ROHINI COLLEGE OF ENGINEERING & TECHNOLOGY
Near Anjugramam Junction, Kanyakumari Main Road, Palkulam, Variyoor P.O - 629401
Kanyakumari Dist, Tamilnadu., E-mail : admin@rcet.org.in, Website : www.rcet.org.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject Code : **CS8592**

Subject Name : **Object Oriented Analysis and Design**

Academic Year : **2020-2021**

Year/Semester : **III/V**

Regulation : **2017**

UNIT III - DYNAMIC AND IMPLEMENTATION UML DIAGRAMS

STATIC UML DIAGRAMS

UNIT - III

DYNAMIC AND IMPLEMENTATION UML DIAGRAMS

3.1 DYNAMIC DIAGRAMS

3.1.1 Interaction Diagrams

The main idea in developing interaction diagram is simplicity.

But interaction diagrams lose clarity with conditional behavior.

To capture complex behavior in a single diagram, we can go for activity diagrams.

Interaction diagrams are models that describe how groups of objects collaborate in some behavior. There are two kinds of interaction diagrams

- Sequence diagrams
- Collaboration diagrams

3.1.2 Sequence Diagram

An object is shown as a box at the top of a dashed vertical line.

This vertical line is called objects life line. The life line represents objects life during interaction.

This was given by Jacobson

Each message is represented by an arrow between the life lines of two objects.

The order of messages occur from top to bottom of the page.

Message contains message name, arguments and some control information.

Self call – is an message that an object sends to itself by sending message arrow back to the same life line.

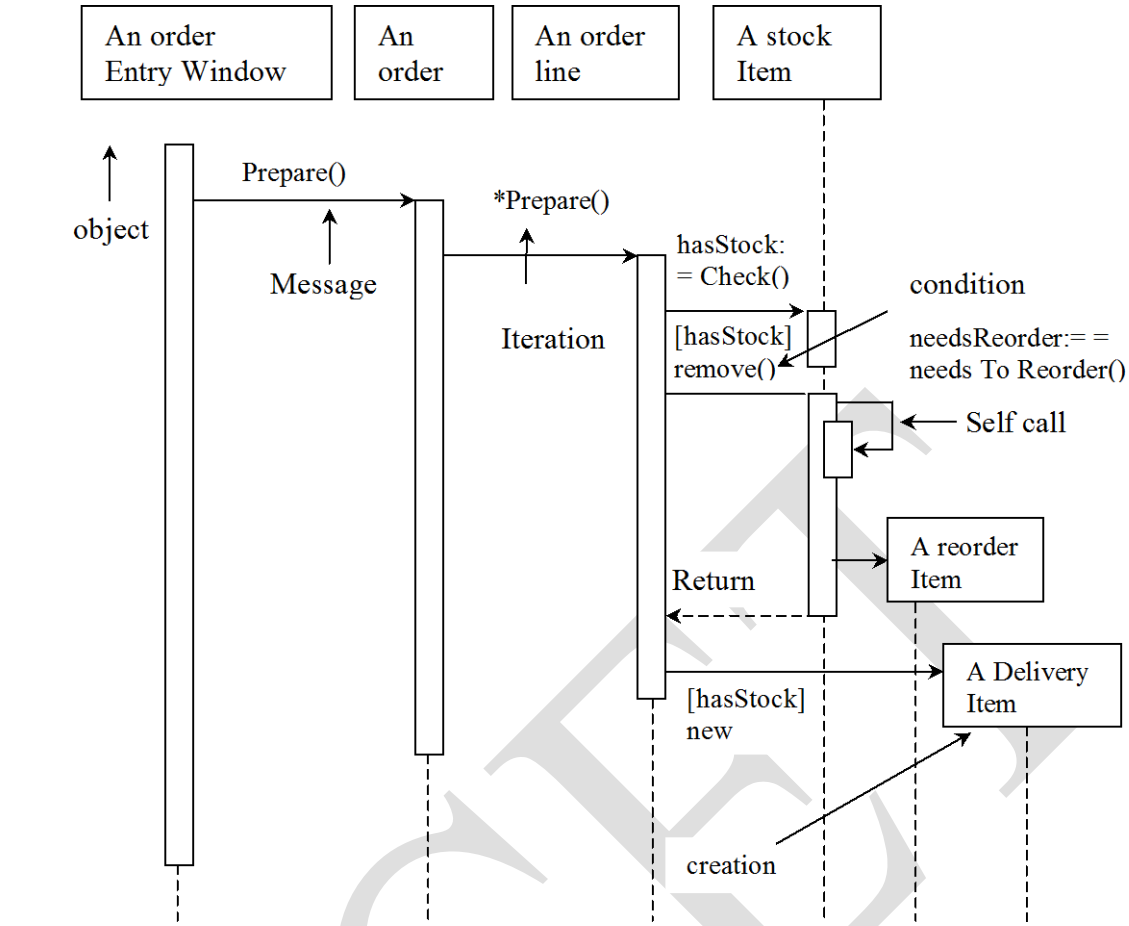


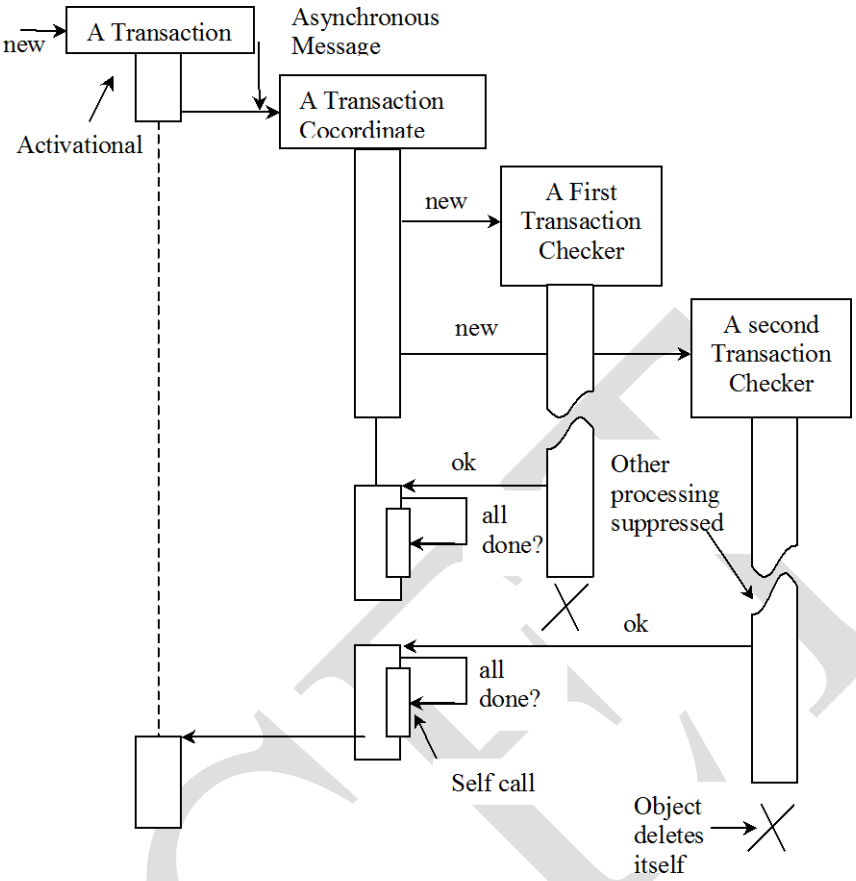
Figure 3.1 Sequence Diagram

Two control information are valuable.

1) There is a condition which indicates when a message is sent only if condition is true.
2) Next is the iteration maker, which shows that a message is sent many times to multiple receiver objects Basics of the iteration are shown within brackets *[for all order lines]
   - A return can be included to indicate the return from a message, not a new message.
   - Return is indicated by dashed line.

Sequence diagrams are also valuable for concurrent processes.

Concurrent Processes And Activations



At the creation of transaction, a Transaction coordinator is created. This coordinator creates transaction checker objects which is responsible for a particular check.

Different checking processes are added. Each checker is called asynchronously and proceeds in parallel.

When transaction checker completes, it informs transaction coordinator. After all checkers are completed the coordinator notifies the transaction that all is well.

The half arrow heads indicate asynchronous message. It does not block the caller but carries its own processing. It does one of the following things.

1) Create a new thread, in which case it links to the top of an activation.
2) Create a new objects
3) Communicate with a thread that is already running.

Object deletion is shown with a large X.

Objects can self destruct or they can be destroyed by another message.

A sequence diagram shown an interaction arranged in a time sequence.

It shows the objects participating in interaction by their life lines and the messages they exchange, arranged in a time sequence.

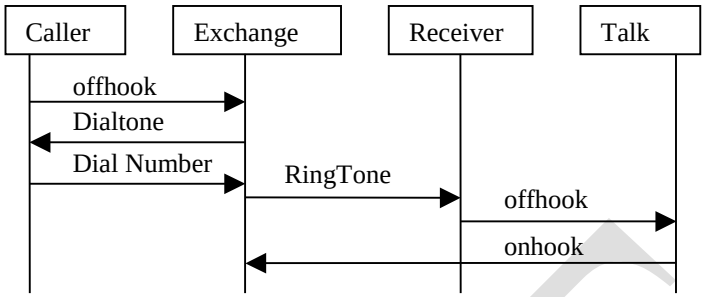


Figure 3.2 Sequence Diagram

A sequence diagram has 2 dimensions

1) Vertical dimension represents time [Lifetime]
2) Horizontal dimension represents different objects

We can look at the sequence diagram and understand the overall control flow of a program.

3.1.3 Collaboration Diagram

A collaboration diagram represents a collaboration which is a set of objects related to achieve a desired outcome.

In collaboration, the sequence is indicated by numbering the messages several numbering schemes are available.

With Simple Numbering

COLLABORATION DIAGRAM

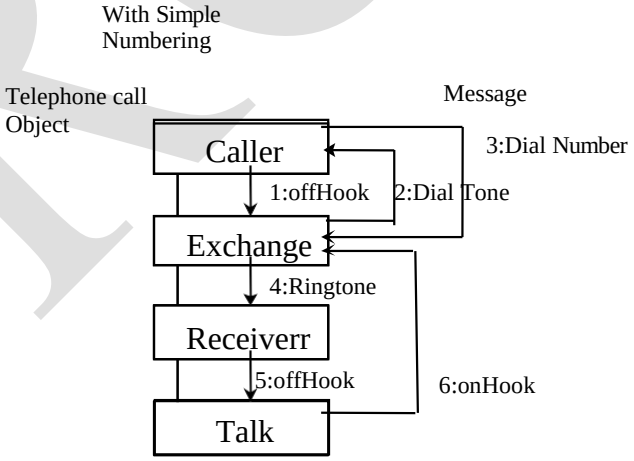
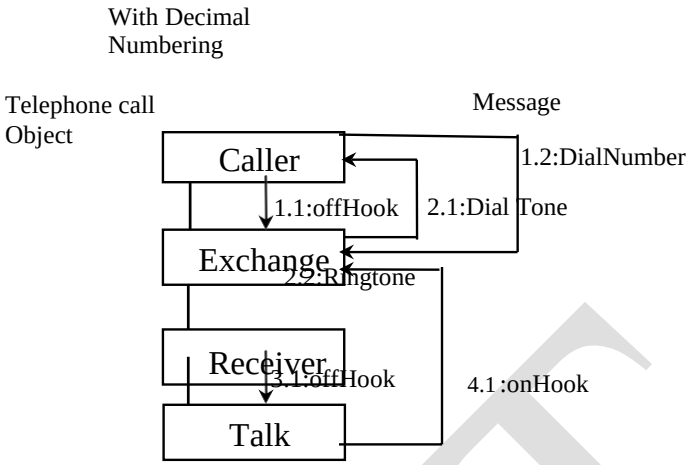


Figure 3.3

With Decimal Numbering



It is the second form of interaction diagrams.

Here in collaboration diagrams, we number the messages and indicates their sequences.

Several numbering schemes are used for collaboration diagrams.

1) Simple Numbering scheme [used in past]
2) Decimal Numbering scheme [used in UML]

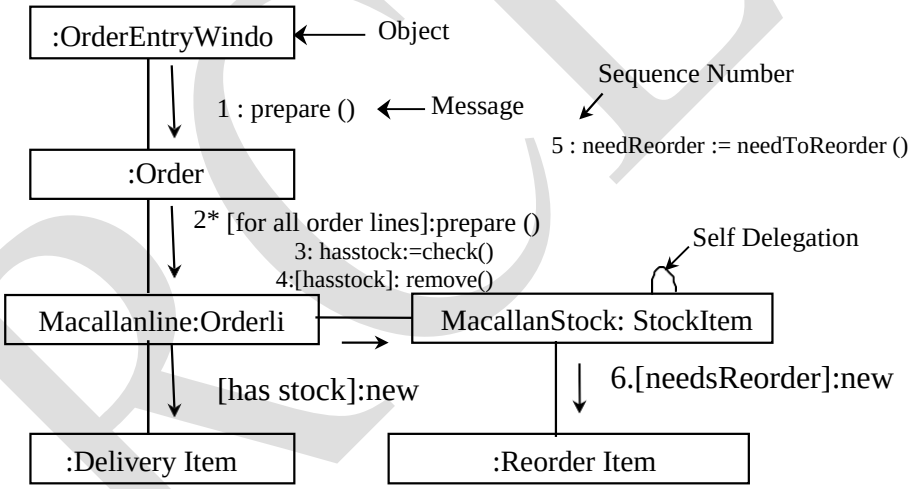


Figure 3.4

Collaboration Diagram With Decimal Numbering

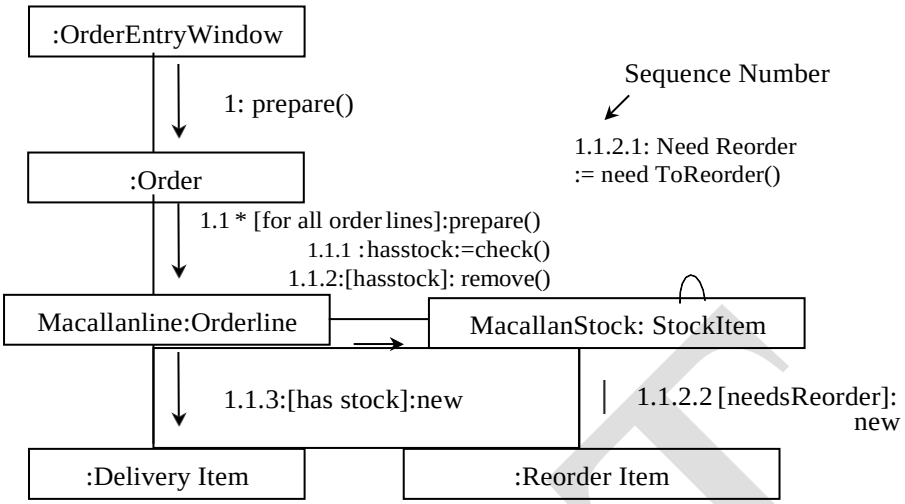


Figure 3.5

The same kind of information are added regardless of what numbering systems used.

3.1.3.1 Comparing Sequence and Collaboration Diagrams

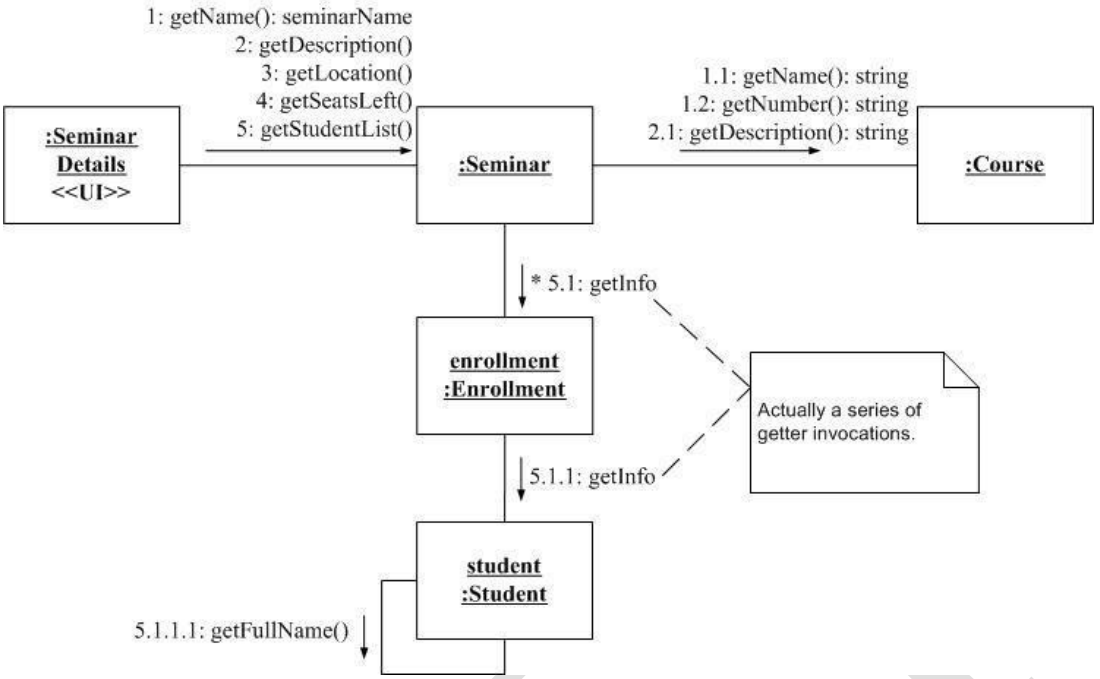
The principal feature of the interaction diagram is their simplicity.

Sequence diagram puts emphasis on sequence. It is easy to see the order in which the things occur.

Collaboration diagram uses the layout to indicate how objects are statically connected.

3.1.4 When to use Communication Diagram

communication diagrams show the message flow between objects in an OO application and also imply the basic associations (relationships) between classes. Figure 1 presents a simplified collaboration diagram for displaying a seminar details screen or page. The rectangles represent the various objects involved that make up the application. The lines between the classes represent the relationships (associations, composition, dependencies, or inheritance) between them. The same notation for classes and objects used on UML sequence diagrams are used on UML communication diagrams, another example of the consistency of the UML. The details of your associations, such as their multiplicities, are not modeled because this information is contained on your UML class diagrams: remember, each UML diagram has its own specific purpose and no single diagram is sufficient on its own. Messages are depicted as a labeled arrow that indicates the direction of the message, using a notation similar to that used on sequence diagrams



3.1.5 State Machine Diagrams and Modelling

A state machine diagram models the behaviour of a single object, specifying the sequence of events that an object goes through during its lifetime in response to events. As an example, the following state machine diagram shows the states that a door goes through during its lifetime.

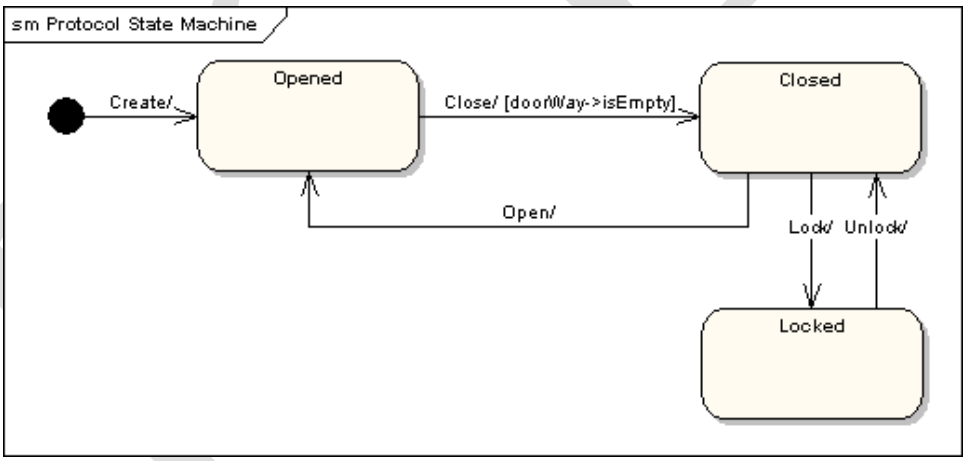
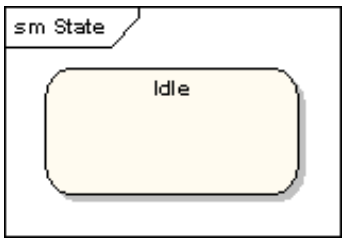


Figure 3.7

The door can be in one of three states: "Opened", "Closed" or "Locked". It can respond to the events Open, Close, Lock and Unlock. Notice that not all events are valid in all states; for example, if a door is opened, you cannot lock it until you close it. Also notice that a state transition can have a guard condition attached: if the door is Opened, it can only respond to the Close event if the condition doorWay->isEmpty is fulfilled. The syntax and conventions used in state machine diagrams will be discussed in full in the following sections.

States

A state is denoted by a round-cornered rectangle with the name of the state written inside it.



Initial and Final States

The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.

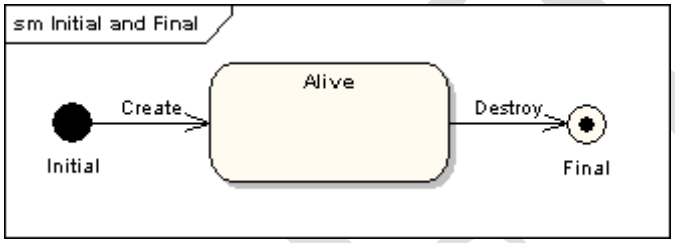


Figure 3.8

Transitions

Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.

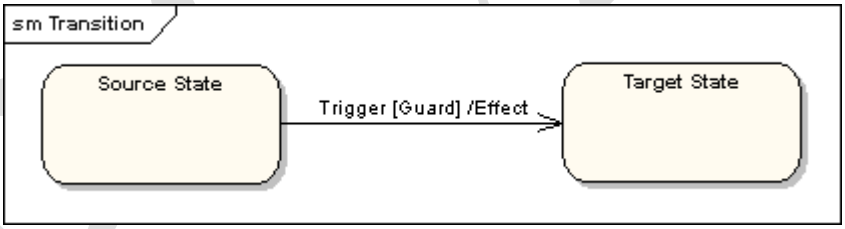


Figure 3.9

3.1.6 When to Use State Diagrams

3.1.6.1 Statechart Diagram

A statechart diagram (sate diagram) shows the sequence of states that an object goes through during its life in response to outside stimuli and messages.

A state is represented as a rounded box which contains one or more compartments.

The name compartment holds the optional name of the state. States without names are anonymous.

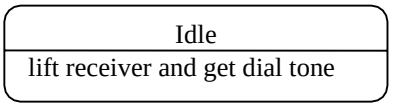
The internal transition compartment holds a list of internal actions or activities performed in response while the object is in the state, without changing states.

Syntax: *Event.name Argument List/Action. Expression*

Example: *help/display help*

3.1.6.2 State Diagram

A simple idle state



A nested dialing state

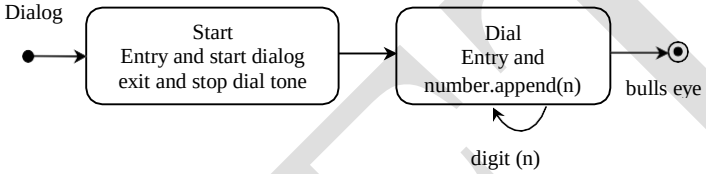


Figure 3.10

A complex transition

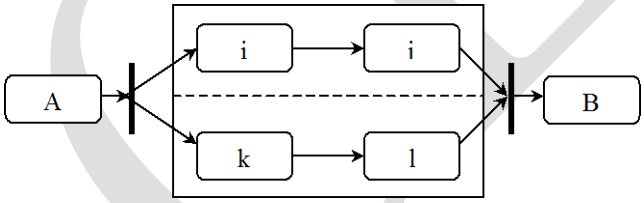


Figure 3.11

State diagrams describe all the possible states that a particular object can get into and how the objects state changes as a result of events that reach the object. They describe the behavior of a system.

We begin at start point.

An initial transition called as "/getfirstitem". Leads to the checking state.

Syntax for transition label

Event[Guard]/Action

All are optional. In this case we have the action "get first item"

Checking state has an activity associated with it do/checkitem.

Syntax: *do/activity*

Action for transaction and Activity for the state.

UML state diagram for an order in the order processing system.

Actions

Actions are associated with transitions. They are the processes that occur quickly and are not interruptible.

For a real time system "quickly" may be within s few machine instruction.

For regular information systems, "quickly" means less than few seconds.

Activity

Activities are associated with states and can take longer. An activity may be interrupted by some event.

When no events are there in the transaction label, transition occurs as soon as any activity associated with the given state is completed.

Here after completing checking state, there transitions come out. All the three have only guards on their label.

Guard

A guard is a logical condition that will return only "true" or "false". A guarded transition only if the guard resolves to "true".

One transition is taken out of given state, so the guards are mutually exclusive for any event. We got 3 conditions.

1) If we have not checked all items. We get the next item and return to checking state.
2) If we have checked all items and were all in stock, we transition to dispatching state.
3) If we have checked all items but not all of them in the stock, we transition to the waiting state.

Waiting State

Then we have waiting state. There are no activities for this state.

Two transactions are out of waiting state labeled ItemReceived event.

From the waiting state, guards on the transitions are evaluated and appropriate transitions are made either to Dispatching or back to waiting.

Dispatching State

From dispatching state, an activity initiates delivery. There is alos a single unguarded transition triggered by delivered event.

Once the "initiate delivery" activity is finished, given order remains in the Dispatching state until delivered event occurs.

The final thing is the transition called "cancelled" we ant to cancel at any point. We cab draw separate transition from checking, waiting and dispatching. Alternative is to create a superstate of all 3 states and draw single transition from that.

Substates inherit any transition on superstate.

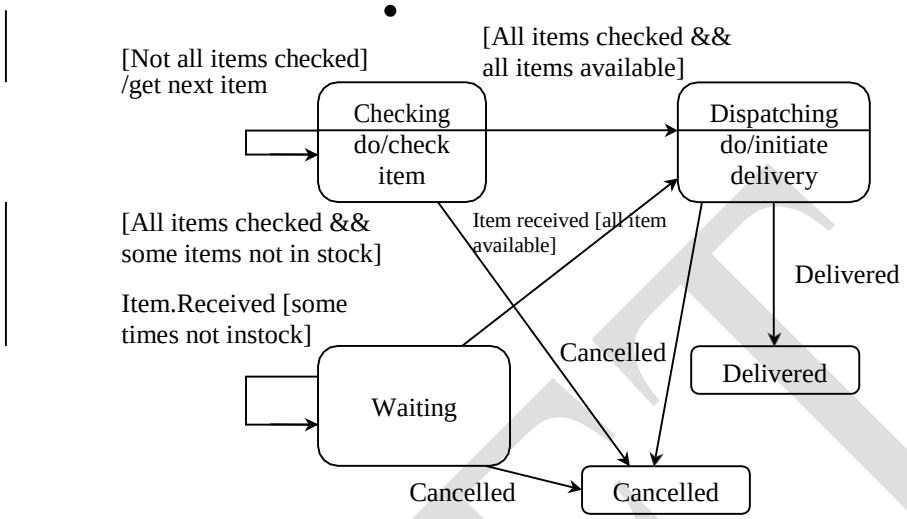


Figure 3.12

State Diagram With Superstates

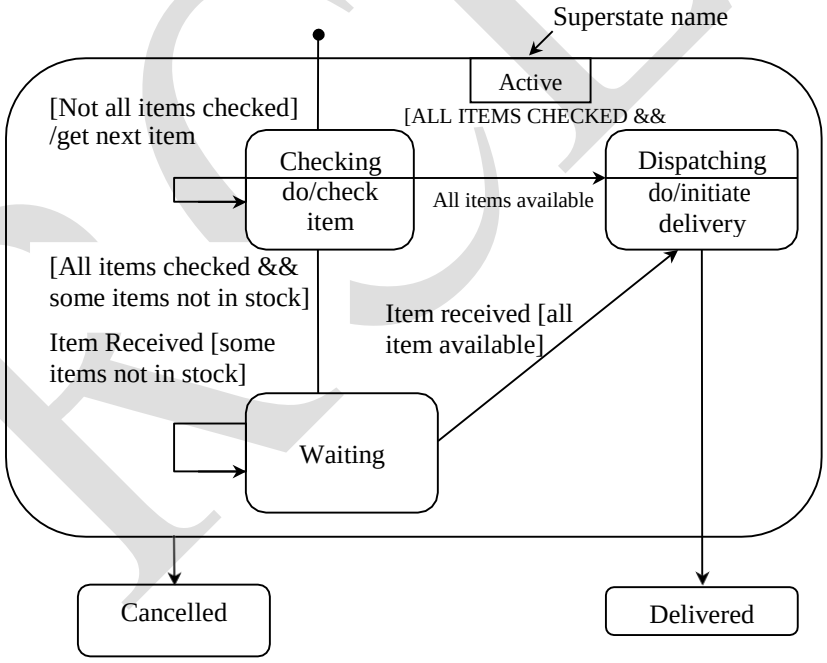


Figure 3.13

There are a couple of other types of events in addition to named events from outside.

An event generated after a period of time. This is indicated with the keyword "after".

Example: We say after (20 minutes)

An event can be generated when condition becomes true. Keyword is "when"

Example: when (temperature > 120 degrees)

Special Events

The special events are entry and exit

An entry event is executed whenever the given state is entered fro transaction.

An exit state is executed whenever the state is lift via a transition.

If the transition goes back to the same state with an action, exit is executed first, then transition action and finally entry action. Such transition is called self transition.

Concurrent State Diagrams

Concurrent state diagrams are useful when an object has sets of independent behaviors.

If several complicated concurrent state diagrams are there for an objects, then split the objects into separate objects.

Payment Authorization

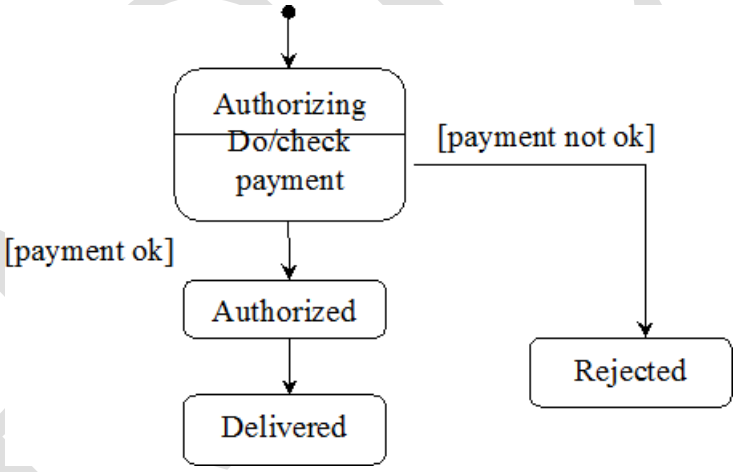


Figure 3.14

Here we begin by doing authorization. Check payment "activity finishes and approves payment.

If payment is "OK", order waits in Authorized state until "deliver" event occurs. Otherwise order enters Rejected state.

Concurrent State Diagram

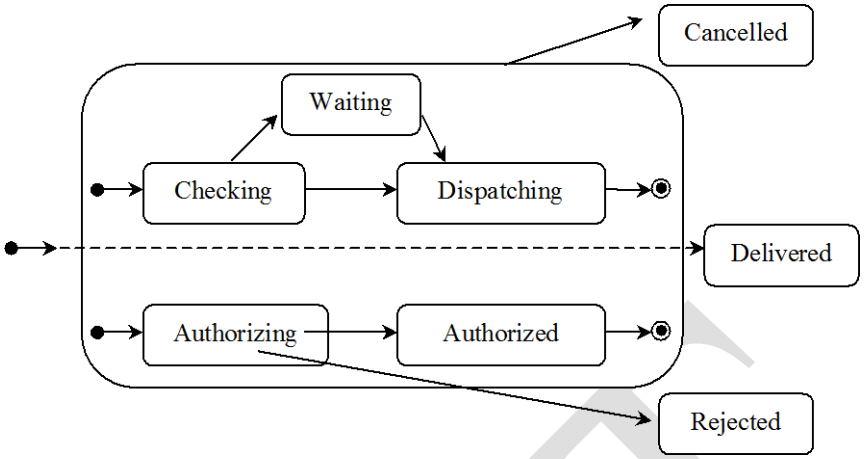


Figure 3.15

The associated states and the cancelled states are combined on a concurrent state diagram.

In concurrent sections of state diagrams, the given order is in 2 different states.

If "check payment" activity of the Authorizing state completes successfully, the order will be in checking and Authorized states.

If "cancel" event occurs, the order will be in cancelled state.

When to Use state diagrams

State diagrams are good at describing the behavior of an object across several use cases.

State diagrams can be used for classes that exhibit interesting behavior.

Building state diagrams helps to understand what is going on.

3.1.6.3 UML Activity Diagrams

An activity diagram is a variation or special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of operations.

Activity diagram is similar to state chart diagram where a token represented an operation. An activity is shown as a round box, containing name of operation.

Concurrent control is indicated by multiple arrows leaving a synchronization bar, represented by short thick bar with incoming and outcoming arrows.

Activity diagram shows internal state of object but external events ay also appear. The two possible states in activity diagram are wait state and activity state.

An Activity Diagram for Processing Mortgage Requests

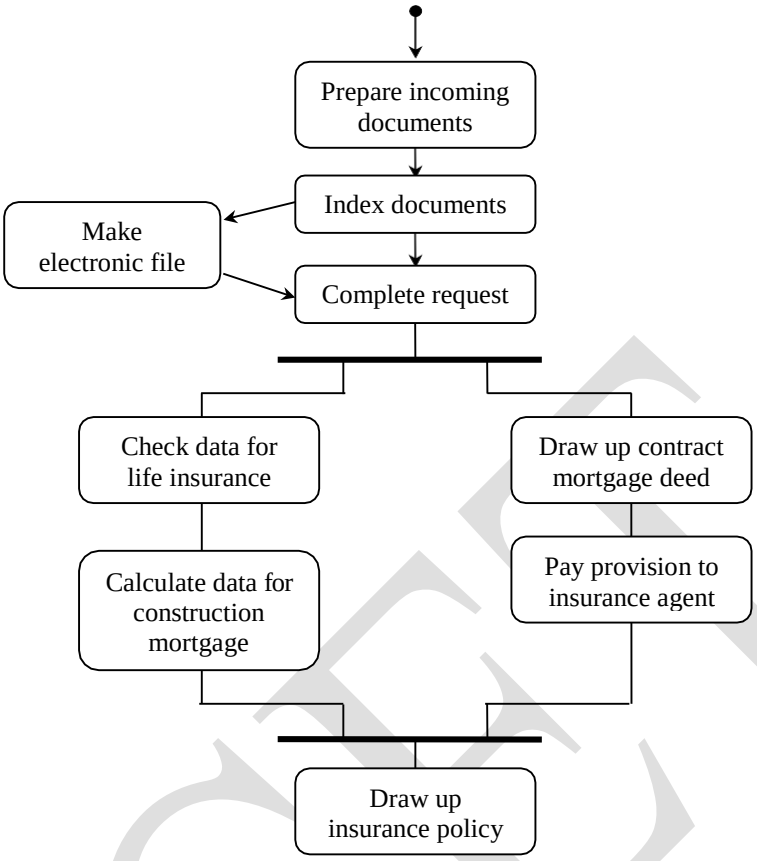


Figure 3.16

3.1.7 Activity Diagrams

Activity diagram represents the performance of operations and transitions are triggered by completion of operations.

Activity diagram shows control flow and data flow.

The business processes are very complex and hence it is better to represent them in pictures rather than text.

Activity diagrams are highly are highly useful for these purpose.

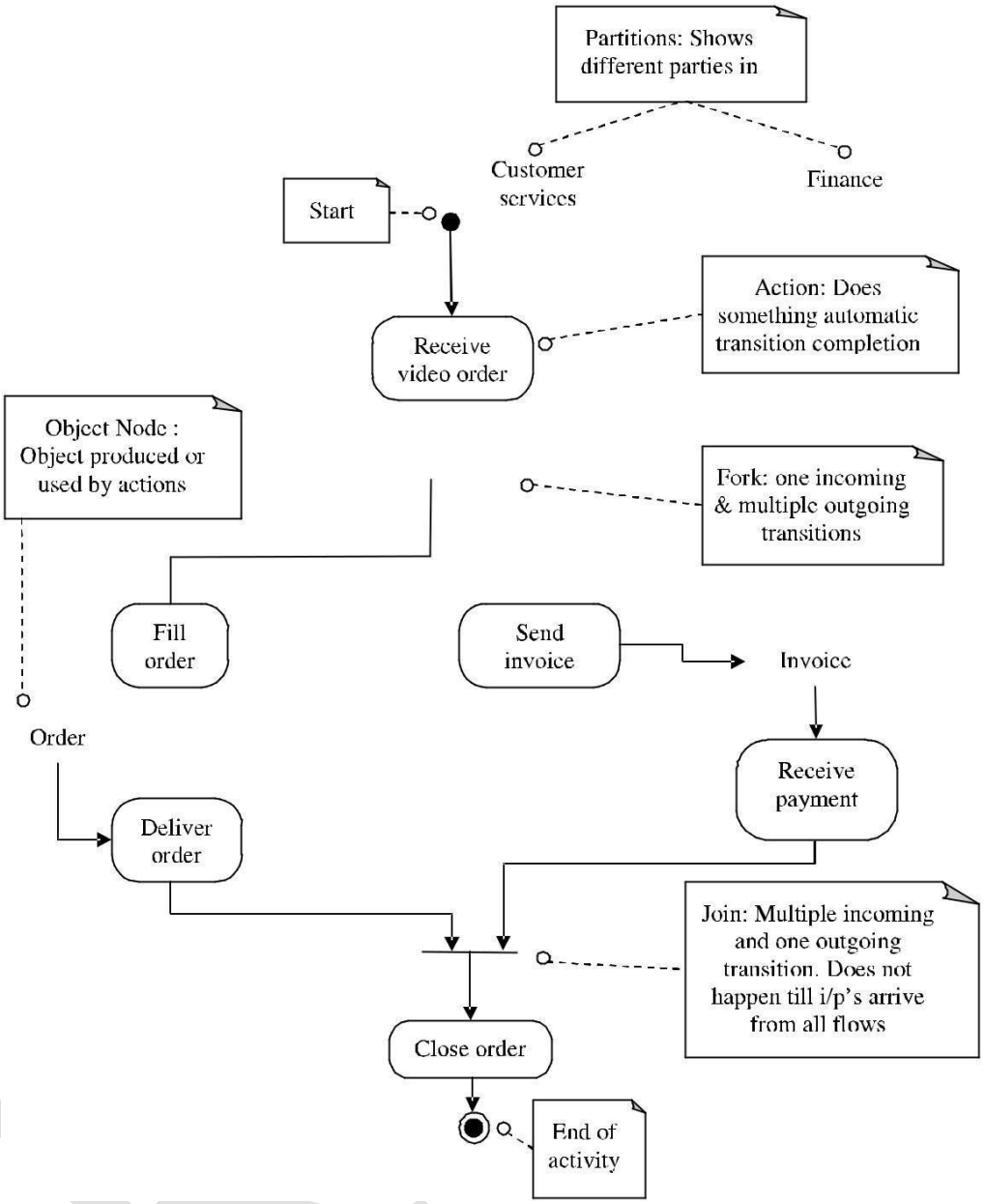


Figure 3.17 Activity Diagram

Data Flow Modeling

Data Flow Diagrams (DFD) are used to visualize the major steps and data involved in software system processes. DFDS were used to document the data flows.

In Gane-Sarson notation of DFD the process steps are numbered to indicate order.

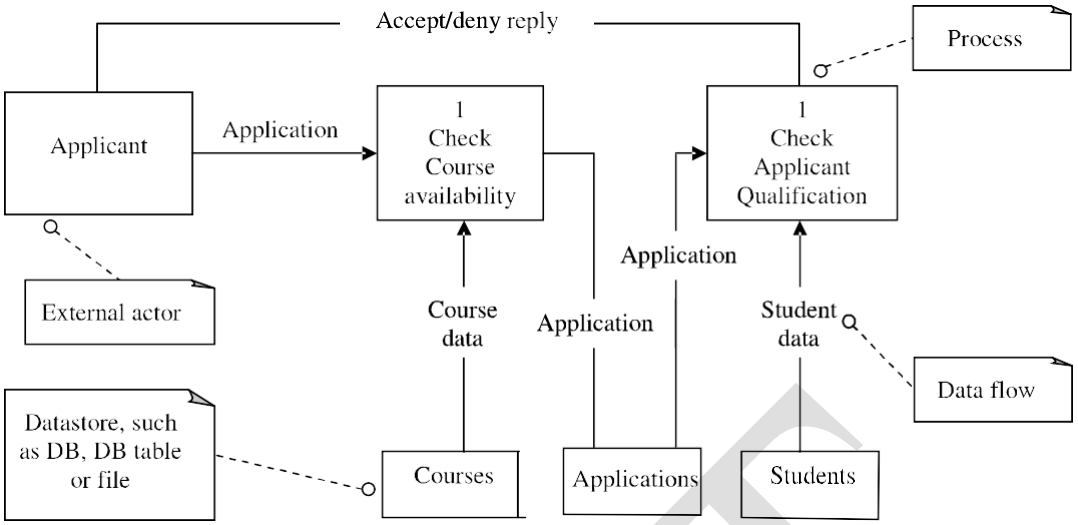


Figure 3.18 DFD (Data Flow Diagrams) in Gane-Sarson notation

UML does not have DFD notations.

The DFD is replaced by activity diagrams in UML.

In addition to the object nodes the UML activity diagram includes „datastore" node for the above example.

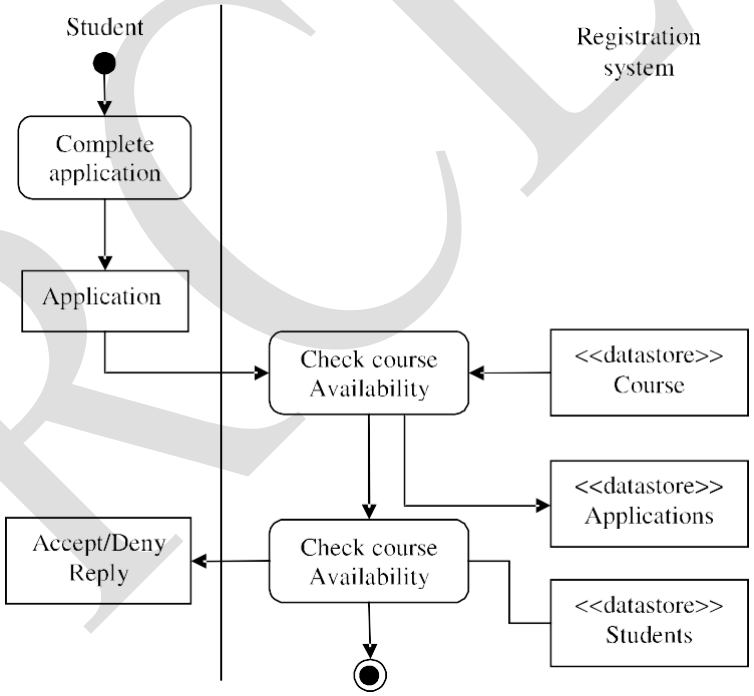


Figure 3.19

Action may be organized into swim lanes each separated from its neighboring swim lanes by vertical solid lines on both sides.

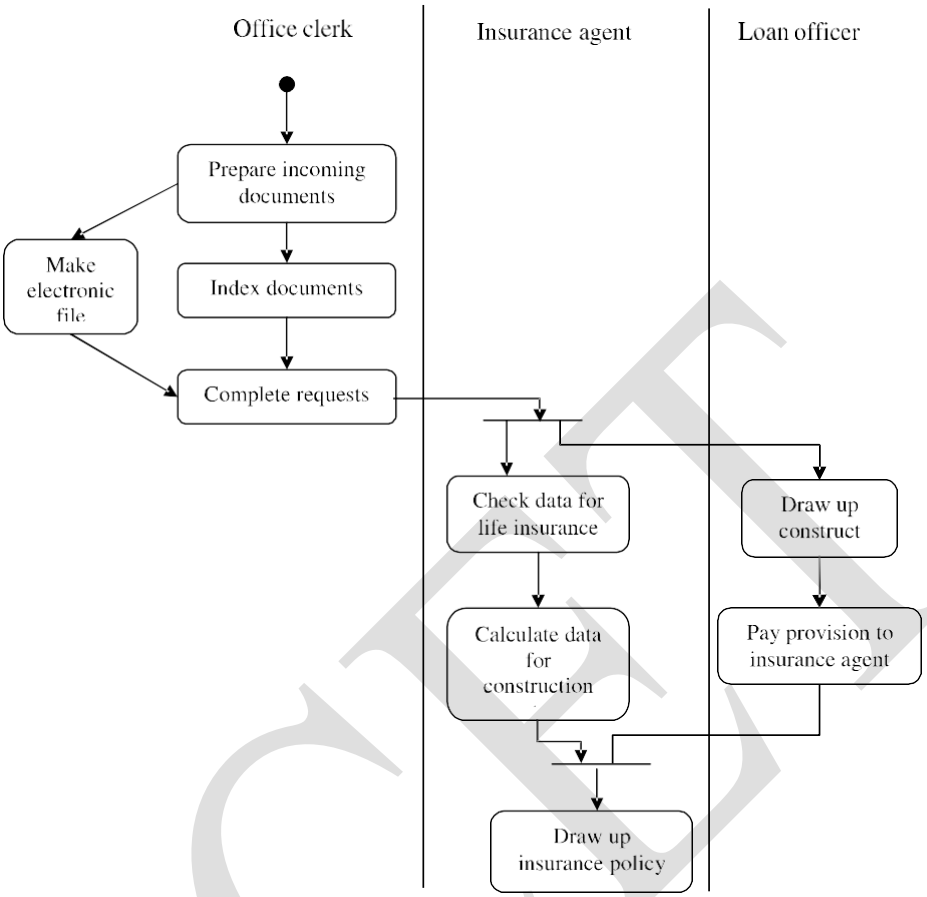


Figure 3.20 Activity diagram for insurance system

3.1.7.1 Guidelines

1) Activity diagrams are used for complex processes.
2) In Business process modeling used „rake" notation and sub activity diagrams.

   Level 0 – very high level of abstraction

   Level 1 – expand the details

   Level 2 – more expansion and so on
3) Levels of abstraction must roughly be equal within a diagram.

Example

The „Process Sale" usecase is shown as an Activity diagram

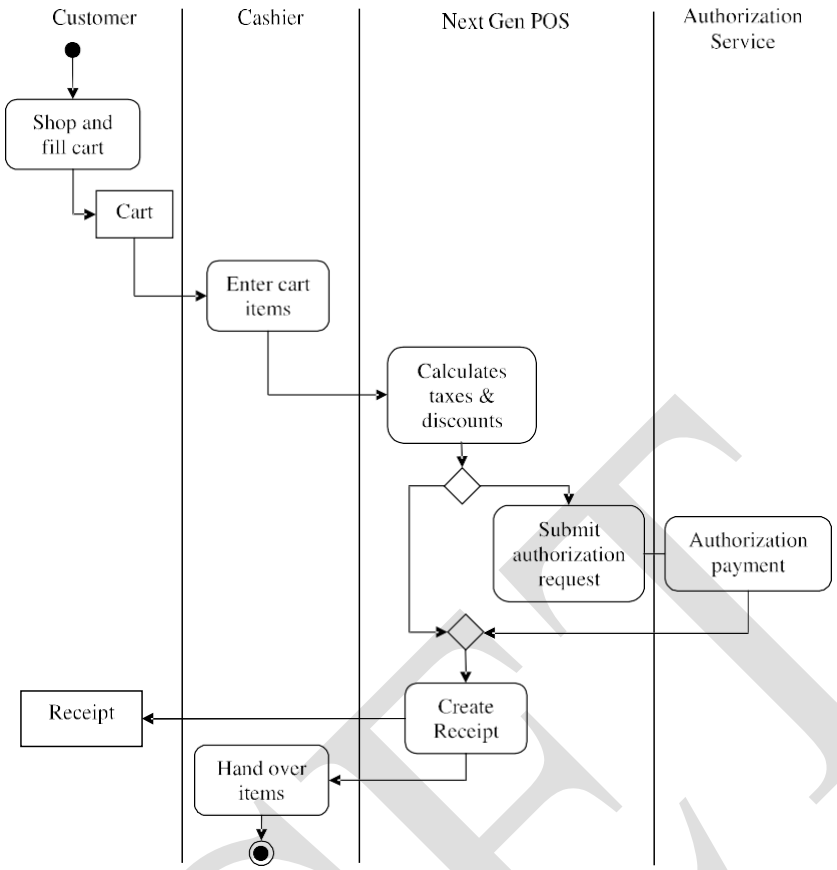


Figure 3.21

Activity diagrams are applicable within the business modeling discipline of UP.

3.1.7.2 Advanced Activity Diagrams

Activity diagrams are useful in connection with workflow and in describing behavior that has a lot parallel processing.

The core symbol is “Activity state” or “Activity”.

Activity is a state of doing something

- Either real world process [Example: type a letter]
- Or execution of software runtime [Example: method on a class]

Activity diagram describes sequencing of activities, with support for conditional and parallel behavior.

Activity diagram is a variant of state diagram in which states are activity states.

Much of the terminology follows state diagrams

Conditional behavior is given by

- Branch
- Merge

(a) Branch

- It has single incoming transition and several guarded outgoing transition.
- Only one of the outgoing transitions can be taken. So guards are mutually exclusive.
- [else] is a guard, „else‟ transition should be used if all other.
- In the example, after an order is filled, there is a branch.
  - If the order is rush, do an overnight delivery.
  - If the order is regular, do a regular delivery.