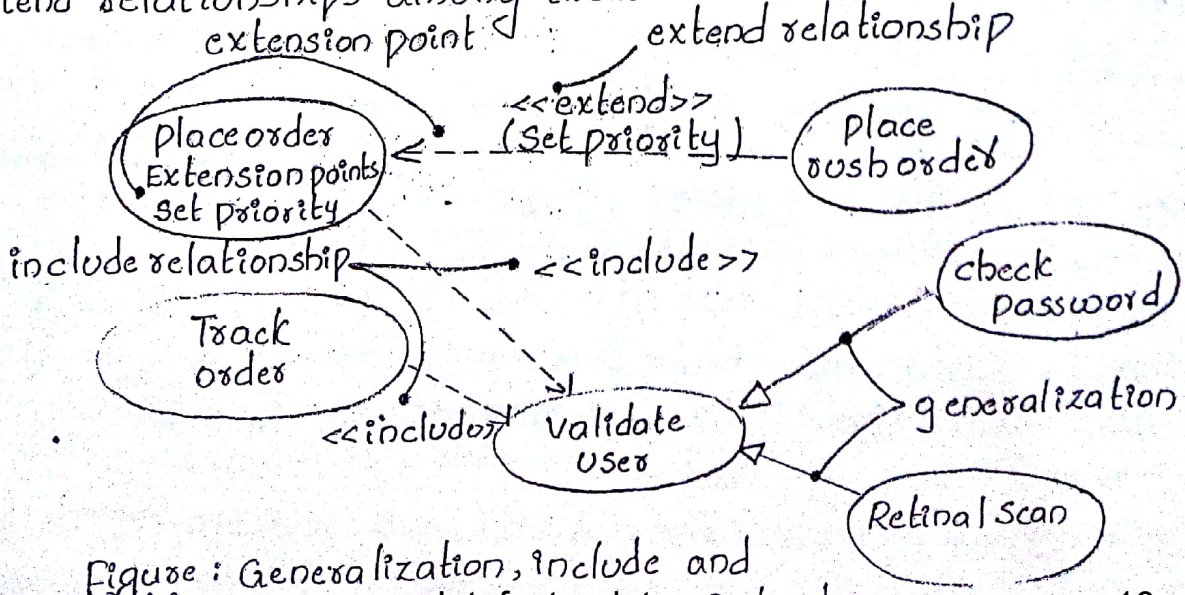


Figure: Generalization, include and extend

flow of events: obtain and verify the order number, include validate user). For each part in the order, query its status, then repo back to the user.

Main flow of events: include (Validate user). collect the user's order items (set priority). submit the order for processing.

Usecase Diagrams:

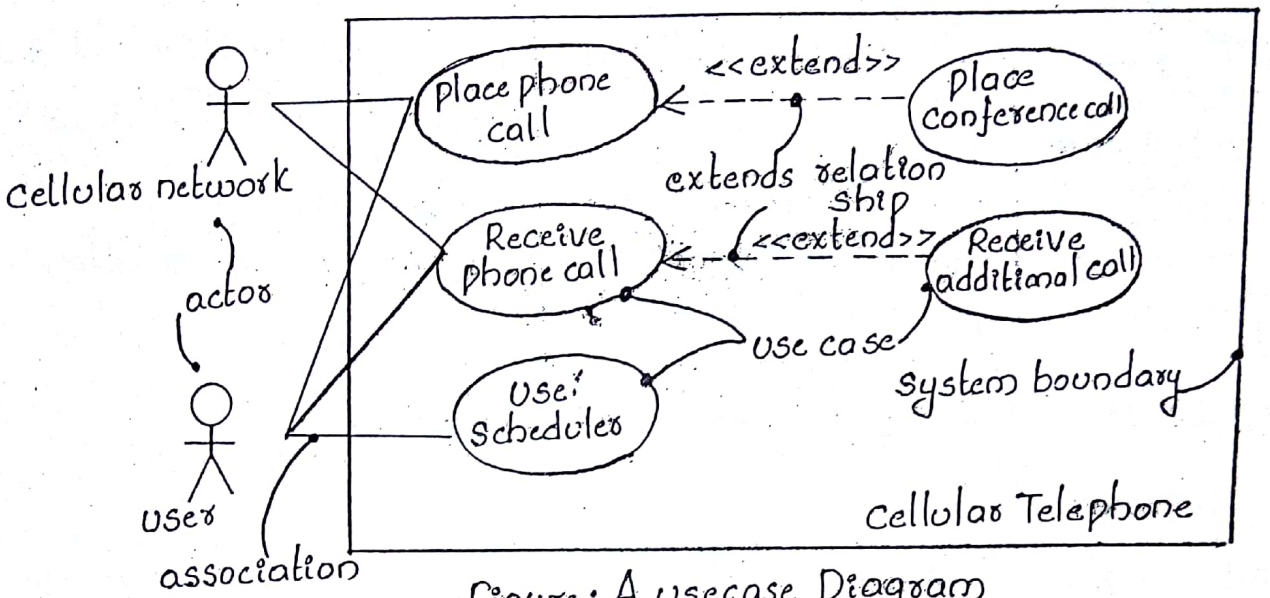


Figure: A usecase Diagram

Common Uses:

→ When you model the static use case view of a system, you will typica apply use case diagrams in one of two ways

1. To model the context of a system
2. To model the requirements of a system.

Common modeling Techniques: To model the context of a system,

→ Identify the actors that surround the system by considering which gro require help from the system to perform their tasks, which groups a needed to execute the systems functions, which groups interact with external hardware or other software systems and which groups f form secondary functions for administration and maintainence.

→ Organize actors that are similar to one another in a generalization/sp ialization hierarchy.

→ Where it aids understandability, provide a stereotype for each such a

→ Populate a usecase diagram with these actors and specify the path communication from each actor to the systems use cases.

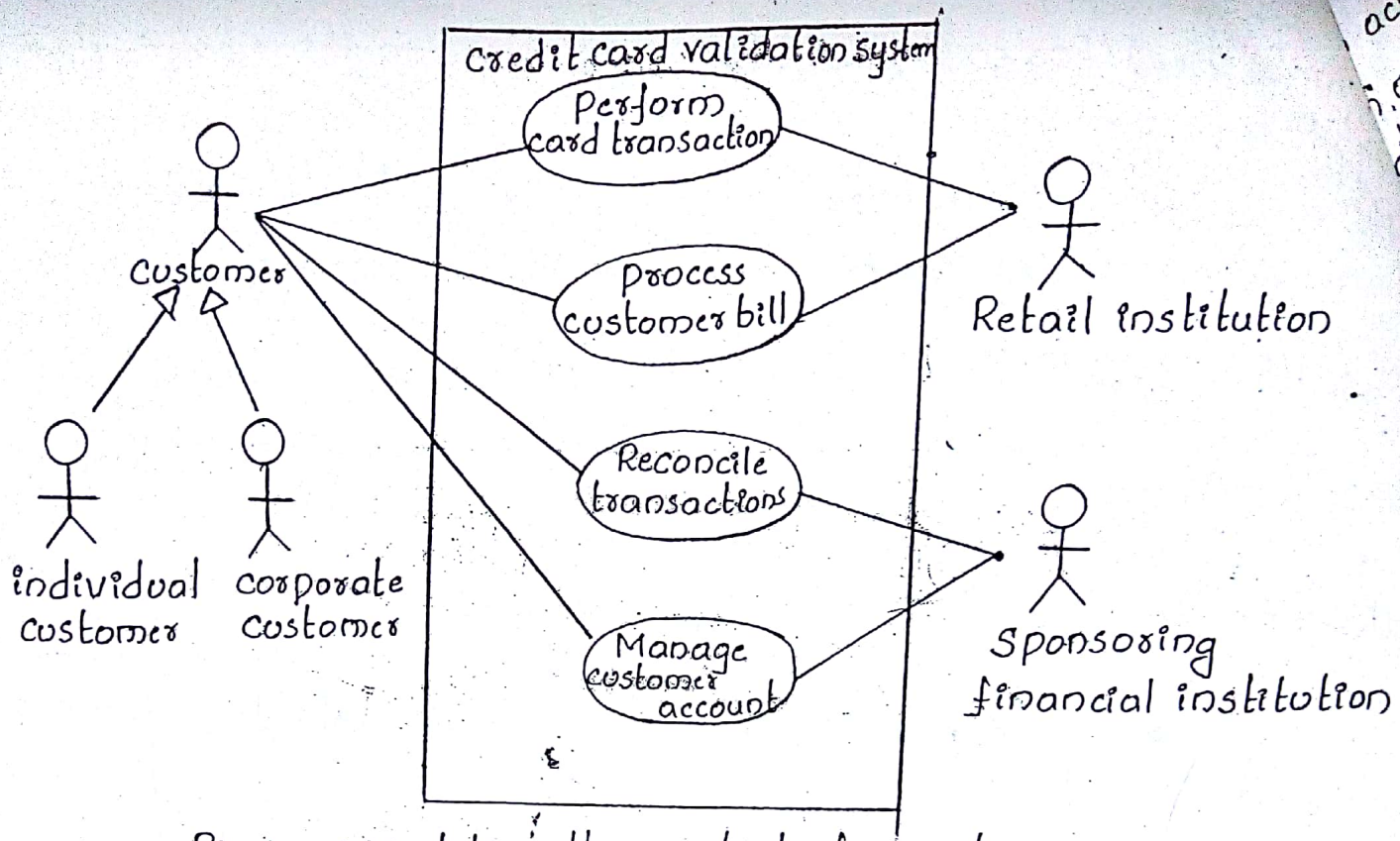


Figure : Modeling the context of a system.

Activity Diagrams :

- An activity diagram is essentially a flow chart, showing flow of control from activity to activity.

You use activity diagrams to model the dynamic aspects of a system. Activity diagrams are not only important for modeling the dynamic aspects of a system, but for constructing executable systems through forward and reverse engineering.

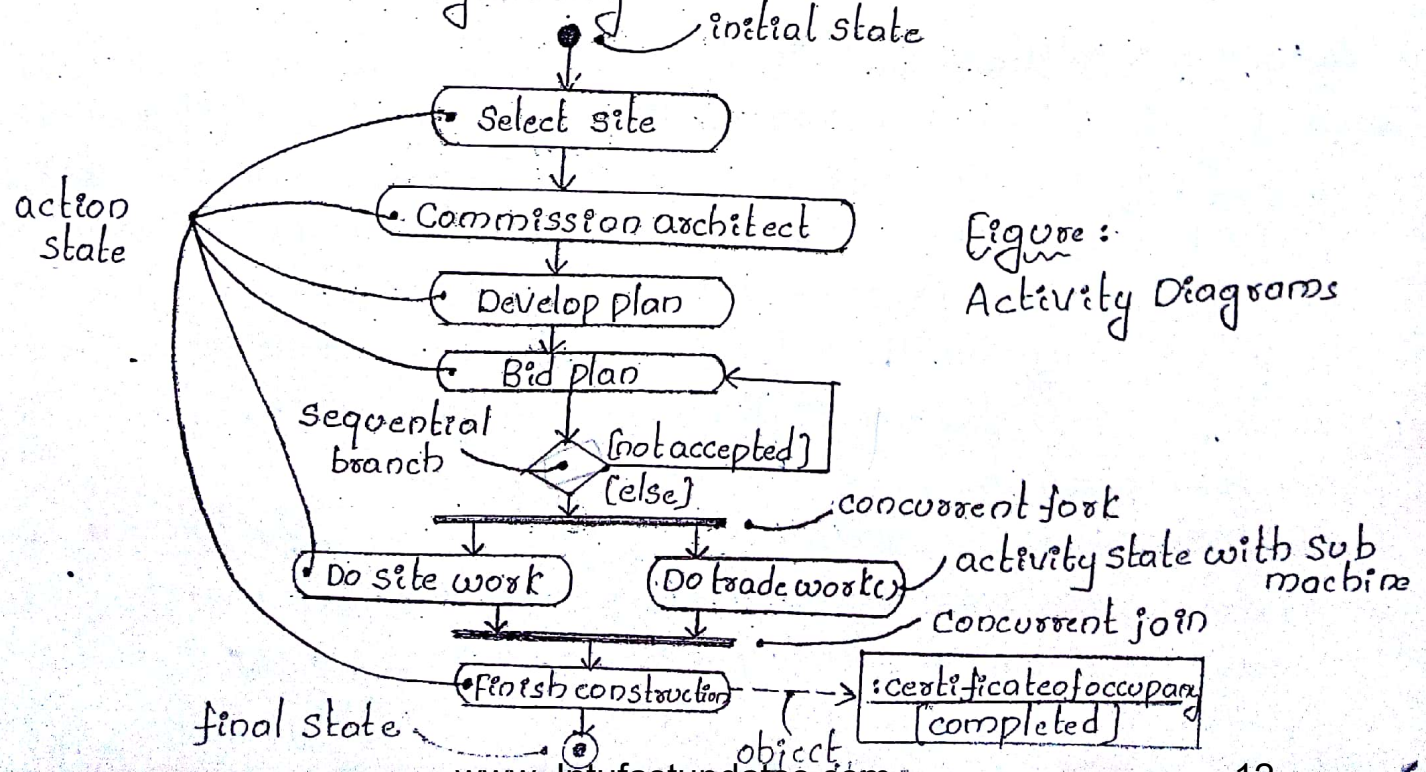


Figure : Activity Diagrams

> activity diagram shows the flow from activity to activity. (7)
n activity is an ongoing nonatomic execution within a state mach
ctivities ultimately result in some action, which is made up of e
cutable atomic computations that result in a change in state of th
system or the return of a value.

- Actions encompass calling another operation, sending a signal, creating or destroying an object, or some pure computation, suc as evaluating an expression.
- Graphically, an activity diagram is a collection of vertices and arcs.
- Activity diagrams commonly contain
  - (*) Activity states and action states
  - (*) Transitions
  - (*) objects

Action states and Activity states:

Action states:
- You might evaluate some expression that sets the value of an attrib or that returns some value.
- Alternately, you might call an operation on an object, send a signa an object, or even create or destroy an object.
- These executable, atomic computations are called action states
- Action states cannot be decomposed.
- Furthermore, action states are atomic, meaning that events may occ but the work of the action state is not interrupted.

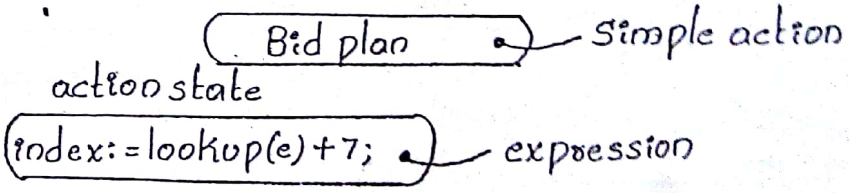
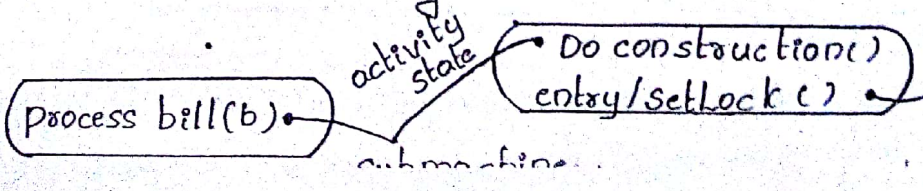


Figure: Action states

Activity states:-
- In contrast, activity states can be further decomposed, their activ being represented by other activity diagrams.
- An action state is an activity state that cannot be further decomp



Transitions:

When the action or activity of a state completes, flow of control passes immediately to the next action or activity state.

You specify this flow by using transitions to show the path from one action or activity state to the next action or activity state.

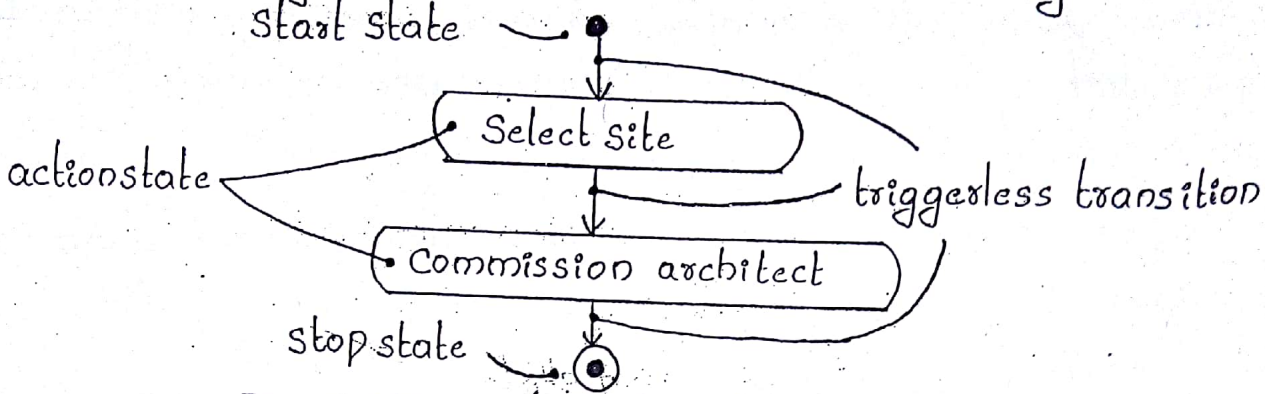


Figure : Triggerless Transitions.

Branching:

As in a flowchart, you can include a branch, which specifies alternate paths taken based on some Boolean expression.

As shown in figure, you represent a branch as a diomand.

A branch may have one incoming transition and two or more outgoing ones.

As a convenience, you can use the keyword else to mark one outgoing transition.

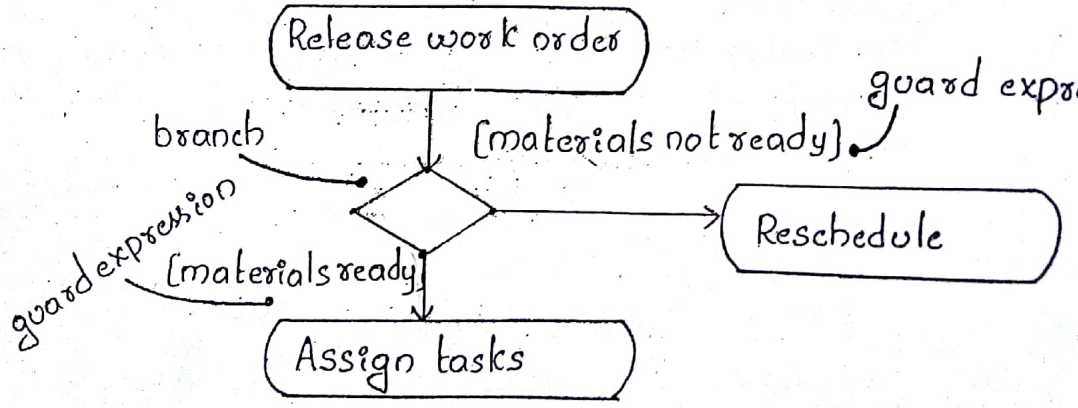


Figure : Branching

Forking and Joining:

Simple and branching sequential transitions are the most common path you will find in activity diagrams.

However – especially when you are modeling work flows of business processes – you might encounter flows that are concurrent.

, the UML, you use a synchronization bar to specify the forking and joining of these parallel flows of control.

A synchronization bar is rendered as a thick horizontal or vertical line.

A join may have two or more incoming transitions and one outgoing transition.

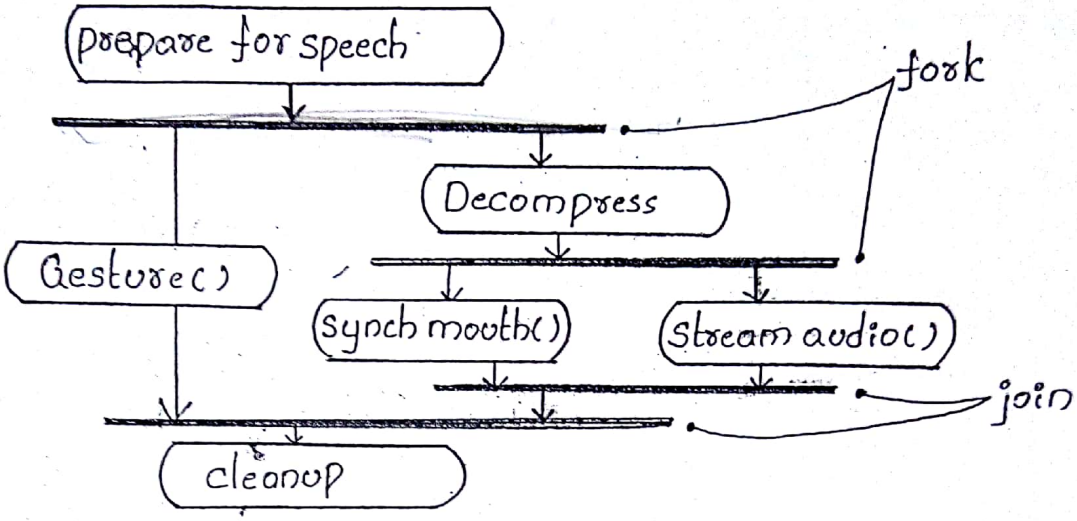


Figure: Forking and joining

Swimlanes:

- To partition the activity states on an activity diagram into groups, ea group representing the business organization responsible for those acti
- In the UML, each group is called a swimlane.
- Because, Visually each ground is divided from its neighbour by a Vertica solid lane.
- A swimlane specifies a locus of activities.
- Each swimlane has a name unique within its diagram.
- In an activity diagram partitioned into swimlanes, every activity belongs to exactly one swimlane, but transitions may cross lanes.

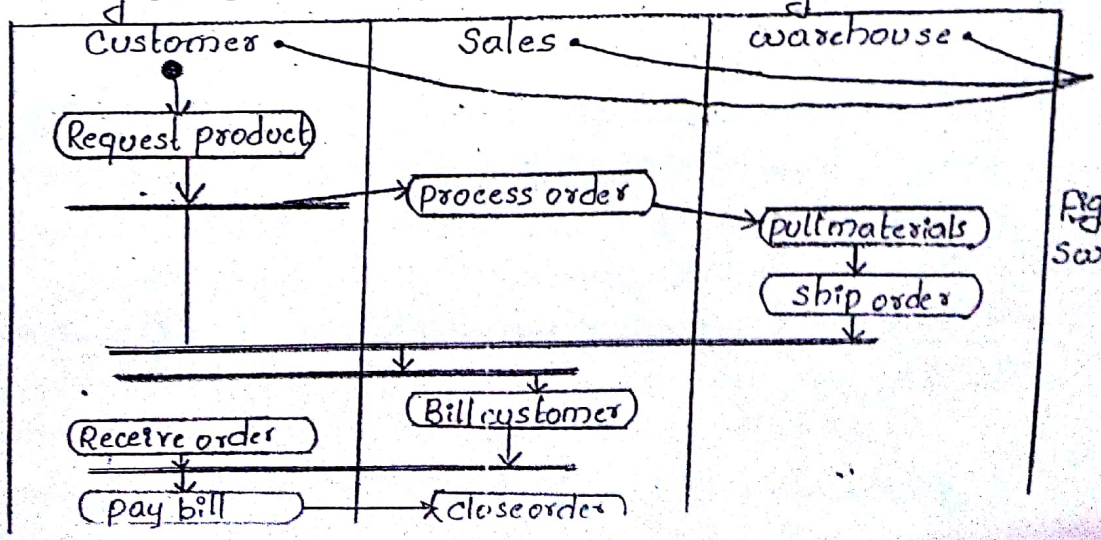


Figure
Swim

Object flow:

- objects may be involved in the flow of control associated with a diagram.
- Instances of these two classes will be produced by certain activities cess order will create an order object, for example) other activities modify these objects (for example, ship order will change the state the order object to filled).

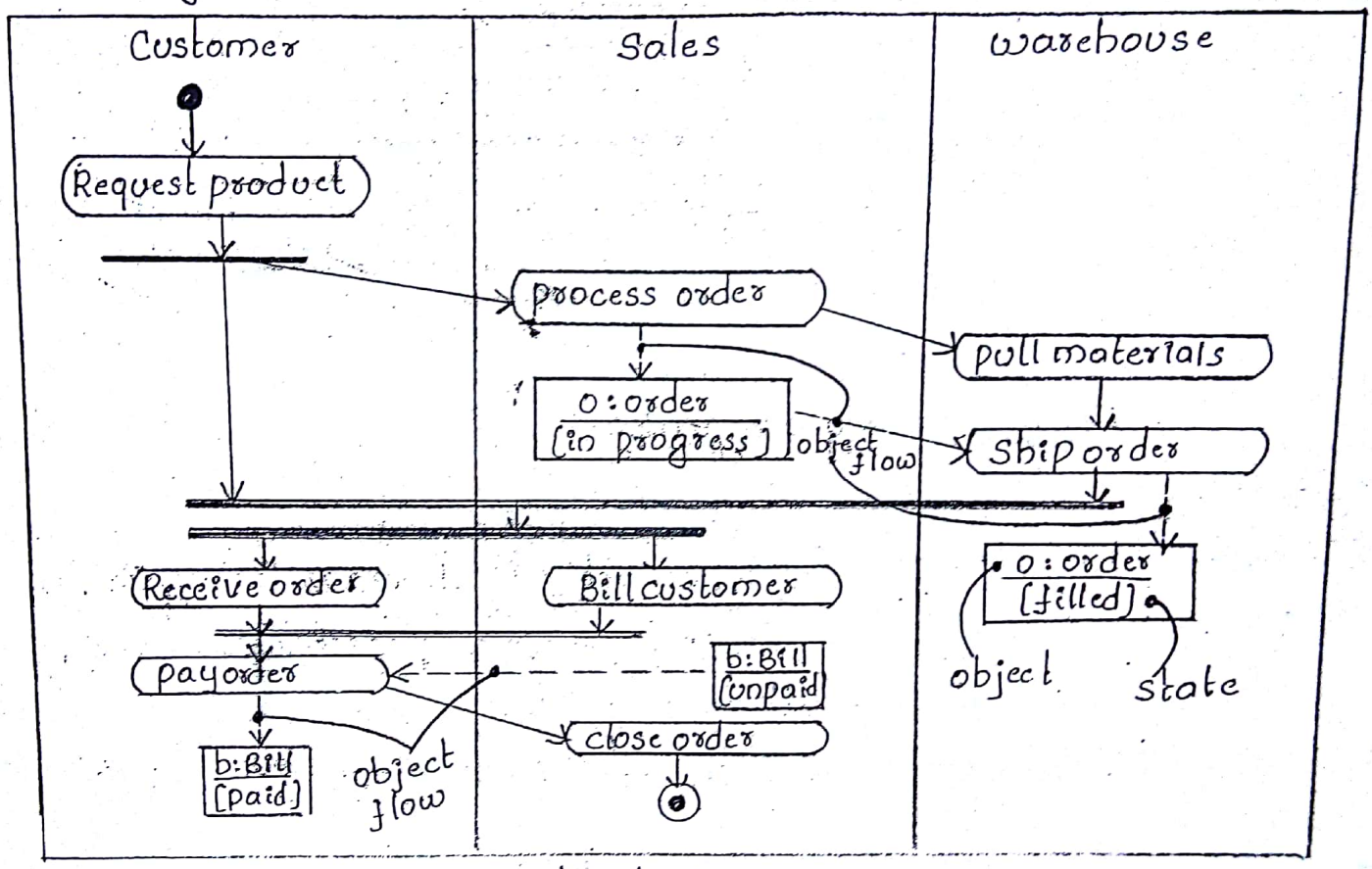


Figure: object flow.

Common uses: (1) To model a workflow (2) To model an operation CMT in notes

State Machines:

- Using an interaction, you can model the behavior of a society of objects that work together.
- Using a state machine, you can model the behavior of an individual objec

A state machine is a behaviour that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.

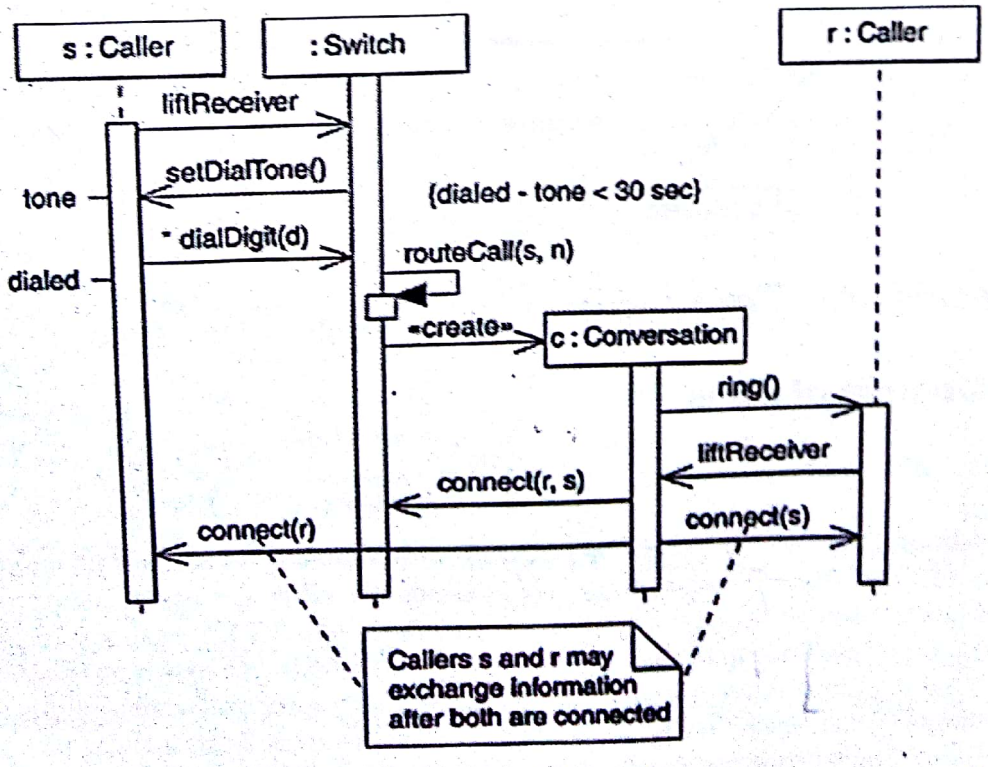
- The UML provides a graphical representation of states, transitions, event and actions.
- This notation permits you to visualize the behavior of an object in a way that lets you emphasize the important elements in the life of that object.

COMMON MODELING TECHNIQUES

Common Modeling Techniques for Interaction Diagrams

Modeling Flows of Control by Time Ordering (Sequence Diagram):

- 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.
- Lay them out on the sequence diagram from left to right, placing the more important objects to the left and their neighboring objects to the right.
- Set the lifeline for each object. In most cases, 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. S
- 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 (such as its parameters), as necessary to explain the semantics of the interaction.
- 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- and post-conditions to each message.

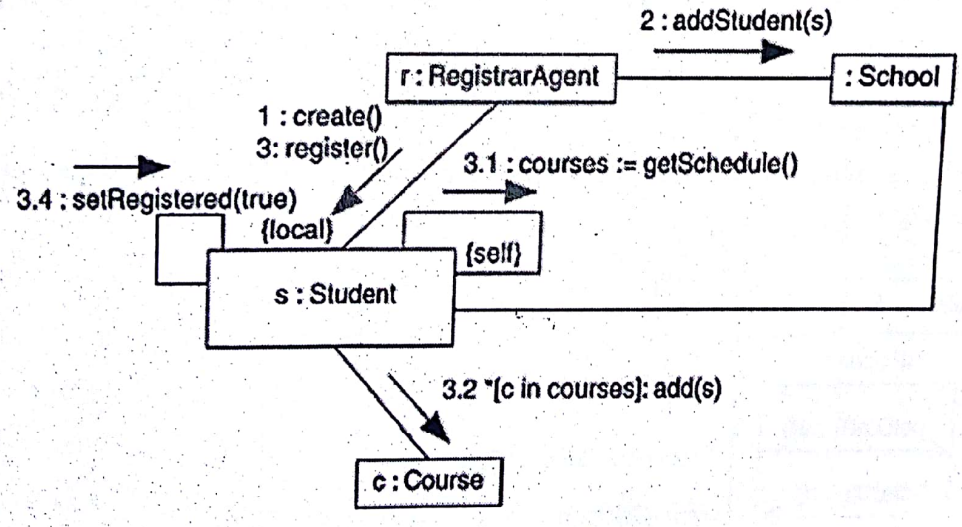


Seq – Flow of control by Time ordering

forward & Reverse engineering

Modeling Flows of Control by Organization (Collaboration Diagram/Communication Diagram):

- 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.
- Lay them out on the communication diagram as vertices in a graph, placing the more important objects in the center of the diagram and their neighboring objects to the outside.
- 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 annotations (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.

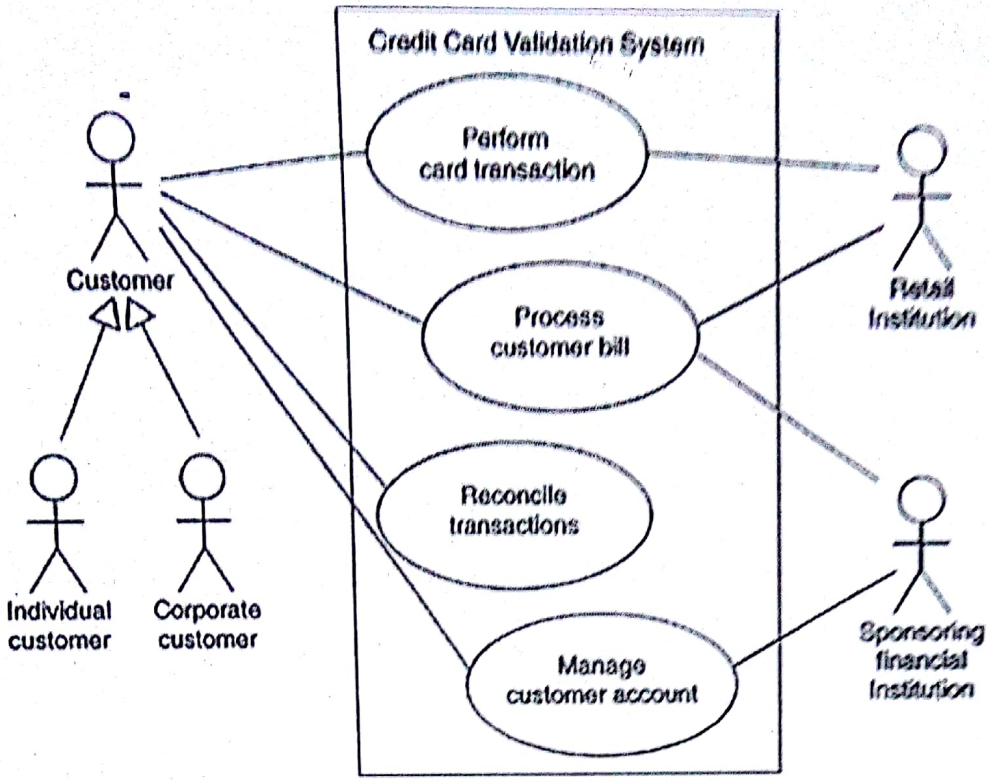


Common Modeling Techniques for Use Case Diagram

Modeling the Context of a System:

To model the context of a system,

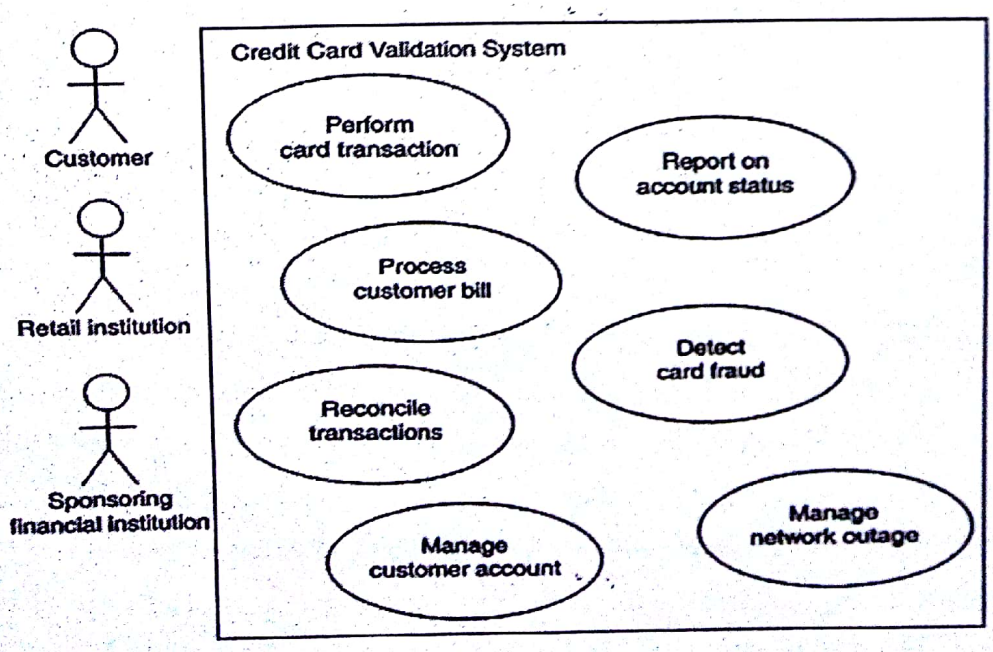
- Identify the boundaries of the system by deciding which behaviors are part of it and which are performed by external entities. This defines the subject.
- Identify the actors that surround the system by considering which groups require help from the system to perform their tasks, which groups are needed 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.
- Where it aids understandability, provide a stereotype for each such actor.



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 or constraints that assert nonfunctional requirements; you may have to attach some of these to the whole system.

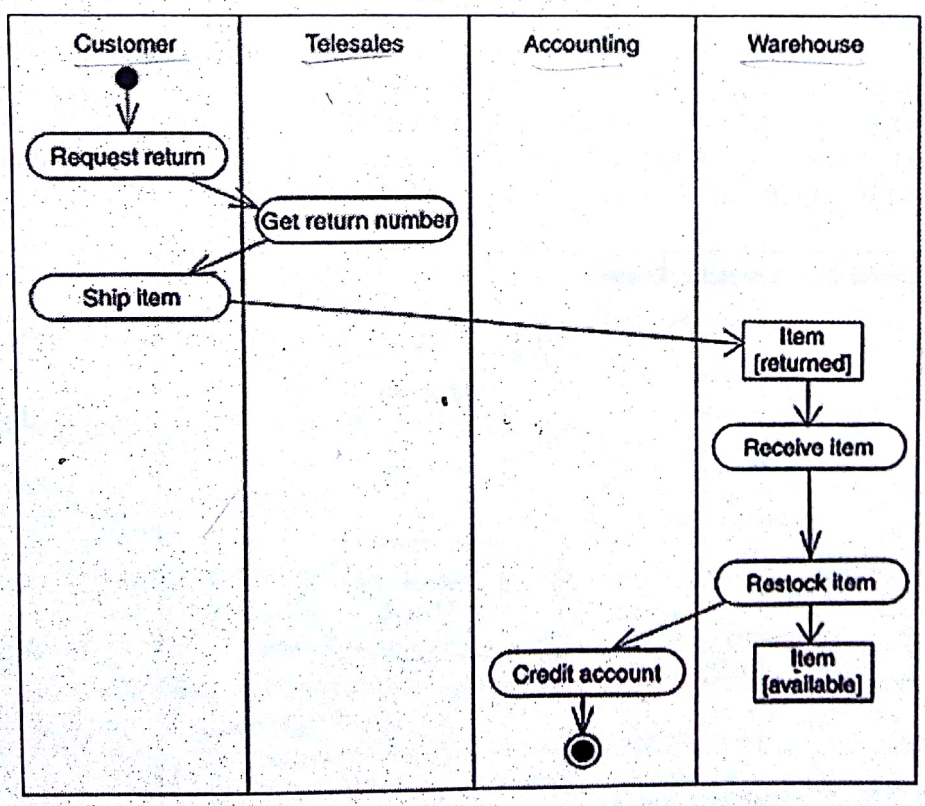


Common Modeling Techniques for Activity Diagram

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 swim lane for each important business object or organization.
- Identify the preconditions of the workflow's initial state and the post conditions 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 actions that take place over time and render them in the activity diagram.
- For complicated actions or for sets of actions that appear multiple times, collapse these into calls to a separate activity diagram.
- Render the flows that connect these actions and activity nodes. Start with the sequential flows in the workflow first, next consider branching, and only then consider forking and joining.
- If there are important object values 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.



Modeling an Operation:

To model an operation,